Systematic Control Logic Fault Identification and Localization of Heating, Ventilation, and Air Conditioning Systems

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philosophy
in
Civil and Environmental Engineering

Xuechen Lei
B.S., Civil Engineering, Southeast University
M.Eng., Architectural and Civil Engineering, Southeast University

Carnegie Mellon University
Pittsburgh, PA
May, 2019

Keywords: FDD, HVAC, control logic, fault definition, fault localization.

Executive Summary

It is estimated that more than 15% of the building heating, ventilation, and air conditioning (HVAC) problems are due to control software programming. The estimated annual energy impact of building HVAC control logic faults in the United States is 12 trillion BTU.

Current industry practice adopts two approaches to deal with HVAC control logic faults: manual logic verification during commissioning and fault detection and diagnosis (FDD) during system operation. Both approaches have limitations that make them ineffective in verifying HVAC control logic. The manual logic verification process requires subjective ad-hoc reasoning, which is costly (knowledge and labor intensive) and error-prone. Existing FDD tools and studies do not provide an approach to specify possible logic faults that might exist, and they rely heavily on user input to diagnose faults effectively. This research targets the problem of HVAC control logic faults by proposing an HVAC control logic fault identification and diagnosis framework following the software unit testing paradigm.

This research specifically focuses on air handling unit (AHU) systems, since control logic faults are more frequently found in them and they are one of the most important and prevalent types of equipment in commercial building central HVAC systems. Two specific challenges this research addresses are: 1) the need for a formalized approach to define control logic faults customized based on system-specific information, 2) the need for a computer-aided approach to help diagnose control logic fault causes in the control logic program effectively.

To address the first challenge, the contributions I made in this research include: 1) developing a formalism of defining applicable AHU control logic faults based on system-specific information, and 2) developing an AHU component and control ontology that specifies the information requirements for defining control logic faults. The generality of the fault definition formalism (including the ontology) and the precision/recall of the faults it defined are validated with 27 different AHUs specified by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). The implemented prototype is able to provide customized control logic fault definition for all 27 AHUs with an average precision of 94.2% and recall of 83.0%.

To address the second challenge, the contributions I made in this research include: 1) developing a framework of casting AHU control logic fault diagnosis problem into a software fault localization task, 2) evaluating the performance of spectrum-based and mutation-based fault localization algorithms for locating AHU control logic fault causes, and 3) identifying effective mutation operators and conducting sensitivity analysis to explore setup options for AHU control logic fault localization computation. The implementation of the developed framework supported the evaluation of 39 considered spectrum-based and mutation-based fault localization algorithms on 11 real-world control logic fault cases I developed from two real-world AHUs. The evaluation showed that the mutation-based Metallaxis method outperformed all other considered algorithms for diagnosing AHU control logic faults.

The outcomes of this research are expected to alleviate the significant energy waste caused by HVAC control logic faults through motivating the industrial deployment of the proposed HVAC control logic fault identification and diagnosis framework for a more effective and systematic control logic verification process. This thesis also points out multiple future research directions, such as HVAC information inference for control logic specification, and HVAC control logic code analysis.

To my family.

Acknowledgments

First and foremost, I am sincerely grateful for the opportunity to work with Professor Burcu Akinci and Professor Mario Bergés, who have served as my advisors, mentors, and friends at Carnegie Mellon University and have been the biggest help during my whole Ph.D. life. I am indebted to them for their great support for all aspects of my professional and personal development.

I would like to thank Professor Xuesong Liu, Professor Claire Le Goues and Dr. Yan Chen for valuable suggestions and feedback during my thesis development and serving as my committee members. This thesis would not have been possible without the support from them.

I want to thank my colleagues, Raghuram Sunnam, Varun Kasireddy, Chiyon Cho, Xuan Li, Yujie Wei, Bingqing Chen, Yasamin Hashemi, Minkyung Kang, Justin Ker-Wei Yeoh, Jingkun Gao, Henning Lange, Jingxiao Liu, Francisco Ralston Fonseca, and Chukwudi Udeani, for their help, company, and encouragement.

I would like to acknowledge the Pennsylvania Infrastructure Technology Alliance (PITA), the Julia and Michael Ellegood Strategic Doctoral Fellowship, and the Mao Yisheng Graduate Fellowship at Carnegie Mellon University, for providing financial supports of my research.

Finally, I would like to thank my parents and grand parents. They are the best family for me for the past 30 years. Since the day I became an adult, my parents have always trusted my decisions and supported me with all their love. They are always by my side to share my joys and help me overcome difficult times. This achievement is theirs as well.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Industry problem

Building heating, ventilation, and air conditioning (HVAC) system control logic faults are implementation errors of the control logic programs causing violations of HVAC control objectives and resulting in inefficient system operation and building service compromises. Ardehali and Smith [4] conducted a review of existing HVAC control case studies to identify how issues related to controls and building energy management and control systems (EMCSs) contribute to inefficient energy consumption. The review identified 384 control-related problems in more than 118 buildings and 67 reported case studies. The reviewed case studies ranged from the 1980s to the 2000s and were mostly conducted in the United States. The review also showed that HVAC control logic programming problems are responsible for the largest portion (31.3%) of all identified problems in these case studies. Besides this quantitative evidence, the significance of HVAC control logic faults was also pointed out by qualitative expert opinions. From the viewpoints of experts (including consultants, facilities engineer, utility company technologist), HVAC control programming problems have the largest occurrence rate as well as the largest energy impact among all control related faults [5][1]. It is estimated that the annual energy impact of building HVAC control software programming faults is 12 trillion BTU [6]. These facts indicate the prevalence and magnitude, in terms of energy waste, of HVAC control logic faults.

Current industry practice adopts two approaches to deal with HVAC control logic faults: (1) manual logic verification [7] during commissioning, and (2) fault detection and diagnosis (FDD) [8] during system operation. Both approaches have limitations. First, the manual logic verification process is instructed by general narrative instructions, and in order to apply the instructions to specific systems under evaluation, commissioners must conduct heuristic interpretation and make subjective judgments on how to apply them to the verification of specific control logic programs. Additionally, during manual logic verification, a given control logic program, which can be highly complex in terms of variable interactions across different parts of the program, needs to be manually interpreted (control logic programmers or control commissioners subjectively read and understand the code from the beginning to the end of the program) to diagnose the causes of faults. This inaccurate manual interpretation of computer code and non-comprehensive identification of all operating scenarios [9, 10] cause the manual logic verification to be costly [11] and error-prone. Secondly, existing FDD tools and studies focus on hardware faults or a combination of mostly hardware and a small number of control logic faults without systematic exploration of all possible control logic faults that might exist. For example, within the 28 defined HVAC faults in the AHU Performance Assessment Rules

[1]Other control related problems include hardware related problems, human factor related problems, and other aspects of software related problems, which are input / output implementation problems, operation problems and data management problems

(APAR) [12, 13], only two faulty control logic behaviors are specified: 1) incorrect cooling mode, 2) simultaneous modulating of mixing box damper, heating coil valve, and cooling coil valve. The definition of these two faults is not a systematic control logic fault definition for AHU systems because, for instance: 1) it is only applicable to systems having a cooling coil, a mixing box and a heating coil; 2) it does not include any faults about single control loop behavior, while there are popular control logic faults with regard to single control loop behavior observed in real-world cases [4]. All these limitations make the current practice ineffective for verifying HVAC control logic.

1.2 Problem abstraction

The general objective of this research is to detect and diagnose HVAC control logic faults in the implementations of logic programs. HVAC control logic is essentially a software program, which acquires inputs and provides outputs to a physical process, as stated in [14]. This is also known as a hybrid system (i.e. cyber-physical system (CPS)). The target of resolving HVAC control logic faults is to verify the correctness (compliance to the operation requirements of HVAC systems) of the control logic software.

Any software implementation can be inspected manually in a heuristic manner [15]. In this approach, software products are reviewed using reading techniques (a series of steps or procedures that guide an inspector in acquiring a deep understanding of the software product), such as ad-hoc based reading [16, 17] and checklist based reading [18, 19], to detect defects [20]. Software systems can also be validated computationally. There are four principal categories of computational approaches that are used for validating complex systems in the software and CPS domains.

These four categories are: simulation (making experiments on an abstracted model of the system), testing (making experiments on the actual product), deductive verification (using of axioms and proof rules to prove the correctness of the system) and model checking [21, 22] (verifying a finite state machine [23] model with formal specifications (e.g. temporal logic [24])). All four categories of techniques are about verifying the systems with specifications that encode the requirements of the systems under evaluation.

One major advantage of testing methods is that testing works directly on the software program source code, the actual system, while the other three techniques all require abstractions of actual systems into models. For example, for model checking, the system needs to be abstracted to a finite state machine, and for deductive verification, a mathematical automata model [25] is needed. Abstracting these models from software source code and using formal methods to verify them requires manual reasoning of a given software program. Thus, these techniques do not avoid the human cognition limitations of manual logic verification, and the system to be verified is an abstracted model rather than the actual software program implemented. On the other hand, model checking and deductive verification techniques are more rigorous than testing methods, in the sense that they prove the correctness of the system models with regards to provided specifications. That said, although testing can never prove correctness, there are strategies to improve the confidence of testing results [26]. More importantly, testing involves less manual ad-hoc computation because, instead of abstracted models, testing directly verifies the actual control logic code, which is the focus object of this research. Thus, as an initial attempt to bring computational software verification into the HVAC domain, this research focuses on resolving HVAC control logic faults with the software testing paradigm.

According to three authoritative publications [27, 28, 29], the generic software testing process can be categorized into 4 core activities:

- Generate program input/output data: related activities of generating the test cases containing the input and output data of the software under test.
- Generate program requirements specification: for most engineering professions, the term "specification" refers to the assignment of numerical values or limits to a product's design goals [30]. Here, it refers to the activity that generates an agent that decides, for each execution of the test, whether the software product behaved in compliance with the requirements.
- Execute the test: related activities of the process that take the test cases, program source code and the specification as the inputs, execute the program with the test cases, and evaluate the program outputs according to the specification.
- Evaluate test results: to resolve software faults, this activity analyzes the outcome of the test execution to diagnose the faults and remove them [27].

My envisioned computational framework of HVAC control logic fault identification and diagnosis focuses on these four activities and the process diagram of this computational framework is shown in Figure 1.1.

1.3 Motivating case study and underlying engineering problems

A motivating case study is carried out on 7 air handling units (AHUs) (all with the same component layout, as shown in Figure 1.2) located in one academic

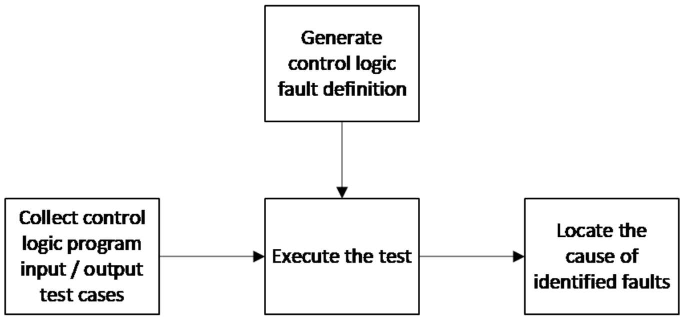
Generate control logic fault definition
Collect control logic program input / output test cases
Execute the test
Locate the cause of identified faults

Figure 1.1: Process diagram of the HVAC control logic fault identification and diagnosis framework

building. In the case study, I manually identified and diagnosed control logic faults by conducting the four activities stated in Figure 1.1.

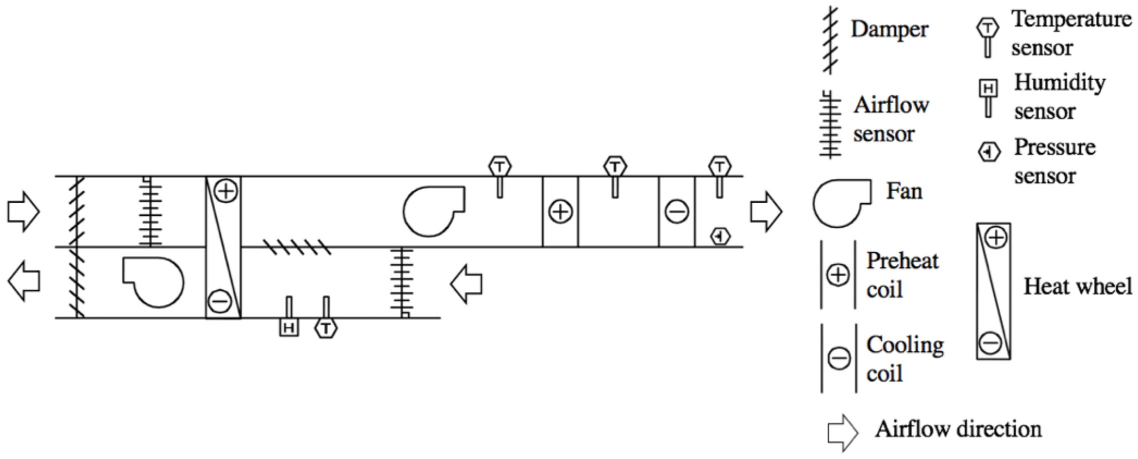

Figure 1.2: System layout of the 7 case study AHUs

1.3.1 Generate program requirements specification

To identify what control logic faults exist in a specific HVAC system, the first step involves detailing what possible control logic faults one should check for that specific system. A review of existing HVAC operation instructions [31, 32, 33, 34, 35, 36, 37, 38] and studies about HVAC control logic faults [4, 12, 13, 39, 40, 41, 42, 43] showed that no existing instruction or study specifies HVAC control logic faults in a systematic manner. Only a few control logic fault definitions were proposed in an ad-hoc way in existing studies.

Existing HVAC operation instructions have two issues. First, they use narrative language to describe either how HVAC systems should be operating or what conditions they should avoid, without detailed instructions on checking actual control loops and control input/output variable values. For example: "For units with economizer control, confirm that the outdoor air damper returns to the minimum position at the appropriate temperature (enthalpy) level" [33]. This instruction item does not specify: a) what variables in the control logic should be checked for the outdoor air damper position, b) what "the minimum position" of the outdoor air damper for the specific system is, and c) what "the appropriate temperature (enthalpy) level" is and how one should check that in the specific system. Second, different systems with different components and functions would have different applicable control logic faults, while existing instructions are static narrative documents that are not tailored to specific system information. For example, ASHRAE handbook – Applications [31] uses plots to show AHU supply air temperature control sequencing and implicitly assumes the unit has a modulating preheat coil, a cooling coil and a mixing box with economizer control. Thus, the instruction is not applicable to systems without economizer control.

In terms of existing studies about HVAC control logic faults, Ardehali and Smith [4] summarized the HVAC operation problems caused by programming problems into 11 types with items like "Improper sequencing" and "Improper control logic". Dexter and Pakanen [39] provided a list of 8 types of controller software faults with items like "Improper control action" and "Incorrect flags". The majority of HVAC fault detection and diagnosis (FDD) studies focus on hardware faults, while several existing studies [13, 40, 41, 42, 43] discussed a combination of both hardware and software faults in the systems [31]. These reviewed studies adopted expert rules to detect control logic faults, together with other hardware faults. None of them developed a non-heuristic strategy to specify what possible control logic faults exist in a given system. As shown in the example discussed in Section 1.1, two control logic faults were defined in APAR [12, 13]: 1) incorrect cooling mode, 2) simultaneous modulating of more than one of the three devices: mixing box dampers, heating coil valve and cooling coil valve. However, this study did not provide a methodology of defining these faults, and these two faults are only applicable to AHUs with preheat coil, cooling coil and mixing box.

Having specifications that unequivocally depict the requirements to control behavior is essential to the testing of control logic programs. However, from existing publications, it was not possible to identify a set of control logic program requirements specification that can be applied to the case study systems without ad-hoc interpretations and reasoning. This is the first challenge I propose to address.

To be able to conduct this case study, after reviewing the aforementioned instructions and studies, I defined a set of control logic fault types to be applicable to the case study, as shown in Table 1.1. These control logic faults indicate implemented control logic violating two general objectives I summarized from existing publications: 1) violating energy efficiency objective for sequencing of different components, or 2) causing a component's capacity to be not fully and correctly utilized (discussion of objectives is detailed in Chapter 2). The defined faults are instances of violations of these two general control objectives with regard to the specific information of the case study AHU systems. Each type of control logic fault is defined to have a symptom in terms of unambiguous mathematical expressions of control logic input/output variable value relationships. A set of 16 faults are defined in an ad-hoc manner for the use of this case study, by manually and heuristically iterating over the components and brainstorming the possible faults.

Fault	Type classification	Fault description
1	Building service - Temp - Cooling	Supply air temperature continuously higher than supply air temperature set point, but cooling coil valve command is less than 100%.
2	Building service - Temp - Cooling	Supply air temperature continuously lower than supply air temperature set point, but cooling coil valve command is greater than 0%.
3	Building service - Temp - Heating	Supply air temperature continuously lower than supply air temperature set point, but preheat coil valve command is less than 100%.
4	Building service - Temp - Heating	Supply air temperature continuously higher than supply air temperature set point, but preheat coil valve command is greater than 0%.
5	Building service - supply air pressure	Supply air static pressure continuously lower than supply air static pressure set point when supply fan command is less than 100%.
6	Building service - supply air pressure	Supply air static pressure continuously higher than supply air static pressure set point when supply fan command is greater than 0%.
7	Building service - Air quality	Outdoor air damper command is less than minimum out door air damper set point.
8	Building service - Air quality	Outdoor air cfm continuously lower than outdoor air cfm set point, but outdoor air damper command is less than 100%.
9	Building service - Air quality	Return air CO_2 level continuously higher than return air CO_2 level set point, but outdoor air damper command is less than 100%.
10	Energy eff. - simultaneous heating & cooling	Preheat coil valve command and cooling coil valve command greater than 0% simultaneously.
11	Energy efficiency - Economizer	Outdoor air temperature lower than supply air temperature set point and the outdoor air damper command is less than 100%, but the cooling coil valve command is greater than 0%.
12	Energy efficiency - Economizer	Outdoor air temperature lower than supply air temperature set point and the outdoor air damper command is larger than needed to satisfy indoor air quality requirement, but the preheat coil valve command is greater than 0%.
13	Energy efficiency - Enthalpy (Heat) wheel - Heating	Outdoor air enthalpy (temperature) lower than return air enthalpy (temperature) and the enthalpy (heat) wheel command is less than 100%, but the preheat coil valve command is greater than 0%.
14	Energy efficiency - Enthalpy (Heat) wheel - Heating	Outdoor air enthalpy (temperature) lower than return air enthalpy (temperature) and the enthalpy (heat) wheel command is greater than 0%, but the cooling coil valve is greater than 0%.
15	Energy efficiency - Enthalpy (Heat) wheel - Cooling	Outdoor air enthalpy (temperature) higher than return air enthalpy (temperature) and the enthalpy (heat) wheel command is less than 100%, but the cooling coil valve command is greater than 0%.
16	Energy efficiency - Enthalpy (Heat) wheel - Cooling	Outdoor air enthalpy (temperature) higher than return air enthalpy (temperature) and the enthalpy (heat) wheel command is greater than 0%, but the preheat coil valve command is greater than 0%.

Table 1.1: Control logic fault definition for the case study AHUs

This heuristic fault definition activity is challenging due to two facts: first, it requires ad-hoc interpretation of ambiguous operational requirements and, second, it also requires subjective identification of how to apply the interpreted requirements to a specific HVAC system's information (component layout and attributes of controllers). This challenge causes the subjectively generated fault definition to be more likely to contain errors and omissions. To overcome this challenge, I envision a formalized approach that utilizes a system that mimics the reasoning procedures used by people to derive control logic fault definition from general control objectives, but avoids the human cognition barriers. This work is detailed in Chapter 2 and is a major component of this research.

1.3.2 Generate program input/output data

One essential activity in the testing process is to generate program input/output data to be used as test cases. The input/output dataset of the control logic program depicts the control behavior of the HVAC system and contains the symptoms of control logic faults that exist in the control logic program implementation of the system.

If the HVAC system under evaluation has a building automation system (BAS) historical dataset containing control logic input/output variable data points, this dataset can be used to provide test cases for the test execution.

On the other hand, if the system has not been put in operation, multiple testing techniques (e.g. structural testing, combinatorial testing and random testing) can be used for test case generation [26]. For example, dynamic symbolic execution (DSE) [44], a recently proposed testing technique, analyzes a software source code to provide a test suite that is able to exercise all possible execution paths of the program and provide full code path coverage. Majumdar et al. [45] also applied DSE in generating test case for control applications. Compared with previous simplified test case generation techniques, such as random testing, with this full path coverage, test cases provided by DSE can most likely trigger the symptoms of program defects by exercising all possible execution paths of a program.

In this case study, I was able to acquire a test case set by leveraging historical BAS operational dataset and by applying DSE, without any major challenges.

1.3.3 Execute the test

In the test execution activity, I implemented a rule-based system that checks the test cases with assertions derived from the fault definition described in Section 1.3.1. The faults are defined as control input/output variable expressions and the test cases are a set of value instances of these variables. The test execution activity assigns a tag of faulty / not faulty (i.e. fail/pass) to each test case instance. This activity identifies the existence of defined faults in the actual implementation of the control logic programs by examining the test cases with regard to the fault definition.

1.3.4 Evaluate test results

After identifying the existence of specific control logic faults of a system under test, the next step is to find the causes of the identified faults. This step is referred to as fault diagnosis [8] or fault isolation [46] within the HVAC FDD domain. It refers to the evaluation of the existing faults and determining their causes [8]. In the software engineering domain, this step is referred to as fault localization, meaning the act of "identifying the locations of faults in a program" [2].

Multiple studies have pointed out the challenges of fault diagnosis in HVAC systems: despite focusing on hardware faults or a combination of hardware and software faults, most FDD tools developed to date focus on fault detection and still rely heavily on user input to diagnose the fault effectively [11]. FDD methods yielded a number of possible causes for a specific fault, and more work is needed to ascertain the cause of the faults [47]. In many cases, it may only be possible to detect rather than diagnose faults. Few, if any, FDD methods can make a conclusive diagnosis for the cause of a detected fault [48]. Current data visualization programs can help users to detect and diagnose faults on AHUs, but a large amount of time can be spent to ascertaining the causes of the possible issues [11]. In general, when an HVAC control logic fault is observed from the control output variables, existing approaches do not provide effective instructions to locate the fault causes inside the programs.

In this case study, after identifying the existence of control logic faults in the test execution activity, I proceeded to manually inspect the control logic code with the help of test cases to find the causes of the identified faults in test bed AHUs. The values of variable in test cases were used to instruct the understanding of what statements were executed during the corresponding test executions.

For example, the most prevalent control logic fault identified in one test bed AHU is "heat wheel heating not maximized before preheat coil heating starts". The symptoms of this fault are:

- Outdoor air temperature is lower than return air temperature
- Heat wheel command is less than 100%
- Preheat coil valve command is greater than 0%

After iteratively checking BAS data and subjectively interpreting the complex control logic code (more than 250 lines of graphical program (lines of the graphical program are designated by the control manufacture's software) with interacting control loops and variables), the cause of this fault is located to be an inappropriate heat wheel heating control loop set point being 50 °F instead of 55 °F (the supply air temperature set point). To get HVAC control logic fault causes, manually reading complex programs is ineffective. For example, for this one fault, several iterations of ad-hoc complex program interpretations are needed and subjective reasoning of the fault cause is prone to mistakes.

1.3.5 Lessons learned and underlying engineering problems

Engineering problem 1

In order to verify the correctness of control logic programs by testing, a set of specifications defining the requirements of the programs in terms of program input and output variable expressions need to be provided. General guidelines and existing studies provide "rules of thumb" instructions, such as "For units with economizer control, confirm that the outdoor air damper returns to the minimum position at the appropriate temperature (enthalpy) level" [33], and "Improper control logic" [4]. These instructions do not discuss specific symptoms of the faults or requirements in terms of the program's input and output variables. Thus, in order to implement these instructions into specifications compatible to the testing process, people who evaluate HVAC systems have to subjectively adapt these instructions with their own interpretations and brainstorm what the applicable types of faults are and how to apply instructions to the specific system configurations under evaluation in an ad-hoc manner. This non-systematic process of heuristic specification elicitation from existing instructions is error-prone and incomprehensive. Thus, there is a need for a formalized approach to define control logic faults without the involvement of heuristic reasoning. This formalized approach needs to provide control logic fault definition in terms of program input/output variable expressions to enable the software testing based HVAC control logic fault identification and diagnosis framework shown in Figure 1. For instance, the fault described in the example shown in Section 1.3.4 shall be defined as "OA_T < RA_T & EW_VFD < 20 & pht_vlv > 0" (variable names and specific numerical values in the expressions are actuators' control related information of the case study AHU). This definition is computer processable without heuristics. Defining the fault in this manner makes it unambiguous and directly adoptable by the fault detection process.

Engineering problem 2

After observing the existence of control logic faults in the test execution, manual diagnosis of the control logic fault causes requires interpretation of complex control logic programs and reasoning of logic execution information from program input/output data. For instance, the diagnosis example presented in Section 1.3.4 involves a subjective interpretation and reasoning of more than 250 lines of graphical control logic program containing interacting control loops and variables, with clues provided by executed test cases. Manual ad-hoc reasoning of such complex code is error-prone and time-consuming due to human cognition barriers. For example, the reasoning of one specific fault in this aforementioned example cost several iterations of reading of the source code with different input/output variable samples to understand the logic of the program before locating the cause of the fault to be a conflicted set point. There is a need to have a computer-aided approach that is able to locate the cause of the identified control logic fault inside the program.

1.4 Research vision and objectives

I envision a modular framework for identifying and diagnosing HVAC control logic faults based on the four core activities of software testing perspective shown in Figure 1.1. Figure 1.3 depicts an IDEF0 diagram of the envisioned framework. This computational framework can be applied by a control logic programmer during control implementation, or by an HVAC commissioner for field testing [33] during (retro-/re-) commissioning.

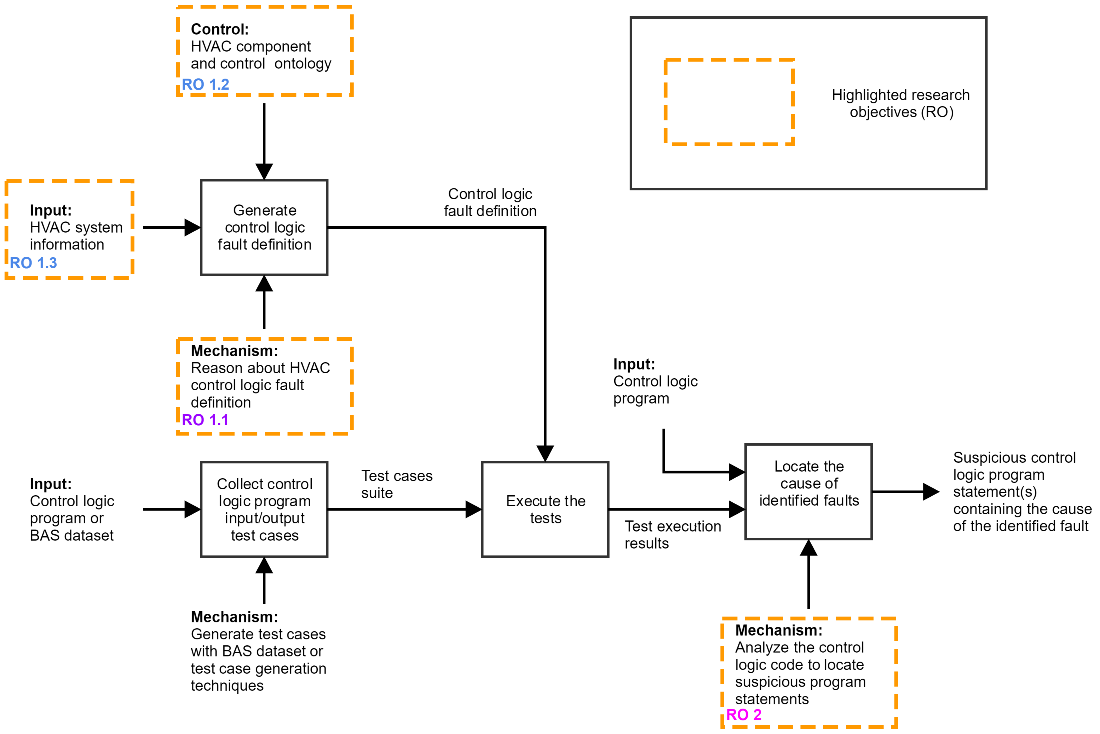
Control:
HVAC component and control ontology
RO 1.2
Highlighted research objectives (RO)
Input:
HVAC system information
RO 1.3
Generate control logic fault definition
Control logic fault definition
Mechanism:
Reason about HVAC control logic fault definition
RO 1.1
Input:
Control logic program
Input:
Control logic program or BAS dataset
Collect control logic program input/output test cases
Test cases suite
Execute the tests
Test execution results
Locate the cause of identified faults
Suspicious control logic program statement(s) containing the cause of the identified fault
Mechanism:
Generate test cases with BAS dataset or test case generation techniques
Mechanism:
Analyze the control logic code to locate suspicious program statements
RO 2

Figure 1.3: Envisioned HVAC control logic fault identification and diagnosis framework

In Figure 1.3, the research objectives are highlighted with dashed bounding boxes.

- Research objective 1 (RO1): Propose a formalized approach that is able to generate control logic fault definition applicable to specific system information. It has the following three sub-objectives:
  - Research objective 1.1 (RO1.1): develop a reasoning mechanism to generate customized HVAC control logic fault definition based on system control objectives, such as the energy efficiency objective discussed in the case study (discussion of these objectives are detalied in Chapter 2).

  - Research objective 1.2 (RO1.2): identify, organize and represent HVAC component and control information in an extensible manner to support generation of customized HVAC control logic fault definitions.
  - Research objective 1.3 (RO1.3): identify the sources of the information to be specified in relation to research objective 1.2 from existing building information model (BIM) schemas.

- Research objective 2 (RO2): identify a fault localization approach that effectively locates the cause of control logic faults inside the control logic program. Here the effectiveness is measured by the amount of code statements in the control logic program that is needed to be manually inspected by the program inspector to reach the actual root cause of the fault. This metric is discussed in more detail in Chapter 3.

1.5 Overall scope and assumptions

In this section, I list the scope and assumptions of research work in this thesis.

1.5.1 Scope

This research has the following scope:

- While the envisioned approach may be applied to different types of HVAC systems, this research only focuses on AHU systems. AHUs are the predominant carriers of control logic faults [4]. They are also one of the most important and prevalent types of equipment in commercial building central HVAC systems. Hence by focusing on AHUs, it is expected that the impact of this research to the industry problem will be more significant compared with focusing on other types of equipment.

- Though this research only considers control objectives, AHU components and corresponding functionalities observed from existing documentations and from test bed systems during approach development of RO1, the intention is to have an extensible approach that is able to encompass new control objectives, components, and functionalities in the future if needed.

- This research only focuses on control logic faults during normal operation mode[2] of the systems violating general control objectives.

- This research does not focus on how to properly set the service level of AHU systems, i.e. this research does not intend to address questions such as "What is the appropriate supply air set point and reset strategy for the AHU?" or "How to properly decide the scheduling (set back) of the AHU?"

1.5.2 Assumptions

The research work is conducted with the following assumptions:

- The general control objectives for operating AHUs do not change among different AHUs. For example, for the energy efficiency general objective, in an AHU cooling scenario, economizer cooling is always preferred over cooling coil based cooling because it is more energy efficient.

- The control logic programs source code is available.

- AHUs are equipped with EMCSs, and BAS data access containing all control logic input / output variables is available.

[2]The system is actively providing services to building spaces and occupants in a normal condition, i.e. not in an abnormal or transitioning status, such as unoccupied or warm-up modes

1.6 Research questions

This section discusses the two research questions targeting the two research objectives (in Section 1.4) that will be addressed in this thesis.

1.6.1 Research question 1

- RQ1: What is a formalized process of defining control logic faults in terms of program input / output variable expressions that is general to be applicable to AHUs with different configurations?

The first research question aims to develop an approach that formalizes the heuristic control logic fault definition process conducted in the motivating case study to generate program requirements specification for AHUs (Section 1.3.1).

Given the magnitude of the control logic fault problem, the faults need to be identified systematically. The first step towards identifying the control logic faults is to systematically define the faults that need to be checked. As discussed earlier, previous studies in this area do not include a non-heuristic strategy to specify what are the applicable control logic faults for an AHU system. Through research question 1, I intend to address this gap by developing a formalism that contains reasoning mechanisms for defining control logic faults and a corresponding HVAC component and control information ontology. This research builds on existing studies of classification system [49] development methodology and HVAC product modeling (The selection of these approaches is detailed in Chapter 2). To the best of my knowledge, this is the first attempt to adopt a formal classification approach for the purpose of defining HVAC system control logic requirement specifications.

Control logic faults applied to a given AHU system vary since different systems have different components and control information. Thus, the formalism needs to be general enough to be applicable to AHUs with different information, and it needs to output customized control logic fault definitions accordingly. It is not possible to consider all possible AHU general control objectives and components once and for all, especially with the advancement of technologies leading to new hardware inventions and new strategies of system control. Thus, the proposed approach also needs to be extensible to include new objectives and components without needing to modify existing system implementation.

1.6.2 Research question 2

- RQ2: What is a fault localization algorithm that is able to effectively identify the location of fault causes in the control logic program of AHUs?

In the motivating case study described in Section 1.3.4, in order to locate the cause of the identified fault of "heat wheel heating not maximized before preheat coil heating starts", I have to iteratively read the whole control logic program from the beginning to the end because the control logic of different components interact with each other through variable calls and value assignments. This research question aims to identify a fault localization algorithm to help this control logic fault diagnosis process by identifying the suspicious piece of code that contains the cause of the identified control logic fault.

The performance of the fault localization algorithm is measured by effectiveness, which is a metric adopted in the fault localization research domain [2]. Fault localization techniques provide suggestions of suspicious code statements containing the actual fault causes. Program inspectors follow the suggestions to manually check the suspiciousness statements. The effectiveness metric indicates the amount of code needed to be manually checked in the program until identifying the actual fault cause. The lower the effectiveness value is, the less manual efforts are needed to locate the actual fault cause.

To address this research question, I first review existing fault localization techniques from the software engineering domain, with selected ones being implemented and evaluated on diagnosing the control logic faults of real world test bed AHU logic programs. I also identify semantic characteristics of the control logic faults that impacts fault localization effectiveness. I suggest a set of mutation operators to support Metallaxis fault localization for AHU control logic fault based on categorization of actual control logic fault fixes in terms of mutations. Additionally, I explore multiple aspects of fault localization task design options, based on which the guidance of fault localization computation settings is proposed for HVAC control logic fault localization.

1.7 Validation plan

This section discusses the plan of validation the research outcomes from addressing the two research questions described in Section 1.6.

1.7.1 Control logic fault definition formalism validation of generality

A software prototype is developed to implement the fault definition formalism proposed in Chapter 2. The generality of the prototype is validated because it is essential for the formalism to be applicable to AHU systems with different configurations.

Generality refers to the formalism's ability to accommodate different AHUs (AHU components and functions are already covered by the developed ontology). The ASHRAE publication "Sequence of Operation for Common HVAC Systems"

[1] provided examples of 27 different AHUs. The ontology instance inputs to the developed prototype corresponding to these 27 examples will be manually generated and used for validating the generality of the developed formalism.

1.7.2 Control logic fault localization algorithms evaluation of effectiveness

Multiple fault localization algorithms are applied to a selected set of control logic fault diagnosis cases to evaluate the performance of these algorithms and identify the winning algorithm for being utilized in AHU control logic fault diagnosis.

I have not identified any real-world AHU control logic program with its configuration information and BAS data that is publicly available to be used as a test bed in this evaluation. Thus, two distinguishable test bed AHUs from different manufactures, with different configurations and service requirements are developed to be used to evaluate the effectiveness of the fault localization algorithms considered in addressing research question 2. As will be shown in Chapter 3, a total of 11 real-world control logic fault cases are developed based on these two test bed AHUs. With 11 control logic fault cases, my target is not to claim the generalizability of the acquired fault localization performance with statistical significance. Instead, the objective is to support the claim of fault localization effectiveness with:

- Demonstration of the diversity of the 11 fault cases, in terms of the differences of the AHUs they come from, the differences of their fault symptoms, and the differences of the semantic characteristics of the fault causes.
- Observations of the relationships between the fault cause characteristics and the fault localization effectiveness, and the reasoning about what fault cause characteristics lead to good/bad fault localization results.

1.8 Dissertation organization

This dissertation consists of five chapters including Introduction and Conclusions. Chapter 1 provides an overview of the research problems, the motivating case study to identify the engineering challenges, the research vision and objectives, and the research questions addressed in this dissertation. Chapter 5 highlights the research contributions made from addressing the research questions, practical implications and future research directions. Chapter 2 and Chapter 3 focus on research work of addressing each of the two research questions, and are each organized as a self-contained extended journal paper draft. Chapter 4 contains a detailed and complete real-world use case of the proposed AHU control logic fault identification and localization framework, showcasing applications and values of the research work presented in this dissertation.

Chapter 2

Formalized Control Logic Fault Definition of Air Handling Units with Ontological Reasoning

Control logic programs determine the behavior of Heating, Ventilation, and Air Conditioning (HVAC) systems under different operating conditions. Control logic faults are inconsistencies in the program implementations with regard to the design intent and control objectives, and they account for more than 15% of all HVAC system problems, causing energy waste and occupancy discomfort. The first step towards systematically detecting and diagnosing control logic faults is to have an unambiguous control logic fault definition. Since different HVAC systems have different component information, control logic faults applicable to different systems vary. In this chapter, we propose an object-oriented classification approach to systematically define customized control logic faults in terms of control logic input/output variable expressions. We specifically focus on the implementation of this approach to air han-

dling units (AHUs), since they are core devices of most central HVAC systems and are prone to have control logic faults due to their complex multi-component operation nature. In developing the formal object-oriented approach, we elaborated four control goals from the general objectives of energy efficiency and occupancy comfort, and developed corresponding reasoning mechanisms to derive fault definitions. To be used in the reasoning mechanisms, we also developed an HVAC component and control information ontology by extending existing HVAC information models. To validate, we implemented the developed approach in a prototype system. The prototype was tested with 27 common AHUs specified by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), and the results show that using the developed approach, it is possible to define a customized set of control logic faults applicable to each specific AHU with an average precision of 94.2% and average recall of 83.0%. This demonstrates the generality of our proposed approach in providing customized control logic fault definitions for different types of AHUs.

2.1 Introduction

HVAC Control logic programs are software programs implemented in direct digital controllers (DDC) to define the operating behavior of HVAC hardware systems [38]. The existence of control logic faults compromises the general HVAC system operation objectives of energy efficiency and occupancy comfort, and can even cause equipment damage. More than 15% of building HVAC system problems are attributable to control logic faults [50, 4], and they are estimated to be responsible for around 12 trillion BTU of energy wasted each year in the United States alone [6].

Air handling unit (AHU) systems are core devices of most central HVAC systems.

Since AHUs contain multiple components working together to regulate properties of air, such as temperature, pressure, and air quality, their complex multi-component operation makes them prone to have control logic faults. For example, Figure 1.2 shows the layout of an AHU in a university campus building. Just for supply air temperature control, at least four components are involved: the heat wheel, the mixing box with three dampers in coordination, the preheat coil and the cooling coil, because they all can heat/cool the supply air stream. Each of these components is controlled by a dedicated local controller and has different operating conditions and efficiencies. For instance, preheat coil and heat wheel can both heat the supply air stream. While preheat coil can always heat the air, heat wheel can only heat the supply air when the return air temperature (or enthalpy, depends on the designed functionality of the heat wheel) is higher than the outdoor air temperature (or enthalpy), and it does it with a higher efficiency compared with preheat coil. As another example, the mixing box can change both supply air temperature and the supply air quality at the same time by controlling its dampers. Trying to fulfill multiple air property requirements simultaneously with an optimal efficiency makes the multi-component coordination complex and error-prone.

In current industry practice, control logic programs are manually inspected and tested together with the installed hardware systems in functional tests during commissioning. This process is instructed by narrative guidelines in different forms, such as standards, handbooks, and checklists. Existing guidelines of this process [31, 32, 33, 34, 35, 36, 37] provide general instructions of sequencing strategies of components. For example, in ASHRAE Guideline 11 – Field Testing of HVAC Controls Components [33], one instruction item about air temperature control sequencing is "For units with economizer control, confirm that the outdoor air damper returns to the minimum position at the appropriate temperature (enthalpy) level". Instructions like this are not customized for specific system information and use ambiguous narrative language such as "appropriate". In contrast, for the example AHU shown in Figure 1.2, the customization of this instruction could be "if OAT $>$ RAT, then OAD $=$ 20%", assuming in the control logic: OAT represents outdoor air temperature, RAT represents return air temperature, OAD represents the outdoor air damper command, and the minimum opening position for outdoor air damper is 20%. Since different HVAC systems are unique in terms of their components, layout, functionality and service requirements, things to inspect within the control logic programs are different. Using existing guidelines, in order to verify a specific system under evaluation, commissioners must conduct heuristic interpretation of the narrative instructions in the guidelines and make subjective judgments on how to apply them during verification of a specific control logic program. As a result, this non-systematic control logic fault identification process is highly subjective and error-prone.

Thus, there is a need for a formalized approach that provides HVAC control logic fault definition that is customized according to specific system information. This chapter describes an approach that is developed to address this need.

Having customized fault definition for specific HVAC systems will not only benefit the current practice of HVAC manual verification, but also facilitate HVAC control logic fault detection and diagnosis (FDD) through computational verification techniques [51], such as testing, simulation and formal verification. In order to utilize these verification techniques, a set of clearly defined quantitative requirements for the software to be verified is needed. These verification techniques have been applied to software artifacts of systems, such as chemical process control [52], automotive body assembly line [53], and rotating-turbine machinery [54].

In this research, we aim at developing an formalized approach of deriving control logic fault definition through ontological reasoning of information about specific AHU system. An overview of our approach is provided in the next section, after which the rest of the chapter is organized as follows. In Section 2.3 we provide a review of four domains relevant to the research work, namely requirement definition approaches, HVAC control logic specifications, HVAC FDD approaches, and HVAC information models. Then, the chapter describes the AHU control goals used in control logic fault definition derivation. Section 2.5 provides detailed discussions about the techniques selection, design and development of our proposed approach, after which we discuss the prototype implementation and validation. Then we discuss the application boundaries of the proposed research. Finally we conclude the chapter and provide an outlook on future work.

2.2 Approach Overview

Several existing industry guidelines [1, 34] provide example control specifications for specific HVAC systems. In each of these examples, a major part of the specification focuses on detailing control behavior rules during normal operation. These rules are designed to achieve the high-level objectives of energy-efficiency and occupancy comfort. In order to specify these rules for a specific HVAC system, certain information about this system needs to be considered. Observed from these examples and sequence of operations from real-world HVAC system test beds, we summarize the considered system-specific information into three categories, based on which an HVAC ontology specifying the information requirements of deriving control logic fault definition will be developed later in this chapter:

- System component information: this specifies information about a system's physical components, such as what components are available and what functions each of the components can perform.

- System control information: this specifies the control loop information of each component, such as what the output command variables should be and their saturation ranges (i.e., the output command's maximum and minimum boundary values).

- System service requirements: this specifies the desired control targets of the HVAC system, such as what supply air temperature set point should be.

In this research, our target is to formalize the process of deriving unambiguous HVAC control logic fault definition with system-specific information. To do so, we focus on developing a formalism that contains an HVAC ontology mentioned above and an inference engine that encodes and executes the reasoning mechanism of deriving customized control logic fault definition based on high-level energy efficiency and occupancy comfort objectives by reasoning about the system-specific information provided by the ontology instances. The control objectives and the formalism approach will be discussed in detail in later sections.

To eliminate ambiguity, the desired output of our formalism is a set of control logic fault definition, in which, each derived control logic fault is defined by its symptoms in terms of control logic input/output variable expressions. For example, "simultaneous heating and cooling" is a typical control logic fault in AHU systems [4], meaning that the control logic instructs the preheat coil valve and cooling coil valve to open at the same time. This fault will be defined as "PHO $>$ 0 & CCO

>0", assuming "PHO" and "CCO" are the control logic output variables for the preheat coil and cooling coil valve opening percentages, respectively.

2.3 Background and related work

In the following three subsections, we first justify our decision of adopting ontological reasoning as the technical approach in this research by discussing related work about requirement definition. We then briefly discuss existing literature about HVAC control logic specifications, after which we discuss existing FDD studies related control logic fault definition, finally we discuss existing HVAC information models that the development of our HVAC ontology is based on.

2.3.1 Requirement definition studies

In this subsection, we justify our decision of adopting ontological reasoning as the technical approach for formalizing the customized control logic fault definition.

From the software requirement engineering perspective, our target of deriving customized control logic fault definition from high-level control goals is to do goal-oriented requirement elicitation, i.e. refining goals to acquire sub-goals / detailed requirements [55]. Goal refinement is the core activity in goal-oriented requirement engineering (GORE) and various approaches are proposed for this activity [56]. In popular GORE methods, such as KAOS [57], GBRAM [58], and GRL [59], the essentials of discovering sub-goals are through heuristic thinking. For example, the KAOS method focuses mainly on transforming high-level goals into concrete system requirements [60]. In KAOS, the strategy of identifying sub-goals from their parent goal is by "asking HOW questions" [61]. These also echo software requirement elicitation techniques, which typically are directed by strategies of heuristic activities such as brainstorming, interview, survey, and document analysis [62].

Although the elicitation of refined sub-goals are through heuristic thinking, there exists formalized approaches to support goal refinement process. Darimont and Lamsweerde [61] proposed the use of generic refinement patterns to specify the completeness of refined goals. This approach is also leveraged in the requirement refinement of cyber-physical systems (CPS) by Bueres et al. [63]. Bueres et al. [63] proposed the invariant refinement method, which is a systematic and gradual refinement of higher-level invariant by means of its decomposition into a conjunction or disjunction of lower-level sub-invariants. The authors formally defined that decomposition of a parent invariant I_p into a conjunction of sub-invariants $I_{s1}, ..., I_{sn}$ is a refinement if the conjunction of the sub-invariants entails the parent invariant, i.e., if it holds for the patterns of entailment ($I_{s1} \wedge ... \wedge I_{sn} \Rightarrow I_p$) and consistency ($I_{s1} \wedge ... \wedge I_{sn} \nRightarrow false$). As will be shown later, the control logic fault definition derivation approach we proposed in this study can be viewed as the refinement of high-level control goals into low-level requirements, each specified as a specific potential violation of the high-level goal it is decomposed from, and our design of the fault definition inference engine embraces the entailment and consistency refinement patterns discussed above.

In this study, to support algorithmic reasoning of control logic fault definition, we use ontology to specify the needed context of the HVAC system. Among context modeling and reasoning techniques, ontology-based models have benefits for the two reasoning tasks: 1) automatically derive knowledge about the current context, and 2) detect possible inconsistencies in the context source [64]. These two tasks are essential to the process of control logic fault definition, in the sense that task 1) is about deriving new knowledge of customized control logic fault definition, and task 2) is about using the ontology to specify and acquire information needed for control logic fault definition with consistency checking in terms of ontology instance validation check.

There are multiple studies about using domain ontologies in GORE for requirement elicitation and specification [65]. Although the prevalent use of ontologies in these studies is to model requirements with ontologies (e.g. [66]), there are studies about using ontologies to model system functionality and identify/refine goals and elicit requirements based on mapping between ontology and requirement document through morphological analysis [67, 68].

Our intended approach of ontological reasoning follows the same idea of Shibaoka et al. [68] for utilizing ontology to represent domain knowledge and help the goal refinement activity. However, there are major approach differences: In [68], the new requirements are identified through reasoning that is based on mappings between existing requirements and the domain ontology. For example, if in the ontology, there is a "require" relationship between class A and class B, and in the requirement document, it is identified that there is a requirement of "A needs to exist", then the proposed approach [68] provides suggestions to human requirement analyst to include requirement of "B needs to exist". In our approach, the inference mechanism conducts reasoning solely based on the ontology (i.e. no initial list of faults involved) and the output is not narrative suggestions to the research analyst, but machine-readable control logic fault definition that can be directly used in control logic fault detection.

It is worth noting that, in the building domain, there is a recent study [69] about knowledge-based FDD with integrated ontologies modeling multiple domains (building, mechanical system, BAS, and FDD rules). This study conducted fault detection with if-then inference rules documented in the "Fault" ontology to check information and data encoded in other ontologies. It also conducted fault diagnosis through the chains of if-then inference rules on the ontologies of fault states (symptoms), hypothesis and evidence. However, instances of the "Fault" ontology (including information about fault states, hypothesis and evidence), i.e. the considered faults in the FDD method, are hard-coded without any information about how they are elicited.

2.3.2 HVAC control logic specifications

As highlighted in Section 2.1, currently, existing knowledge on how to control and operate AHU systems during normal operation exists in narrative guidelines of different forms [31, 32, 33, 34, 35, 36, 37]. Due to the large number of possible component arrangements and control loop setup options, it is practically impossible to cover all possible AHU configurations in narrative guidelines. For example, the AHU component layout shown in Figure 1.2 is not covered in any of these reviewed guidelines, not to mention that even for AHUs with the same components layout, their control loop setup can be very different due to reasons such as different user requirements (e.g. different temperature requirements) and actuator specifications (e.g. actuation signal ranges).

In current practice, while the ASHRAE handbooks contain discussions of general control objectives of AHU systems, most of the reviewed documents describe example instantiations of these general objectives on specific categories of AHU systems in different formats, such as checklists, plots, and textual descriptions. Under this situation, applying these instructions to a system configuration that is not covered directly by the guidelines is highly subjective and unformalized. In this research, we identify general control goals from existing guidelines and develop a formalized approach to derive control logic fault definition based on the reasoning of these goals with no subjectivity. The research work of general control goal identification is discussed in detail in Section 2.4.

2.3.3 Control logic faults in HVAC FDD studies

Many systems have been developed for detecting and diagnosing HVAC faults, e.g. [13, 12, 70, 71, 72, 73, 74]. Each of these existing methods conducts FDD on a set of defined faults with observations of building automation system (BAS) data. We did not identify any HVAC FDD study that provides a systematic approach to specify a set of control logic faults of focus.

From existing HVAC automated FDD studies [46], we identify a handful of studies containing some control logic faults specified, as shown in Table 2.1. As can be observed from this table, the majority of these studies only provide high-level control logic problem descriptions (e.g. consider "control logic fault" as one type of fault) without quantitative specification of the control logic faults, while the ones providing quantitative control logic fault specifications only focused on specific HVAC systems considered in their studies. A systematic method to define control logic faults applicable to a variety of different systems is lacking.

In this research, we address this limitation by developing a formalized system to derive control logic fault definition based on specific AHU system information. The derived control logic fault definition are in terms of control logic input/output variable expressions, which are in the same form as those specified by existing studies [13, 12, 40, 75, 76]. Defining control logic faults in this way avoids ambiguity through mathematical representation, and it is easy for HVAC professionals to grasp and use for fault detection because the faults are specified with control logic variable (BAS data point) names of HVAC context.

Publication	System	Control logic faults specified in publication
Dexter and Pakanen [48]	AHU	Improper control action Incorrect initial values Improper range selection Improper run-time Incorrect flags Improper step size Scheduling errors Errors in the control logic
Choinière [42]	AHU	logic, tuning, signal, instability
Schein et al. [13, 12]	AHU	Outdoor air temperature too high for mechanical cooling with 100% outdoor air ($T_{oa} > T_{co} + \epsilon_t$) Outdoor air temperature too low for mechanical cooling with minimum outdoor air ($T_{oa} < T_{co} - \epsilon_t$) At least two of components (cooling coil valve, heating voil valve, and mixing box dampers) are modulating simultaneously ($u_{cc} > \epsilon_{cc}$, $u_{hc} > \epsilon_{hc}$, $\epsilon_d < u_d < 1 - \epsilon_d$)
Hyvarnen et al. [77]	AHU	E.g. supply air temperature controller - improper sequencing of valves and dampers mixed air controller - control signal (no signal, incorrect signal)
Brambley et al. [73]	AHU	E.g. Improper value for supply air temperature set point Error in control code (logic) Improper value for zone air temperature set point
Wang et al. [75, 40]	VAV	E.g. Too high minimum air flow rate set point ($T_{set} - T > \epsilon_t$ and $\|F - F_{set} < \epsilon_f$ and $F_{set} = F_{min}$ and $\|F_{min} - F_{D,min}\| > \epsilon_f$)
Lee et al. [76]	AHU	E.g. $U_{cc} - U_{cc,es} > \epsilon_{T_s}$ and $T_m - T_{m,es} < \epsilon_{T_m}$
Li [78]	AHU	Return fan at fixed speed Cooling coil valve control unstable cooling coil valve reverse action mixed air damper unstable Cooling coil control unstable Sequence of heating and cooling unstable
West et al. [79]	AHU	Cooling coil valve control fault

1. mathematical expressions of the control logic faults are listed if they are provided by the publication
2. mathematical symbol notations are omitted due to space limits, and are in cited publications

Table 2.1: HVAC control logic faults specified in existing studies

2.3.4 HVAC information models

Three categories of HVAC system-specific information to support the reasoning of control logic fault definition are discussed in Section 2.1. In order to derive control logic faults applicable to specific AHU system, all three categories of HVAC information need to be collected and provided to the reasoning mechanisms. In this section, we first discuss existing HVAC information model standards, then we discuss studies about conducting reasoning of HVAC information models for different purposes.

Many building information modeling (BIM) standards have been proposed. Two recent publications in 2018 [80] and 2010 [81] provided comprehensive reviews of this topic. Among existing BIM standards, the ones that modeled HVAC systems semantically at least to the component level are IFC [82] (including its model view definitions (MVD) e.g. HVACie [83]), gbXML [84], and Computer Models for the Building Industry in Europe (COMBINE II) HVAC equipment model [85]. As will be discussed in detail in Section 2.5.2, none of these BIM standards cover all the information needs for our control logic fault definition reasoning purpose. Thus, no existing BIM standard can be utilized directly to support the information requirements of deriving control logic fault definition.

Besides BIM standards, there have been several studies focusing on reasoning about additional HVAC related information from either existing BIM standards or newly developed ontologies, as shown in Table 2.2. Among these studies, all of the extended and newly developed HVAC ontologies aim at supporting the information requirements associated with the specific objectives of their research, as detailed in Table 2.2, and none of them contain all the system-specific information needed for control logic fault definition.

A notable new HVAC information model is the Brick schema proposed by Balaji et al. [93], for representing metadata (e.g. sensors' locations and types) in buildings. This schema specifies this metadata with a concrete ontology for sensors, subsystems and relationships among them. It utilizes the tagging concept of Haystack [94], the location concepts of IFC, and a semantic representation to utilize its flexibility and

Studies	Information model utilized	Objective
Bazjanac et al. [86]	extended the HVAC part of IFC schema	to facilitate interoperability between software used in HVAC system design, operation and analysis
Yang and Ergan [87]	used IFC schema	to deduce the functionalities of HVAC components from topological information in IFC schema
Liu et al. [88, 89]	proposed a functional taxonomy of AHU components	to formally represent AHU component functions and to infer component functional relationships
Bulbul and Akin [90]	proposed the Embeded Commissioning Model	to standardize the commissioning process
Chen et al. [91]	proposed a data schema	to represent building control knowledge
Schneider et al. [92]	proposed an ontology	to semantically model building control systems
Balaji et al. [93]	proposed the Brick schema utilizing concepts of Haystack and IFC	to represent metadata (e.g. sensors' locations and types) in buildings.

Table 2.2: Studies of reasoning about HVAC information from existing BIM or newly developed information models

extensibility properties [93].

In this research, by combining and extending existing BIM standard and product models, we develop a new HVAC ontology covering the information requirements for control logic fault definition. This will be discussed in detail in Section 2.5.2. Our objective is to 1) generate a unified schema to specify information requirements for deriving control logic fault definition, and 2) identify information sources for deriving control logic fault definition by mapping items specified in our ontology to those specified in existing BIM standards.

2.4 Goals of AHU control

As mentioned in Section 2.1, the desired control logic fault definition for specific AHUs are customized applications of the general objectives of energy efficiency and occupancy comfort. In this section, we elaborate these two general objectives into four goals at two levels of AHU system control so that each goal can be described with a precondition (if) and a postcondition (then) and can be applied to a specific

AHU algorithmically (details in Section 2.5.3) to identify all possible instances of a specific goal's application in a specific AHU system.

2.4.1 Two levels of HVAC control

fTwo levels of control exist in the control logic of AHU systems: local control and supervisory control [95]. Local control functions, such as proportional-integral-derivative (PID) control and ON/OFF switch control, adjust the control variable of a single device to achieve control process objectives, such as maintaining a set point [96]. Supervisory control functions are higher-level controls that are responsible for implementing the sequencing strategies [31] with consideration of device interactions and energy efficiency among multiple system components [97].

Sometimes in AHU control logic programs, there is another level of supervisory control, here referred to as optimal control that builds on top of the above mentioned supervisory control. In some studies, this optimal control is also referred to as supervisory control [96, 98]. Example functions of optimal control are scheduling strategies and dynamic set point reset strategies. These strategies are more related to requirements of building owners and mechanical designers to achieve energy saving objectives than they are related to physical principles of operating the systems. Thus, this level of control is out of the scope of this chapter.

We elaborate the general control objectives into goals, so that each goal can be represented by a sequence of algorithmic steps to solicit its application instance with regard to a specific AHU system. The general control objective of occupancy comfort maximization is about using the component's functionality in the correct way and at designed capacity. This objective can be elaborated into two goals at the local control level. The general control objective of energy efficiency maximization is about sequencing components to achieve best overall usage of multiple components.

This objective can be elaborated into two goals at the supervisory control level. The following two subsections describe these goals in detail.

2.4.2 Goals at local control level

The goals of achieving maximum occupancy comfort at local control level indicate that the functionalities of AHU components shall be used in the correct way and with maximum capacity if needed, to fulfill occupants' requirements. These goals are representations of the characteristics of close-loop control theory. In ASHRAE handbooks, example descriptions of this objective are: "A closed loop or feedback control measures actual changes in the controlled variable and actuates the controlled device to bring about a change. The corrective action may continue until the controlled variable is at set point or within a prescribed tolerance." (in ASHRAE Handbook - Fundamentals [99] Ch 7.1), and "... the longer error e exists, the more the controller output changes in attempting to eliminate the error..." (in ASHRAE Handbook - Fundamentals [99] Ch 7.3). As indicated by the formula below (t is the time variable), for a closed-loop PID controller, the controller output command, $u(t)$, is computed from the error value, $e(t)$, through a weighted sum of the proportional, integral, and derivative terms. K_p, K_i, and K_d are the weights of these three terms, respectively.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

If the control objective is to have a process variable (e.g. supply air temperature) track a specific set point value $r(t)$ (e.g. supply air temperature set point), then $e(t) = r(t) - y(t)$, where $y(t)$ is the measured process variable. If the control objective is to maintain a process variable above a low limit set point value, then $e(t) = r(t) - y(t)$, if $y(t) \leq r(t)$; and $e(t) = 0$, if $y(t) > r(t)$. Similarly, if the control objective is to maintain a process variable below a high limit set point value, then $e(t) = r(t) - y(t)$, if $y(t) \geq r(t)$; and $e(t) = 0$, if $y(t) < r(t)$. This characteristic of the control objective leads to the specialization of set point types in the development of the ontology, which will be detailed in Section 2.5.2.

AHU system control mostly uses proportional-integral (PI) controllers [38] because a proportional (P) only controller is not able to track a set point without error, and the appearance of a derivative (D) term makes the controller hard to tune while the main contribution of the derivative term is to have the controller more responsive in tracing the change of set points, which is not an essential need for AHU control. The following two goals are depictions of the saturation characteristic of closed-loop PI controllers:

- Goal 1: If positively actuating a component (increasing the control command) helps achieve the control objective and the control objective is continuously[1] not reached, then the actuation should saturate at its maximum. For example, if the supply air temperature is continuously lower than the supply air temperature set point, the preheat coil valve command should saturate at its maximum point, i.e. the preheat coil valve should be commanded to fully open.

- Goal 2: If negatively actuating a component (decreasing the control command) helps achieve the control objective and the control objective is continuously not reached, then the actuation should saturate at its minimum. For example, if the supply air temperature is continuously higher than the supply air

[1] "continuously" essentially represents the amount of time for the error term $e(t)$ to accumulate so the integral term in a PI controller saturates the control signal. This can be implemented as a fault trigger timer during fault detection activity. It is the commissioner's discretion to decide the amount of time that represents "continuous". We use this term multiple times throughout the description of Goal 1 and 2, with the same meaning.

temperature set point, the preheat coil valve command should saturate at its minimum point, i.e. the preheat coil valve should be commanded to close.

2.4.3 Goals at supervisory control level

The goals at supervisory control level are elaborated from the energy efficiency objective of sequencing multiple components. To achieve maximum energy efficiency, one should always try to use the most efficient way to achieve service requirements, and one should avoid simultaneously activating multiple energy-consuming components "fighting" with each other. Example descriptions of this efficiency objective at supervisory control level are: "Supervisory control variables should be chosen to maximize the coefficient of performance (COP)[2] of the system at all times while meeting the building load requirements." (in ASHRAE Handbook - Application [31] Ch 42.4). At the supervisory control level, sequencing of multiple components should comply with the following two goals about multi-component sequencing:

- Goal 3: If actuating each of multiple components helps achieve the control objective, then the more energy efficient component should be actuated before using the less energy efficient component, and the less energy efficient component should only be active when the actuation of more energy efficient component has been saturated. For example, if the heat wheel can be used for heating, it should be actuated to the maximum capacity before the actuation of the preheat coil valve starts.

- Goal 4: If actuating two components have opposite effects on the control objective, then they should not be active at the same time. For example, the preheat coil valve command and the cooling coil valve command should not

[2]COP: "Heating or cooling output divided by electrical power input; may include chilled-water(CW) pump and tower fan as well as compressor power" [31]

be positive at the same time. Note that if the actuation has other effects in addition to the conflicting control objective, then this is not necessarily a fault. For example, a cooling coil valve command and a reheat coil (a heating coil that is at the downstream of the cooling coil) valve command being positive at the same time is not fault, as the cooling coil is also used to dehumidify the supply air. Cases like this will be solved by the fault definition conflict resolution mechanism discussed in Section 2.5.4.

In this goals identification work, Goal 1 and 2 are based on the saturation feature of closed-loop PI/PID controllers, and Goal 3 and 4 are based on efficiency maximization of multi-component sequencing. Although Goal 1 and 2 are based on the same control theory, their reasoning steps to derive fault definition are not identical, so they are split into two goals. Goal 3 and 4 are split into two goals due to the same reason.

Existing case studies discussing HVAC control issues are reviewed by Ardehali and Smith [4]. In these reviewed case studies, 14 different control logic faults about AHU normal operation are specifically mentioned. Among these 14 types of control logic faults, 12 faults are violations of the above four goals. This justifies the prevalence of the identified goals. More importantly, these four prevalent and straightforward goals served as the starting point of our formalism development. As will be discussed in Section 2.6.3, we develop our proposed approach to be extensible, so new goals can be added without the need of modifying goals already implemented.

2.5 Fault definition formalism development

2.5.1 Fault definition derivation strategy

In this research, with the focus of control logic faults, we propose an approach to formalize the process of fault definition. The vision is to develop a formalism that reasons about a specific AHU component and control information, and derives a set of applicable control logic faults for this specific AHU. Each control logic fault describes the symptom of a possible violation of general control goals in terms of control logic input/output variable expressions. The central of this formalism is a classification task, which begins with input data stating the problem settings and identifies classes as solutions [49].

Classification approaches have been used in areas such as human factor analysis [100] and medical diagnostic problems [101]. As shown in Figure 2.1, we apply the classification paradigm [49] to the control logic fault definition problem by: 1) considering the specific AHU system information as the input data; 2) considering the control logic fault definition as the output solution; and 3) considering the fault definition reasoning process as an integration of three sequential steps – extracting only needed information from input data for fault definition reasoning use (data abstraction), deriving potential fault types (knowledge based mapping) and assemble final fault definition expressions with conflict resolution (solution refinement).

There are two primary inference methodologies that can be used to implement classification systems: forward chaining and backward chaining [102]. Forward chaining is a data-driven search, which uses knowledge to infer the solution from initial information input and facts. Contrarily, backward chaining is a solution-driven search that starts from a hypothetic solution and traces back using knowledge to check whether the initial inputs hold. Thus, forward chaining is appropriate for

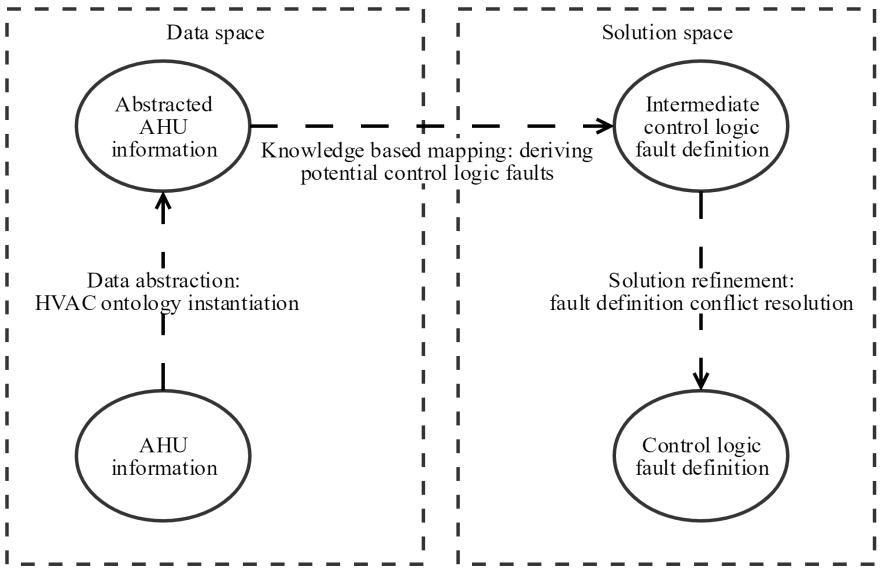
Data space
Solution space
Abstracted AHU information
Intermediate control logic fault definition
Knowledge based mapping: deriving potential control logic faults
Data abstraction: HVAC ontology instantiation
Solution refinement: fault definition conflict resolution
AHU information
Control logic fault definition

Figure 2.1: Control logic fault definition program under the classification paradigm

problems with the initial inputs given and for which there are a large number of potential solutions, while backward chaining is appropriate for problems with an easily formulated hypothetic solution and a large knowledge base [103]. For the control logic fault definition problem we are targeting, the initial information input of AHU component and control information is available while the output control logic fault definition set cannot be pre-defined without knowing anything about specific system information. Because of this, we design our classification system with forward chaining reasoning mechanisms.

Domain knowledge to be used in knowledge systems can be categorized into two types: declarative knowledge and procedural knowledge. Declarative knowledge refers to descriptions of factual knowledge, while procedural knowledge refers to descriptions of how to perform certain activities [104, 105].

In our research, the information we use to specify AHU is factual and the control goal-based reasoning mechanism is procedural. We utilize object-oriented structure to represent both factual and procedural knowledge. The abstraction [106] characteristic of object-oriented approach, enables the extensibility of both the ontology and the reasoning algorithms. Thus, new components and their functions can be included into the ontology, and new goal considerations can be included into the reasoning mechanism easily without modifying existing implementations.

To summarize the AHU control logic fault definition formalism development strategy: we adopt the forward chaining design methodology to develop an approach using the classification paradigm. The formalism development mainly includes an object-oriented information ontology that encodes AHU component and control information, and an object-oriented inference engine that implements the procedural knowledge of how to reason about control logic fault definition according to AHU control goals. Although being separately discussed in the following subsections, in our research work, the information ontology and the inference engine were developed concurrently due to their high inter dependency.

2.5.2 HVAC component and control configuration ontology development

As concisely depicted in Section 2.2, three types of AHU-specific input information are needed to support the customized control logic fault definition process. We follow the "form, function and behavior" design theory of product modeling [107] to develop the HVAC ontology. The ontology is developed by integrating and extending existing HVAC product models. None of the HVAC product models mentioned in Section 2.3.4 cover the information requirements of deriving control logic fault definition. Among these existing product models, we select IFC, gbXML and Brick as candidates for extension because they are semantic-rich information schemas covering HVAC domain semantics at least to the component level of granularity, and their full schemas are expressively specified and publicly available. This provides a solid starting point for extension. Moreover, by mapping information items between our developed ontology to these popular HVAC information models, we specify the information sources to facilitate our fault definition approach.

The information requirements discussed above can be further grouped into the following 5 categories. While the reasons of why this information is needed for deriving control logic fault definition will be clearer when we present the reasoning algorithms later, at this point we first discuss these information categories and their coverages in the three candidate schemas below, with some mentioning of why certain information is needed for the reasoning:

1. AHU component types: while all three schemas specify AHU component type, IFC does not differentiate component subtypes, such as different types of coils. The type specialization of AHU components is important because different components may have different functions to achieve different service requirements, and as will be illustrated later, the fault definition reasoning relies heavily on information about component functions.

2. AHU component controller information: IFC specifies this information with objects such as "IfcController", "IfcActuator", and "IfcSensor"; Brick uses different types of "Point" objects to specify this information, but does not cover information such as actuator saturation range and controller type; gbXML also does not cover this information, except for specifying "controlType". This information is essential to describe the actuation direction and saturation limits of control logic output variable, and is utilized in the definition of every control logic fault in our reasoning.

3. AHU component functionality: this category contains information items such as the air regulation functionality of AHU components (e.g. supply air heating), the component efficiency in terms of COP in achiving the function, and the condition needed for the component to provide its function (e.g. mixing box can only provide the function of supply air cooling when the outdoor air temperature is lower than the return air temperature). None of the schemas cover this category of information, except for gbXML covers the "efficiency" property.

4. AHU service requirements (service set points): service requirements of an AHU are specified in terms of desired air property values of multiple types (e.g. supply air temperature set point). This information is needed in the reasoning to identify if there is a need for a certain function. IFC does not specify this information; gbXML specifies temperature and pressure control designed value; Brick uses "Setpoint".

5. Associations between components, controllers, and functions: gbXML does not specify these associations; IFC uses relation objects; Brick specifies different types of associations.

In general, out of these 5 categories of information, IFC specifies 3; gbXML specifies 2; and Brick specifies 4. None of the information model can provide all categories of information listed above. Additionally, even if an information model specifies a certain category of information, in many cases, it does not specify 100% of the needed information items under that category for deriving control logic fault definition. Thus, we develop the information ontology by combining and extending these three information models to generate a new information model that itself would cover all information needed at desired granularity for deriving control logic fault definition.

The structure of the developed information ontology is demonstrated in Figure 2.2 as a UML class diagram. It contains the classes needed for providing information to define control logic faults for typical AHUs specified by ASHRAE [1]. It's ability to support the reasoning of control logic fault definition will be illustrated in Section 2.6.

For the purposes of 1) identifying the information source of the developed ontology from BIM standards, and 2) specifying which parts of the developed ontology is integrated from which BIM standards, a table explicitly mapping essential information elements specified in the developed ontology with existing HVAC information models is provided in Table 2.3.

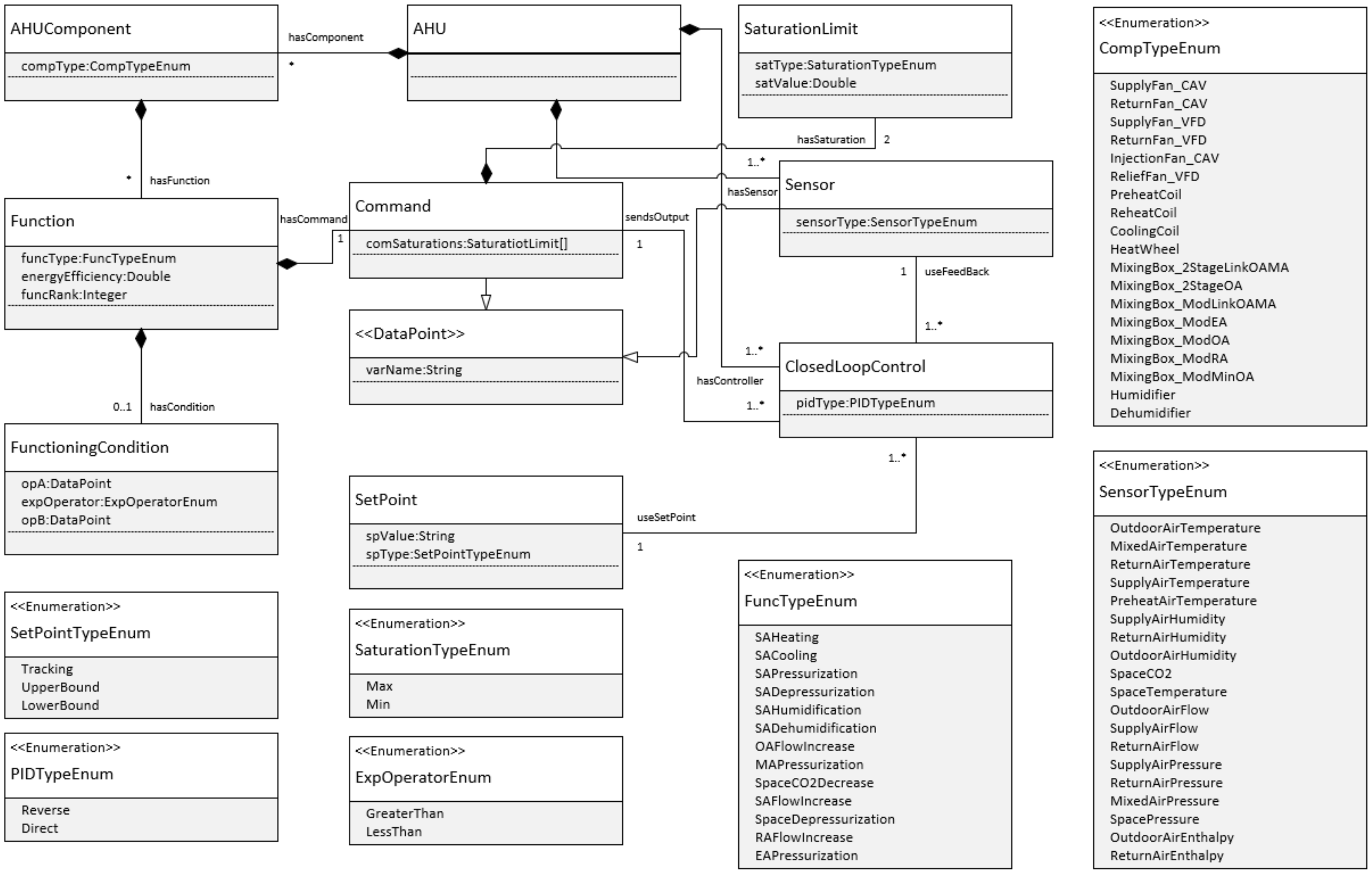

Figure 2.2: Ontology of required information

Information items category	Classes, attributes and associations in developed ontology		Item Type	BIM standards involving detailed HVAC modelling		
				Brick	**IFC 4**	**gbXML 6.01**
AHU component	AHUComponent		Class	Equipment	IfcDistribution-FlowElement	AirLoopEquipment
types information	CompTypeEnum		Enumeration	CoolingCoil, etc.	IfcCoil, etc.	equipmentTypeEnum
AHU component functionality information	Function		Class	/	/	/
	FuncTypeEnum		Enumeration	/	/	/
	funcRank		Attribute-Integer	/	/	/
	energyEfficiency		Attribute-Double	/	/	Efficiency (Efficiency. efficiencyType = "COP")
	FunctioningCondition		Class	/	/	/
AHU component controller information	ClosedLoopControl		Class	/	IfcController (Type:Proportional)	Control
	PIDTypeEnum		Enumeration	/	/	/
	DataPoint		Class	Point	/	/
	Sensor		Class	Sensor	ifcSensor	/
	SensorTypeEnum		Enumeration	Temperature Sensor, etc.	ifcSensorTypeEnum	/
	Command		Class	Command	IfcActuator	Control.controlType
	SaturationLimit		Class	/	/	/
	satValue		Attribute-Double	/	IfcController-P_Set: IfcPropertyBoundedValue	/
	SaturationTypeEnum		Enumeration	/	IfcController-P_Set: IfcPropertyBoundedValue	/
	varName		Attribute-String	/	/	/
AHU service requirements	SetPoint		Class	Setpoint	/	/
	spValue		Attribute-String	/	IfcController-P_Set: IfcPropertyBoundedValue .SetPointValue	TemperatureControl.DesignTemp, PressureControl.DesignPressure
	SetPointTypeEnum		Enumeration	/	/	/
	Class A	**Class B**	**Relation**			
Associations	AHUComponent	Function	has	/	/	/
	Function	FunctioningCondition	has	/	/	/
	Function	EnergyEfficiency	has	/	/	AirLoopEquipment. Efficiency.operationType
	Function	Command	has	/	/	/
	ClosedLoopControl	Command	association	isControlledBy	IfcRelConnectsElements	/
	ClosedLoopControl	SetPoint	association	/	P_Set	/
	ClosedLoopControl	Sensor	association	/	IfcRelConnectsElements	/
	Command	SaturationLimit	association	/	IfcRelConnectsElements	/

"/" indicates the information is not specified in the BIM standard

Table 2.3: Ontology information items and BIM information source mapping

We further explain and clarify several information elements specified in the ontology shown in Figure 2.2 below:

- *Function* class and its attributes: the *Function* class specifies information about AHU component functions, such as supply air heating, outdoor air flow increase, etc. the *energyEfficiency* attribute of this class specifies the efficiency level of the belonging AHU component performing this function, in terms of coefficient of performance (COP), i.e. how many units of energy are consumed by the component in order to provide 1 unit of energy for the function. The value of this attribute is used in the reasoning algorithm to compare efficiency of different components when they can perform the same type of function, for this purpose, the value of *energyEfficiency* does not need to be very accurate, as long as a comparison can be done between multiple components having the same function type. For example, we typically specify the supply air cooling function of cooling coil with *energyEfficiency* value of 1, indicating that it consumes 1 unit of energy from the energy source (e.g. chilled water) to provide 1 unit of cooling energy to the supply air. Another example is the supply air cooling with mixing box, which does not actively consume energy to provide the cooling (i.e. it uses the cooling energy from the cold outdoor air). In this case, we assign an *energyEfficiency* value of 99 or larger to indicate that it does not actively consume energy when providing the function.

- *FunctioningCondition* class: Some AHU components can only provide some of their functions under specific conditions. This class specifies these conditions. For example, the mixing box with three dampers in coordination can only provide the supply air cooling function when the outdoor air temperature is lower than the return air temperature. Note that not all component functions have functioning conditions. For example, the VFD supply fan can provide the function of supply air pressurization under all conditions during normal operation and thus won't need a functioning condition to be specified.

- *SetPoint* class and its attributes: The *SetPoint* class is used to specify the service requirements of the AHU. The enumeration attribute *spType* specifies the type of service requirements. For example, the requirement for supply air static pressure is typically a tracking set point, i.e. the supply air static pressure needs to be maintained around this specific value, neither higher nor lower is deemed fulfilling the requirement. On the other hand, the requirement for supply air temperature heating is a temperature set point of lower bound, i.e. as long as the supply air temperature is at or above the set point, the heating requirement is satisfied. Note that a heating lowerbound set point and a cooling upperbound set point can be used together to specify the user requirement of a deadband. The *spValue* String attribute of this class specifies the actual service requirement in terms of a static numeric set point value or a data point variable name if the design intent is to read dynamic service requirement set point from this data point.

- *ClosedLoopControl* class and its attributes: This class does not model the control logic program in any way. Instead, it is used to semantically link different elements of a closed-loop control process. The notion of two types of PID control [38], i.e., direct acting and reverse acting, is adopted as an attribute to specify the control command's effects on feedback (process) variable. For example, a *ClosedLoopControl* object can be used to organize a closed-loop supply air static pressure control, it includes the service requirement of supply air static pressure set point, the feedback variable of supply air static pressure sensor, and the control command of supply fan VFD speed. This control is a reverse acting control, meaning that a decreasing feedback sensor reading should cause the control command to increase.

2.5.3 Inference engine development

According to the design of our approach that was discussed in Section 2.5.1, we design the inference engine to be a two-layer object-oriented system, as shown in Figure 2.3. The first layer contains reasoning algorithms deriving control logic fault definitions within each of the considered goals, and the second layer contains algorithms combining control logic fault definitions from multiple goals and resolving fault definition conflicts. The first layer is discussed in detail in this subsection, and the second layer will be detailed in the next subsection

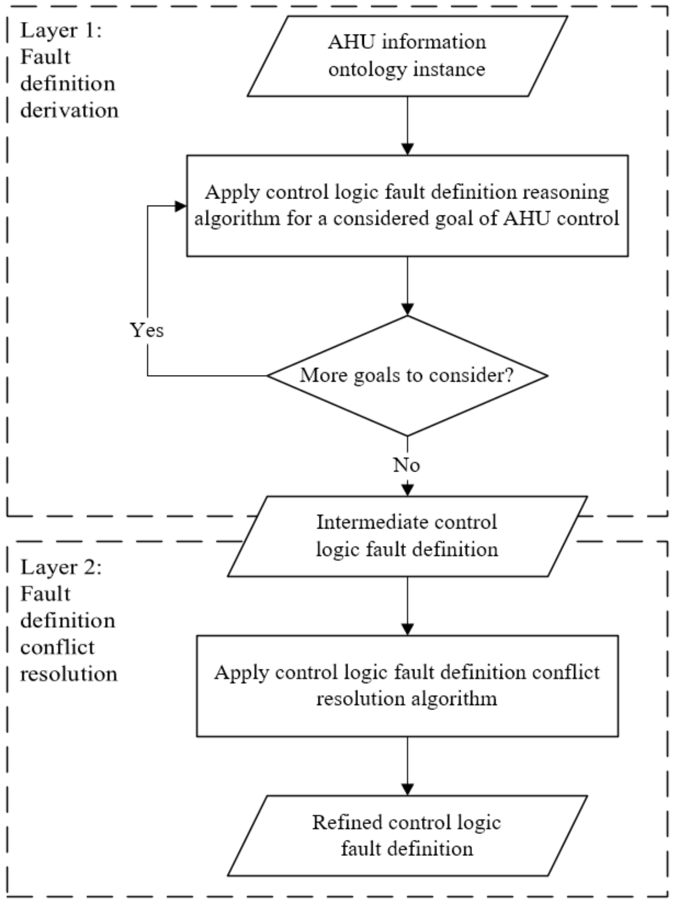
Layer 1: Fault definition derivation
AHU information ontology instance
Apply control logic fault definition reasoning algorithm for a considered goal of AHU control
Yes
More goals to consider?
No
Intermediate control logic fault definition
Layer 2: Fault definition conflict resolution
Apply control logic fault definition conflict resolution algorithm
Refined control logic fault definition

Figure 2.3: Flowchart of fault definition reasoning process

The reasoning mechanisms of deriving control logic fault definition are goal-specific. For each goal discussed in Section 2.4, we develop a reasoning mechanism in terms of algorithmic steps that take the AHU information instantiated from the ontology as the input, and provide definition of control logic faults as the output. Each defined fault is applicable to the specific AHU system information and indicates a violation of the corresponding general control goal.

As discussed in Section 2.5.1, the fault definition derivation mechanism is developed in an object-oriented manner for extensibility purpose. As pieces of procedural knowledge,the reasoning mechanisms can be implemented as algorithmic steps in methods of the class. The implemented forward chaining algorithms of defining control logic faults violating each of the considered four goals in Section 2.4 are shown as flowchart diagrams in Figure 2.4, 2.5, 2.6, and 2.7, respectively. The identification and detailed discussion of these four goals can be found in Section 2.4.

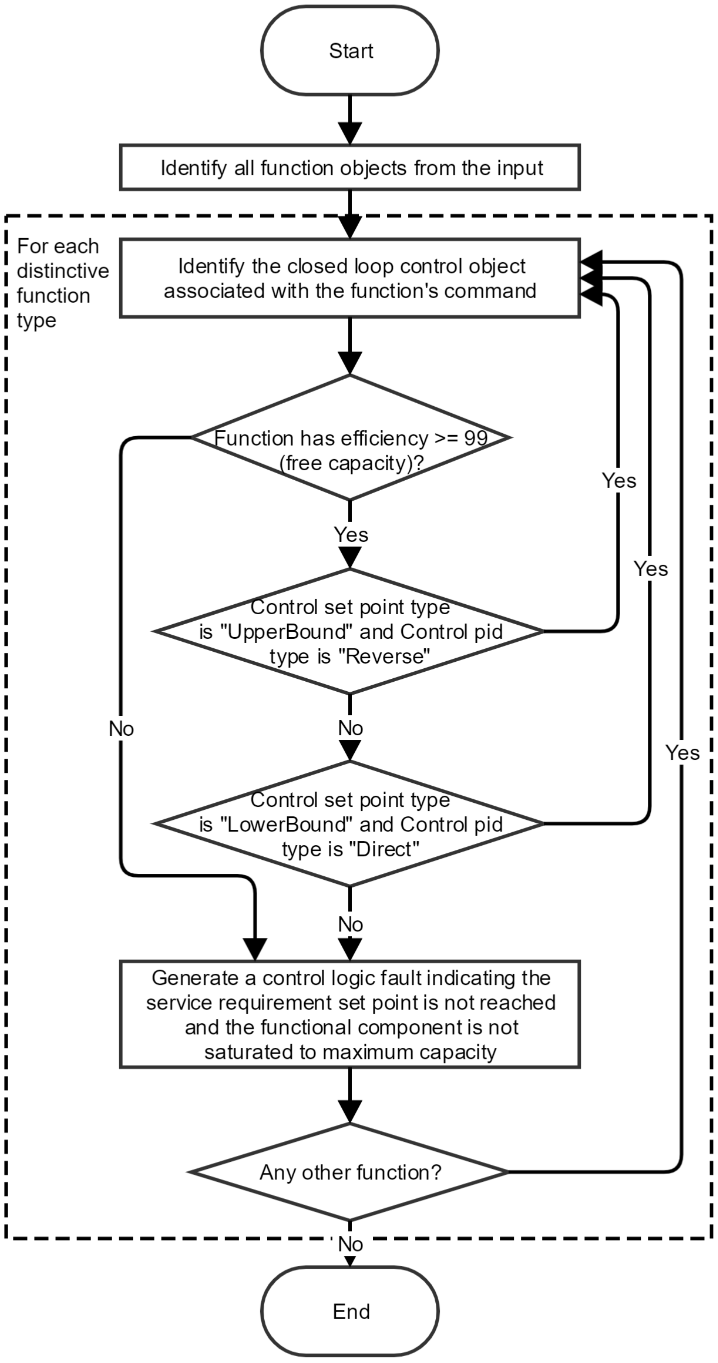
Start
Identify all function objects from the input
For each distinctive function type
Identify the closed loop control object associated with the function's command
Function has efficiency >= 99 (free capacity)?
Yes
No
Control set point type is "UpperBound" and Control pid type is "Reverse"
Yes
No
Control set point type is "LowerBound" and Control pid type is "Direct"
Yes
No
Generate a control logic fault indicating the service requirement set point is not reached and the functional component is not saturated to maximum capacity
Any other function?
Yes
No
End

Figure 2.4: Flowchart of reasoning algorithm for defining control logic faults violating "Goal 1": if positively actuating a component helps achieve the control objective and the control objective is continuously not reached, then the actuation should saturate at its maximum.

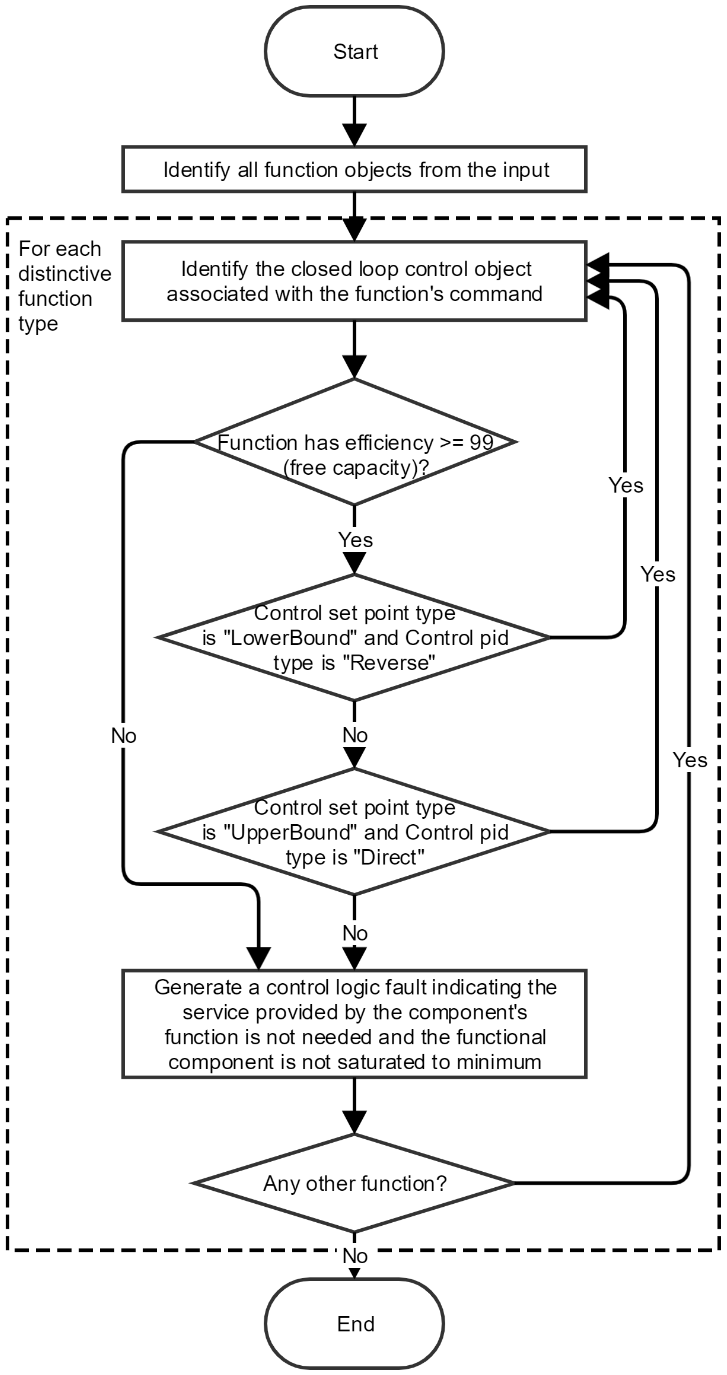
Start
Identify all function objects from the input
For each distinctive function type
Identify the closed loop control object associated with the function's command
Function has efficiency >= 99 (free capacity)?
Yes
No
Control set point type is "LowerBound" and Control pid type is "Reverse"
No
Yes
Control set point type is "UpperBound" and Control pid type is "Direct"
No
Yes
Generate a control logic fault indicating the service provided by the component's function is not needed and the functional component is not saturated to minimum
Any other function?
Yes
No
End

Figure 2.5: Flowchart of reasoning algorithm for defining control logic faults violating "Goal 2": if negatively actuating a component helps achieve the control objective and the control objective is continuously not reached, then the actuation should saturate at its minimum.

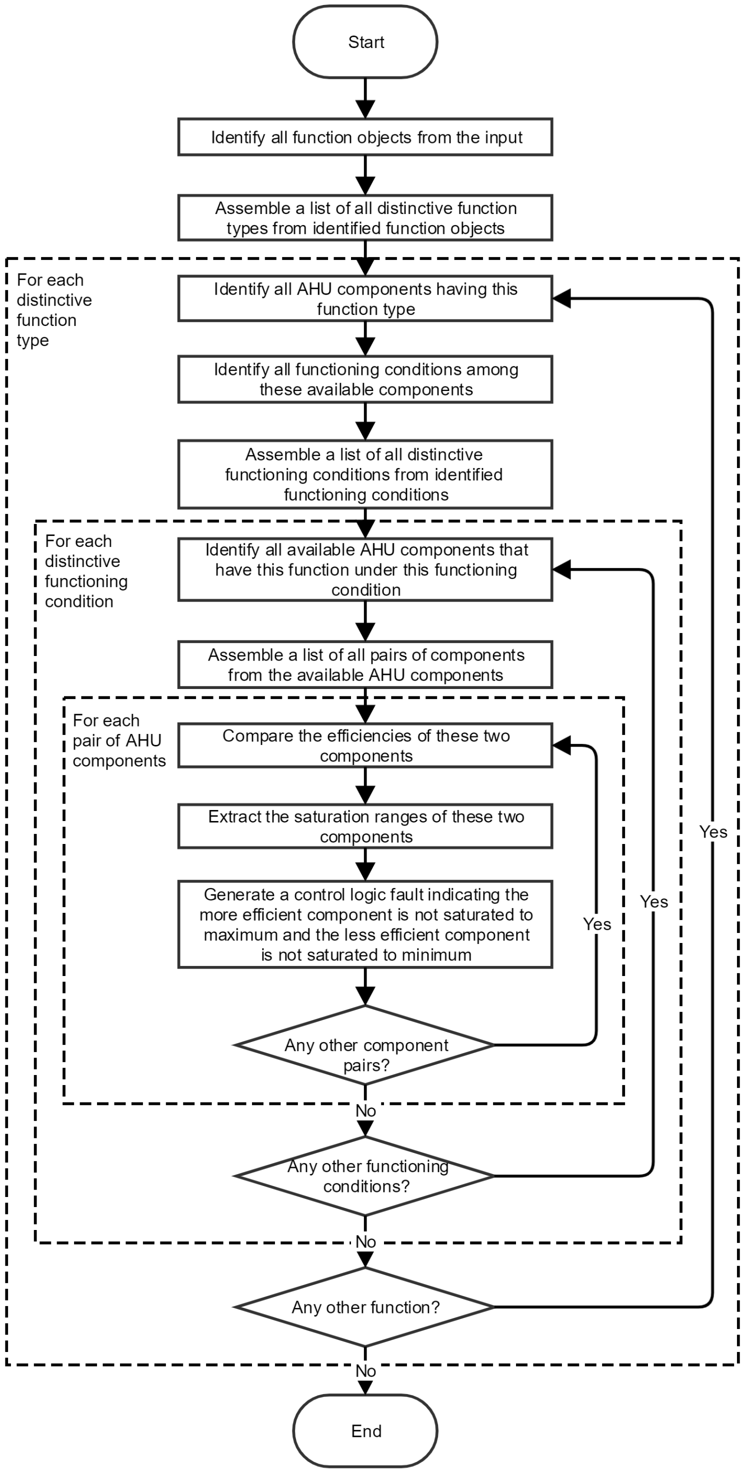
Start
Identify all function objects from the input
Assemble a list of all distinctive function types from identified function objects
For each distinctive function type
Identify all AHU components having this function type
Identify all functioning conditions among these available components
Assemble a list of all distinctive functioning conditions from identified functioning conditions
For each distinctive functioning condition
Identify all available AHU components that have this function under this functioning condition
Assemble a list of all pairs of components from the available AHU components
For each pair of AHU components
Compare the efficiencies of these two components
Extract the saturation ranges of these two components
Generate a control logic fault indicating the more efficient component is not saturated to maximum and the less efficient component is not saturated to minimum
Any other component pairs?
Yes
No
Any other functioning conditions?
Yes
No
Any other function?
Yes
No
End

Figure 2.6: Flowchart of reasoning algorithm for defining control logic faults violating "Goal 3": if actuating each of multiple components helps achieve the control objective, then the more energy efficient component should be utilized before using the less efficient component.

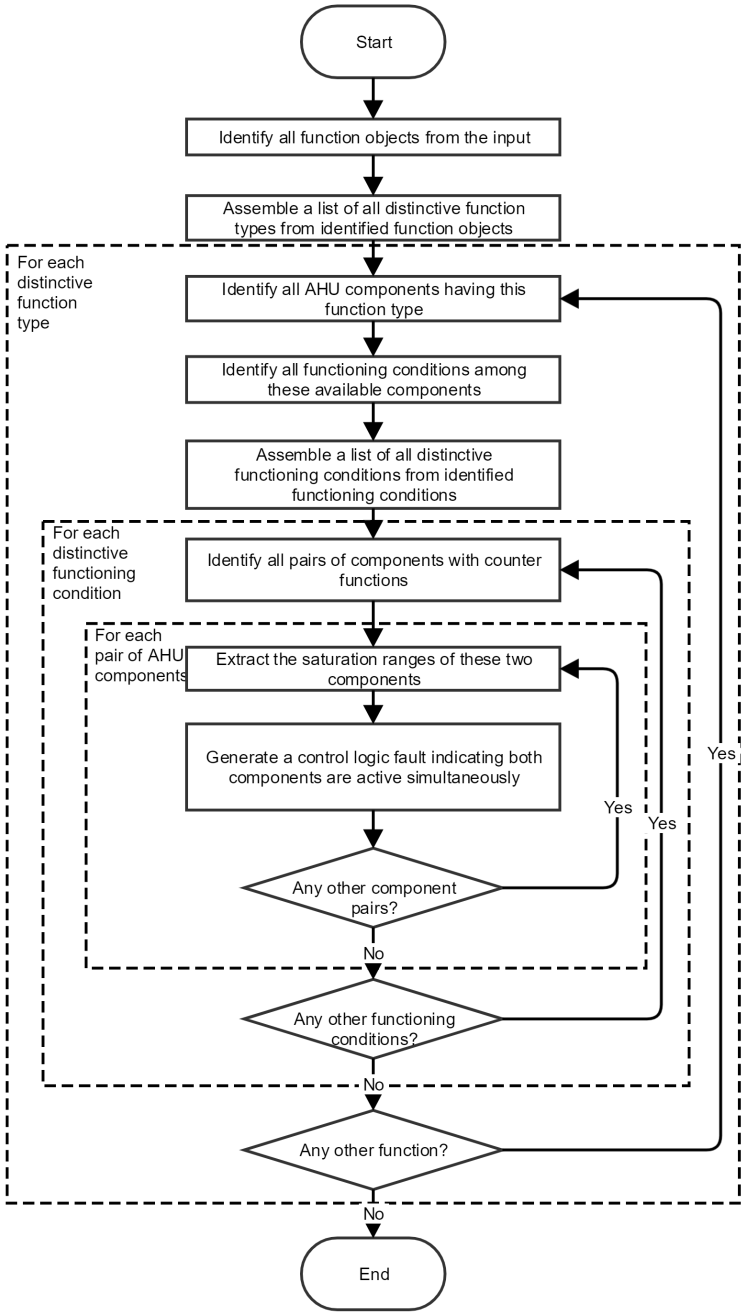
Start
Identify all function objects from the input
Assemble a list of all distinctive function types from identified function objects
For each distinctive function type
Identify all AHU components having this function type
Identify all functioning conditions among these available components
Assemble a list of all distinctive functioning conditions from identified functioning conditions
For each distinctive functioning condition
Identify all pairs of components with counter functions
For each pair of AHU components
Extract the saturation ranges of these two components
Generate a control logic fault indicating both components are active simultaneously
Any other component pairs?
Yes
No
Any other functioning conditions?
Yes
No
Any other function?
Yes
No
End

Figure 2.7: Flowchart of reasoning algorithm for defining control logic faults violating "Goal 4": if actuating two components have opposite effects on the control objective, then they should not be active at the same time. The static information of counter functions (e.g. supply air heating and supply air cooling are counter function pairs) can be implemented as a configuration file of the software so it can be easily updated.

2.5.4 Control logic fault definition conflict resolution

All control goals discussed in Section 2.4 are about controlling component behaviors to achieve certain service requirement targets. Thus, the straightforward application of these goals is within the scope of one particular service function target, such as maintaining supply air temperature set point with supply air temperature control. A typical AHU has multiple service function targets to achieve simultaneously. When combing multiple sets of fault definitions together, conflicts may exist between two defined faults from different targets when, under certain circumstances, the control commands of two faults specify the same control variable to different saturation directions. That is, the negations of these two control expressions have no intersection.

For example, mixing box (control variable MAO, ranging from 0 to 100, representing the percentile portion of intaking outdoor air) can be used in two service functions, supply air temperature control (tracking supply air temperature set point SATSP) and supply air quality control (outdoor air flow rate low limit OAFSP). As discussed by ASHRAE handbook [31], the primary function of mixing box is to supply air quality control, and the secondary function is to supply air temperature control. When applying *Goal 1* in supply air quality control, one of the defined faults is: (OAF $<$ OAFSP) & (MAO $<$ 100). The interpretation is, when the outdoor air flow rate (OAF) is lower than the low limit (OAFSP), if the mixing box does not open to 100% outdoor air, then this is a fault. When applying *Goal 2* in supply air temperature control, one of the defined faults is: (SAT $>$ SATSP) & (RAT $<$ OAT) & (MAO $>$ 0). The interpretation is, when the supply air temperature (SAT) is higher than the set point (SATSP), and the return air temperature (RAT) is higher than the outdoor air temperature (OAT), if the mixing box does not open to 100%

return air (hotter air), then this is a fault. A conflict exists between these two faults because, when the AHU is in the scenario of (OAF < OAFSP) & (SAT > SATSP) & (RAT < OAT), to be fault-free, MAO needs to be 100 (avoiding the first fault) and 0 (avoiding the second fault), which is impossible.

As described above, our objective is to resolve requirements conflicts of the control logic software when merging the requirements from different control objectives. We develop our approach of resolving these conflicts based on existing approaches of software requirement conflict resolution. The work of fault definition for HVAC control logic programs to achieve the compliance of general control goals is a case of requirement definition in goal-driven requirement engineering (GORE) [57, 56], in which a requirement exists because some underlying goal provides a base for it [108, 109]. A well-cited study on requirement conflict resolution in GORE is authored by Lamsweerde et al. [110], in which the authors discussed various techniques for systematically resolving conflicts by introduction of new goals or by transformations of goals / objects towards conflict free versions. Approaches of resolving conflicts through goal transformation include avoiding boundary conditions, goal restoration, conflict anticipation, goal weakening, and alternative goal refinement; Approaches of resolving conflicts through object transformation include object refinement and agent refinement. Detailed definitions of these approaches and their contexts are discussed in [110].

In the context of HVAC control, the goals elaborated from high-level general control objectives of maximizing occupancy comfort and energy efficiency should not be compromised by being replaced with new goals. With this prerequisite, we propose a conflict resolution approach to prioritize control logic faults in terms of the priorities of single AHU component's multiple functions and revise faults of low priority function to avoid their conflicts with high priority faults. This approach is instrumented through the combination of object refinement [110], alternative goal refinement [110] and pairwise requirement prioritization [111, 112, 113], in the following sense:

- Object refinement: In the HVAC ontology (discussed in Section 2.5.2 and visualized in Figure 2.2), the *Function* class of an AHU component is further specialized with the attribute of *funcRank* – function rank to distinguish the function priorities of each component if a component has more than one functions.
- Alternative goal refinement: The original goal is to avoid all possible violations of considered goals for each of the components under every service function. This goal is refined to an alternative goal, that is to avoid goal violations of primary functions first, and then to avoid goal violations of the secondary function of a component only when the primary function of this component is fault free. This process will be illustrated with a concrete example below.
- Requirement prioritization: According to the alternative goal discussed above, the fault definitions are compared pairwise and prioritized according to the function rank of different functions within one component.

To realize this approach, the only additional information needed from the user is the specialization of AHU component function ranks. This information is discussed in the ASHRAE handbook [31] and is provided in sequence of operations of AHU systems in many cases [1]. With this additional information encoded in the ontology, we show the algorithm of this conflict resolution approach in Figure 2.8 as a flowchart. The implementation of this algorithm can resolve conflicts among control logic faults defined with reasoning mechanisms in the first layer of the inference engine that are discussed in Section 2.5.3.

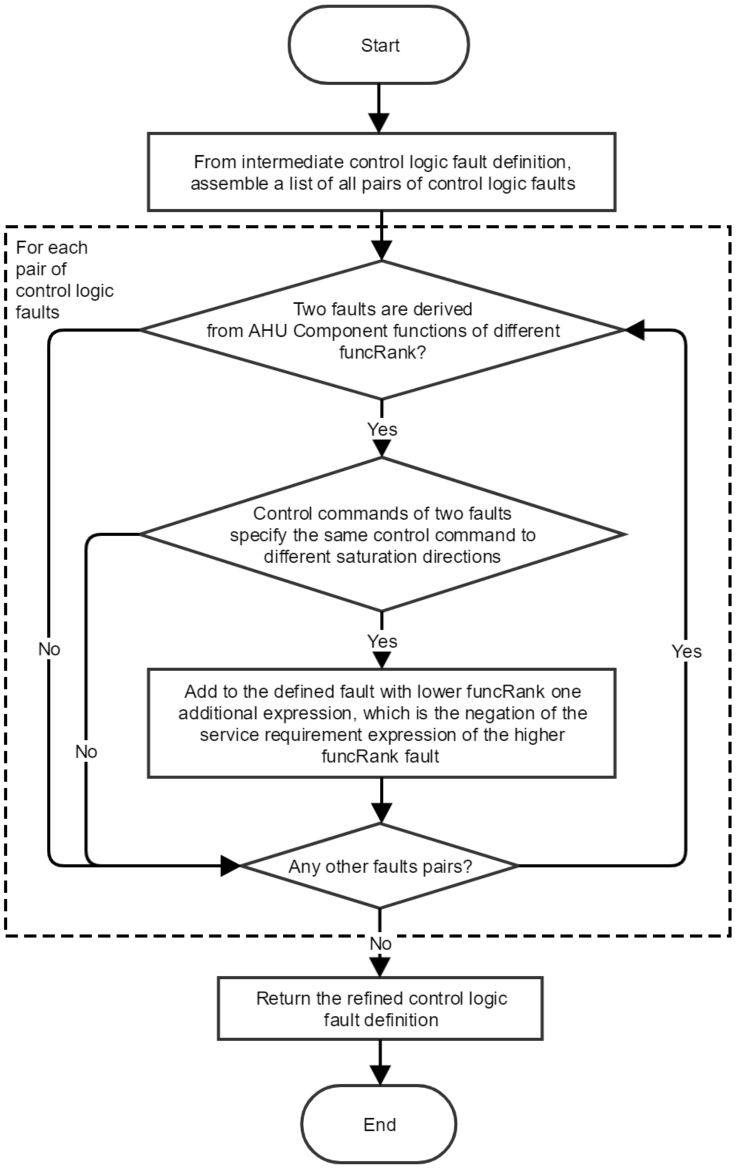
Start
From intermediate control logic fault definition, assemble a list of all pairs of control logic faults
For each pair of control logic faults
Two faults are derived from AHU Component functions of different funcRank?
Yes
Control commands of two faults specify the same control command to different saturation directions
Yes
Add to the defined fault with lower funcRank one additional expression, which is the negation of the service requirement expression of the higher funcRank fault
No
No
Yes
Any other faults pairs?
No
Return the refined control logic fault definition
End

Figure 2.8: Flowchart of control logic fault definition conflict resolution algorithm

Continuing with the example of two conflicting defined faults, to resolve the conflict, we first specialize the component functions with function rank property *funcRank*. For mixing box, the supply air quality control function has *funcRank=1*, and the supply air temperature control function has *funcRank=2*. This is determined according to the functionalities of AHU components described in ASHRAE handbooks and other guidelines. Then with the refined alternative goal stated earlier, the two conflicting fault definitions are compared and prioritized:

- The fault defined under the primary function of supply air quality control is prioritized and remained unchanged: $(\mathrm{RAC} > \mathrm{RACSP}) \wedge (\mathrm{MAO} < 100)$.
- The fault defined under the secondary function of supply air temperature control is specialized by adding one expression, and the resulting fault definition is: $(\mathrm{SAT} > \mathrm{SATSP}) \wedge (\mathrm{RAT} < \mathrm{OAT}) \wedge (\mathrm{MAO} > 0) \wedge (\mathrm{RAC} < \mathrm{RACSP})$

After revising the definition of the second fault, the conflict is resolved.

2.6 Prototype implementation and evaluation results

2.6.1 Prototype implementation

The proposed system discussed in Section 2.5 is implemented into a prototype application for validation. The developed HVAC information ontology is implemented as an XML Schema Definition (XSD) file. Following the methodology of implementing UML structures in XSD discussed in [114, 115], the classes and attributes in the ontology are implemented as XML elements and XML attributes respectively, and associations are modeled by implementing id and reference attributes in the XML schema.

The inference engine is implemented in the Java programming language. It processes the AHU information XML as the input and returns a list of defined control logic faults as the output. [3]

2.6.2 Control logic fault definition results

In order to evaluate the performance of the proposed approach, the prototype is used to generate control logic fault definition for 27 AHUs of which the component and control information is specified by ASHRAE publication "Sequence of Operation for Common HVAC Systems" [1]. These 27 AHUs are different in terms of the following aspects [1]:

- Fan type
- Coil type
- Supply air static pressure control
- Supply air temperature control
- Space pressure control
- Minimum outside air control
- Economizer type

Sequence of operation in normal operation mode, component and sensor arrangements of the 27 AHUs are specified in [1] and are used to instantiate the AHU information XML files based on the developed ontology. The ontology instances of these AHUs are provdied in Appendix A. Information such as control input/output

[3]The schema and prototype will be released as open source code at `https://github.com/leijerry888/` once the review process is over.

variable names and some actuator saturation ranges is not specified in the publication but exists for real-world AHUs. In this evaluation, this information is created based on typical values and naming conventions observed in real-world AHU systems. For example, we assign the variable name of *ahu1pho* for the preheat coil valve of ASHRAE AHU 1, and specify the saturation range of this valve to be from 0 to 100. This information only affects the final representation of the defined control logic faults, and it does not affect what control logic faults will be defined based on the reasoning mechanisms. So our self discretion of assigning this information will not impact the precision and recall of control logic fault definition in our evaluation, which will be discussed below.

In order to validate the control logic fault definition outcome of the formalism we proposed in this research, we need to first build the ground truth control logic fault definition of the validation set AHUs. Following the reasoning ideas discussed in Section 2.4, we interpreted the sequence of operation sentence by sentence and manually generated control logic fault definition in terms of control logic variable input/output expressions. Although the sequence of operation narratives in the validation set are written in a uniform simple sentence structure (e.g. "The air handling unit supply fan speed shall modulate to maintain duct static pressure setpoint."), we recognize the subjectivity in this manual fault definition process, and the following sequential steps are taken to minimize the subjectivity:

1. Decompose the sequence of operation narratives into a list of simple sentences.

2. Exclude sentences providing general descriptions[4], discussing temporal requirements[5], or discussing optimal control requirements[6]. These requirements

[4]E.g. "The supply fan shall be energized."
[5]E.g. "When a cooling stage is called to run, it will run for at least 5 minutes."
[6]E.g. "Supply air temperature setpoint shall be reset based on space temperature according to

are out of the scope of this research, as discussed earlier.

3. For each of the considered sentence:

   - if it is discussing the operation of one component to achieve one setpoint (E.g. "The air handling units supply fan speed shall modulate to maintain duct static pressure setpoint of 0.75 inch of water (adjustable)."), then the reasoning of control goals 1/2 discussed in Section 2.4 are used to define control logic faults.
   - If it is discussing the operation of multiple components in coordination to achieve one setpoint (E.g. "The heating shall stage and the mixed air dampers and cooling coil valve shall modulate in sequence to maintain space temperature setpoint."), then the reasoning of control goals 1/2/3/4 discussed in Section 2.4 are used to define control logic faults.
   - If it is discussing specific conditions that apply to a component when providing functions, (E.g. "There shall be a mixed air low limit function to modulate the mixed air dampers closed to prevent the mixed air temperature from dropping below the mixed air low limit setpoint of 45F (adjustable).") this condition is directly specified and indicated in related faults defined if applicable.
   - If it is solely discussing a command saturation limit of a certain component, (E.g. "The supply fan speed shall not drop below 30% (adjustable) to assure adequate fan motor cooling."), no dedicated fault will be defined only about violations of this saturation limit. However, this saturation limit will be used to specify related faults of the component.

the following reset schedule."

The control logic fault definitions generated by executing the prototype implementation of our approach are compared with the ground truth control logic fault definitions discussed above. The results are summarized in Table 2.4. Precision is used as the evaluation metric to indicate the percentage of prototype defined faults that match the gorund truth fault definition. Precision values are threatened by false positive instances each indicates that a defined fault is actually not a fault applicable to the corresponding AHU. The false positives are caused by specific requirements in the sequence of operation (in most cases component-specific requirements) that invalid certain defined control logic faults under certain conditions. Recall is used as the evaluation metric to indicate the percentage of the ground truth fault definition that are correctly specified in the fault definition formalism output. Recall values are threatened by false negative instances each indicates that a ground truth fault is not specified or not specified correctly in the fault definition provided by the formalism. These false negatives are largely caused by sequence of operation items about control sequencing that are not included in the four reasoning goals discussed in Section 2.4 and the same reason causing false positives. Overall, the prototype defines a total of 343 faults for the 27 AHUs in the validation set and achieves an average precision of 94.2% and an average recall of 83.0%.

In the evaluation, it takes the authors an average of 20 minutes to prepare the input XML file of each AHU based on the ontology (XSD) with the help of an XML editor [116]. For a validation prototype, we didn't implement a graphical user interface. Instead, the XMLs are manually written based on textual templates and the XSD file. It is expected that a graphical user interface and/or a parser to automatically acquire information from available BIM files will make the preparation of the AHU information XML file much more efficient in the future. The rest of the process

AHU ID	**# Faults derived by formalism**	**# Faults specified based on SOO**	**# False positive**	**# False negative**	**Precision**	**Recall**
1	8	9	1	2	87.5%	77.8%
2	8	10	2	4	75.0%	60.0%
3	8	10	2	4	75.0%	60.0%
4	5	5	0	0	100.0%	100.0%
5	9	11	2	4	77.8%	63.6%
6	8	10	2	4	75.0%	60.0%
7	8	10	2	4	75.0%	60.0%
8	8	10	2	4	75.0%	60.0%
9	14	15	3	4	78.6%	73.3%
10	8	8	0	0	100.0%	100.0%
11	16	17	3	4	81.3%	76.5%
12	14	15	0	1	100.0%	93.3%
13	15	16	0	1	100.0%	93.8%
14	15	19	0	4	100.0%	78.9%
15	16	16	0	0	100.0%	100.0%
16	15	19	0	4	100.0%	78.9%
17	16	16	0	0	100.0%	100.0%
18	14	15	0	1	100.0%	93.3%
19	15	16	0	1	100.0%	93.8%
20	11	12	0	1	100.0%	91.7%
21	13	14	0	1	100.0%	92.9%
22	17	22	1	6	94.1%	72.7%
23	17	20	0	3	100.0%	85.0%
24	17	20	0	3	100.0%	85.0%
25	16	20	0	4	100.0%	80.0%
26	16	17	0	1	100.0%	94.1%
27	16	17	0	1	100.0%	94.1%
Total	343	389	20	66	94.2%	83.0%

Table 2.4: Summary of fault definition results of AHUs specified by ASHRAE [1]

is automated and the ending results are machine-processable control logic program input/output variable expressions representing the symptoms of defined faults. In addition to the textual control logic fault definitions, the developed prototype is able to output the defined control logic faults in terms of Python scripts that can be utilized directly for control logic fault detection.

2.6.3 Generality, comprehensiveness and extensibility

Generality

The generality of the developed system is validated by testing with AHUs of different configurations, as shown in Table 2.4.

Comprehensiveness

The authors recognize the fact that we cannot implement all possible AHU control goals, components, and functions in the system design once and for all, especially with the advancing of technologies leading new hardware inventions and new strategies of system control. In this sense, the proposed system is not intended to cover control goals, components and functions comprehensively.

However, for goals implemented in the system, and for components and functions that have been covered in the developed ontology, our proposed system comprehensively defines applicable faults violating considered goals by having the algorithm iterating over all components and functions in an AHU, with an average precision of 94.2% and an average recall of 83.0%

Extensibility

In our design of the system, procedural knowledge of how to derive control logic fault definition from a specific goal (e.g. Figure 2.6) is implemented as a dedicated method in the inference engine. Extending the system with consideration of new general goals is realized by adding new methods in the inference engine. Extending the system with new functions or components is realized by updating the corresponding Enumeration definition in the ontology. Except for these updates, no other changes of the existing system are needed.

To show extensibility, we create Table 2.5 listing typical scenarios of extensibility and corresponding steps of how to extend the system for each scenario in the context of the prototype implementation.

No.	Scenario	Extending steps
1	Extend with a new AHU component type	1. Add the new AHU component type name as a new element in Enumeration class "CompTypeEnum" in the ontology file (XSD) 2. Confirm that functions of the newly added AHU component type are already listed in Enumeration class "FuncTypeEnum", if not, continue to scenario 2
2	Extend with a new AHU function type	1. Add the new AHU component function type name as a new element in Enumeration class "FuncTypeEnum" in the ontology file (XSD) 2. If the new AHU component function type has a counter function (e.g. supply air heating as a counter function of supply air cooling), then add the counter function type as a new element in Enumeration class "FuncTypeEnum" in the ontology file (XSD) if it is not already in there 3. If the new AHU component function type has a counter function, add this counter function pair in the counter functions configuration file (CSV)
3	Extend with a new general control goal	1. Design the reasoning algorithm of this general control goal in a way that it uses information specified by the ontology and outputs a list of control logic faults each representing a possible violation of this general control goal 2. Implement this reasoning algorithm as a new Java method in the "FaultDefinition" class in the following manner (a) The input argument of this method is a object of type "XMLReader", which is a wrapper class we wrote for JDOM (b) This method add each newly derived control logic fault as a "Fault" class object to an "ArrayList<Fault>" object of the "FaultDefinition" class, the "Fault" class is described below (c) The "Fault" Class mainly contains an ArrayList of "FaultExpression" and a double variable recording the "funcRank" value of the fault related function (value from the ontology instance) (d) The "FaultExpression" class mainly contains String type variables of the operator and two operands of a logical expression (e.g. "OAD" "<" "100"), and an expression type enumeration object with potential values of "FUNCTION", "ENVIRONMENT", "CONTROL", "EMPTY" 3. Add a method call to the newly added method with the argument of ontology instance "XMLReader" object in the Main class adjacent to the method calls of already considered reasoning algorithm methods

Note: only key prototype implementation details related to the extension are mentioned in the table. Complete details of the prototype implementation can be found in the prototype source code.

Table 2.5: Extensibility scenarios of the prototype

2.7 Discussion of application boundaries

The way we designed our approach puts certain boundaries for the AHU systems and their control software that, if going over the boundaries, the proposed formalism would not apply without modifications. With the appearances of more advanced control methods for HVAC systems, such as model predictive control [117, 118, 119, 120], fuzzy logic control [121, 122, 123, 124], and neural network control [125, 126, 127], it is likely that in the future, new HVAC systems will be controlled in different ways than the traditional DDC control. Thus, we are motivated to discuss the application boundaries of our approach as follows:

- Deterministic service requirements and component functional information: the control logic faults are defined based on the information provided by the ontology instance, i.e., if the AHU information specified in the ontology changes, the control logic fault definition may change accordingly. Thus, if this information is nondeterministic, then a static control logic fault definition derived based on the service requirements and component functional information of the AHU at some specific time point is likely to be inaccurate.
- Representation of service and control information: in the developed formalism, the service requirements of the AHU are specified in terms of set points (static numeric values or data point variable names) of different air regulation functions (e.g. supply air heating). In order for the formalism to be applicable to an AHU, the service requirements of this AHU need to be specified in this manner. The formalism is also designed based on the idea that the actuator of an AHU component is associated with one control command with its actuation range represented by two saturation limits (max and min).

- The reasoning of control logic fault definition is based on general control goals for operating AHUs. This requires that these goals do not change. For example, for the purpose of maximizing energy efficiency, in an AHU cooling scenario, economizer cooling, if functional, is always preferred over cooling coil based cooling because it is more energy efficient. Thus, if a more advanced control method dictates a dynamic compliance with general control goals, e.g. energy efficiency goal shall be compromised during some specific scenario, then this will cause the control logic fault definition provided by the proposed research ineffective.

- Single AHU scope: our formalism is designed with the scope of a single AHU. That is, we view a single AHU as the subject of application and view it as a multi-input multi-output (MIMO) system following general control goals with high-level service requirements specified. In current practice, we observe the existence of AHUs controlled in coordination with other systems at optimal control level. Our formalism can still apply to these scenarios as optimal control is one level above the focus of this formalism, i.e. supervisory and local control. In the future, if the coordinated control of multiple systems are introduced into supervisory control level, and the service requirements cannot be specified within the scope of one AHU, then our proposed formalism will not be able to apply to these scenarios.

To clarify, the applicability of the proposed approach does not require the control logic being written explicitly as white-box program code. For example, the control logic can be implemented as a neural network model and the control logic fault definition formalism would still be applicable as long as the neural network model based control logic falls within the boundaries discussed above.

2.8 Conclusions and future work

This chapter highlights the need of systematically defining customized HVAC control logic faults. To address this challenge, we developed an object-oriented classification formalism that contains an HVAC component and control information ontology and a two-layer inference engine. The formalism takes factual information about one AHU system as the input and provides a set of control logic fault definition applicable to that AHU, specified in terms of control logic input/output variable expressions that can be directly used for control logic fault detection. Four general control goals of AHUs are identified and implemented in the system as the starting point. Our formalism is able to derive control logic fault definition indicating potential violations of these control goals applicable to specific AHU systems with an average precision of 94.2% and an average recall of 83.0 % when testing on 27 different AHUs specified by ASHRAE [1].

Future work of this research we recognized includes: 1) Studying inference of component functionality and control information through techniques such as topological reasoning to alleviate the need of user's efforts to specify information required in the ontology; 2) studying the potential of defining control logic faults for more operation modes besides normal operation mode, through the consideration of new goals and functionalities; 3) investigating the applicability of the developed system to other HVAC equipment, such as boilers and chillers.

Postamble

In this chapter, I present the research work addressing the first research question discussed in Chapter 1, structured as a self-contained journal paper draft.

This research targeted the need of a formalism to derive unambiguous control logic fault definition that is applicable, i.e. with customized contents, for specific AHU systems. In current practice, in order to decide what potential control logic faults should be checked in a specific AHU, people who evaluate the AHU have to subjectively adapt "rules of thumb" instructions from existing HVAC control guidelines with their own interpretations and brainstorm what the applicable types of faults are for the specific AHU in an ad-hoc manner. With the formalism I developed in this research, instead of ad-hoc brainstorming, the user only needs to provide factual information about the component availability, control information, and service requirements of the AHU in a standard format specified by the developed ontology. With this information, the proposed fault definition formalism is able to derive a set of control logic faults in terms of control logic input/output variable expressions that is unambiguous and can be applied for fault detection directly.

I validate the generality of the proposed formalism, i.e. the ability to accommodate different AHUs, by testing the developed prototype against 27 AHUs specified by ASHRAE [1]. The results show that the prototype derived customized control logic fault definitions for these 27 AHUs with an average precision of 94.2% and an average recall of 83.0 %.

The contributions of this research is summarized into the following two points:

- A Formalism of defining applicable AHU control logic faults based on system-specific information.
- An AHU component and control ontology that specifies the information requirements for defining control logic faults.

The potential industrial deployment of the formalism developed in this research is expected to greatly facilitate the deployment of automatic FDD for AHUs by addressing the need of a formalized approach to systematically define control logic faults.

Chapter 3

AHU Control Logic Fault Diagnosis with Software Fault Localization Algorithms

Software faults in the control logic programs for heating, ventilation, and air conditioning (HVAC) systems are prevalent and cause significant energy waste and building occupancy comfort compromises. This chapter presents a framework that incorporates software engineering fault localization techniques to diagnose control logic fault causes. Performance of 39 existing spectrum-based and mutation-based fault localization algorithms are evaluated using 11 real-world control logic fault cases from 2 air handling unit (AHU) test beds. The results show that the mutation-based Metallaxis fault localization technique outperforms all other fault localization techniques in diagnosing control logic faults. We characterize the fault cases in the test beds in terms of causes of control logic faults and software code to fix the faults, to show the diversity of the fault cases in the test beds and to study the relationships between control logic fault characteristics and fault localization performance of Metallaxis. We categorize control logic fault fixes in terms of mutation operations and make suggestions of mutation operators to consider for diagnosing control logic faults. Additionally, we conduct sensitivity analysis of multiple aspects of fault localization scenario setup options, including test suite size, test case selection down sampling methods and failed test case rate in test suite. The results show that Metallaxis computation provides peak fault localization performance with efficient computation time when it: 1) utilizes a test suite of size 100, down sampled via clustering algorithms and 2) adopts failed test cases ratio ranging from 60% to 90% in the test suite.

3.1 Introduction

Building heating, ventilation, and air conditioning (HVAC) system control logic programs are software programs implemented in direct digital controllers (DDC) to define the operating behavior of HVAC hardware systems. Control logic faults are implementation errors of the control logic program, causing inefficient system operation (e.g., simultaneous heating and cooling in air handling units (AHUs)[1]) and compromises in building service (e.g., insufficient heating when heating needs accumulates in AHUs). More than 15% of building HVAC system problems are attributable to control logic faults [4, 50], and they are estimated to be responsible for around 12 trillion BTU of energy wasted each year in the United States alone [6].

Multiple review studies (e.g. [11, 128]) have reiterated that existing computational HVAC fault detection and diagnosis (FDD) approaches are not effective in diagnosing HVAC control logic faults. First, existing FDD studies mostly focus

[1]We specifically mean preheat coil heating and cooling coil cooling in this example.

on hardware faults or a combination of hardware and a handful of software faults. Second, most FDD tools developed to date focus on fault detection and they rely heavily on user input to diagnose the fault effectively [11]. Third, FDD methods yield a number of possible causes for a specific fault, and more work is needed to ascertain the cause of the faults [128].

Assuming access to control logic input and output data through building automation system (BAS), when an HVAC control logic fault exists, its symptoms can be observed from BAS data readings related to control logic input and output variables. In general, existing HVAC FDD approaches do not provide effective computational support to locate the fault causes inside control logic programs and locating the cause of the fault requires manual diagnosis [11]. The operation logic of multiple components within an HVAC system is highly interactive in the sense that different blocks of the logic are interdependent, and the control logic is executed iteratively with stateful variables (variable values from the previous execution of the control logic are needed in the current execution). Manual diagnosis of possible causes of control logic faults requires interpretation of such control logic programs and reasoning of logic execution information from input/output data, which is error-prone and time-consuming due to the human cognition barriers.

Computational software fault localization techniques, such as spectrum-based fault localization techniques [2], and more recently, mutation-based fault localization techniques [129, 130], have shown their value in studies from software engineering domain. In this chapter, we investigate the feasibility and effectiveness of utilizing these software fault localization techniques to help diagnose control logic faults in HVAC systems. We specifically focused on AHU systems, since control logic faults are more frequently found in them [131] and they are one of the most important and prevalent types of equipment in commercial building central HVAC systems.

In order to computationally diagnose control logic faults with software fault localization techniques, a quantitative specification of the control logic fault definition is required. In this study, we utilize the system-specific control logic fault definition generated by Chapter 2, which specifies control logic fault definition about violations of general control objectives (e.g. energy efficiency maximization) during system normal operation mode in terms of control logic program input/output variable expressions.

Multiple studies (e.g. [2, 3, 129, 130]) have evaluated the effectiveness of these software fault localization techniques. However, the software programs used in these studies for fault localization evaluation are either small programs with very simple control flow[2] and data dependency[3], or software applications of large size (tens to hundreds of thousands of lines of code (LOC)). Control logic programs of AHU systems have their unique features when comparing with either of these programs: 1) AHU control logic programs are relatively small in size (hundreds of LOC) and have relatively many (dozens of) input and output variables; 2) The whole AHU control logic program is executed repeatedly (like under a loop statement) with stateful variable values and stateful functions; 3) All AHU control logic faults are specified as multiple expressions of input/output variables inequality expressions satisfied at the same time [13, 12]. Since the control logic of real-world AHUs are too complicated to be used as an illustrative example, here we provide an example with a fictional AHU to illustrate the aforementioned features: assuming an AHU is serving supply air of required temperature range with two components, a preheat

[2]Control flow is the order in which individual statements of a program are executed.

[3]Data dependency is a situation in which a program statement refers to the data of a preceding statement.

coil and a cooling coil, the pseudo control logic of this AHU is shown below[4]. A control logic fault for this AHU is defined as "(cooling coil command $>$ 0) and (preheat coil command $>$ 0)" to indicate the inefficient operation of simultaneous heating and cooling.

```
while AHU is in normal operation mode, repeat the following:
    cooling coil command = feedback control (cooling loop set point,
        supply air temperature sensor, cooling control direction, cooling
        output range, cooling previous error, cooling loop PID gains)
    preheat coil command = feedback control (preheat loop set point,
        supply air temperature sensor, heating control direction, preheat
        output range, preheat previous error, cooling loop PID gains)
    sleep for 5 minutes
```

Each of the aforementioned three unique features of AHU control logic programs may have an impact on the performance of fault localization computation. The combination of these features makes the control logic programs different from typical software engineering applications. Thus, a dedicated performance evaluation is needed to study the application of software fault localization techniques on AHU control logic program fault diagnosis.

In general, this chapter aims at making the following contributions:

- Develop a framework of casting AHU control logic fault diagnosis problem into a software fault localization task
- Evaluate performance of spectrum-based and mutation-based fault localization

[4]the control logic shown in the pseudo code and the component constitutes it implies is intentionally oversimplified for a clear illustration. Control logic programs of real-world AHUs is significantly more complex

algorithms for the purpose of locating AHU control logic fault causes and study the relationships between characteristics of control logic faults and fault localization performance.

- Evaluate the mutation operators' impact on fault localization and make mutation operators suggestions based on analysis of actual fault fixes patterns.

- Conduct sensitivity analysis of multiple aspects of setup options of AHU control logic fault localization computation and provide guidelines for improving HVAC control logic fault diagnosis performance using fault localization.

The remainder of the chapter is organized as follows. Section 3.2 provides brief introduction of existing HVAC fault diagnosis approaches and software fault localization. Section 3.3 describes in detail the fault localization algorithms and the corresponding evaluation metrics we considered in this chapter. Section 3.4 presents the framework we formulated to adopt software fault localization for control logic fault diagnosis. After discussing the test beds and fault cases we developed for this study in Section 3.5, we present our research work on fault localization evaluation, mutation operators suggestion, and setup exploration in Section 3.6, Section 3.7, and Section 3.8 respectively. Finally, we conclude the chapter with discussions about the implications of our findings, limitations and future work in Section 3.9.

3.2 Research background

3.2.1 HVAC fault diagnosis approaches

A recent review of HVAC FDD studies [46] summarizes 89 HVAC automated FDD methods. Within the 89 FDD studies, 34 studies involving systematic fault diagnosis approaches[5] are listed in Table 3.1. The targeted systems and focusing faults of these studies are provided [46] and also listed in the table.

As shown in Table 3.1, among the 34 existing HVAC FDD studies that performed systematic fault diagnosis tasks to find out the causes of the detected faults, the majority adopted expert rules that mirror human reasoning processes, while the others used statistical and machine learning methods (solely or together with expert rules) to analyze BAS data to infer the possible causes of the faults. These identified HVAC fault diagnosis studies mostly focused on hardware faults, such as sensor failures and stuck valves, while only a few studies considered software related faults at a high-level (i.e. consider "control logic fault" as one type of fault). We have not identified any HVAC fault diagnosis research that focuses on a general and formalized approach of locating control logic fault causes. This chapter aims at filling this gap by adopting software fault localization techniques in the HVAC FDD domain and evaluating their performance of diagnosing the fault causes inside control logic programs.

3.2.2 Software fault localization

In the software engineering domain, software testing is the process of making experiments on the software to evaluate its properties. When testing a piece of software program, the program under test is executed with a set of test case inputs, and the results of these executions, i.e. behavior of the program under test case conditions, are checked by test assertions [158] that implement the requirements of this program. If a test case input/output pair satisfies an assertion rule, then regarding this rule, it is a passed test case. Otherwise, it is a failed test case.

[5]The authors consider a fault diagnosis approach being systematic if the approach is repeatable by following the documented process, and the fault diagnosis results are reproducible and not experience dependent

Publication	**System**	**Fault diagnosis method**
Castro and Vaezi-Nejad [43]	AHU	Functional testing
Wang et al. [132]	AHU	Expert rules
Yang et al. [70]	AHU	Expert rules
Fernandez et al. [72]	AHU	Expert rules
Fernandez et al. [133]	AHU	Expert rules
Brambley et al. [134]	AHU	Expert rules
Wang et al. [75]	VAV	Expert rules
Wang et al. [40]	VAV	Expert rules
Li et al. [135]	AHU	Expert rules
Lee et al. [76]	AHU	Expert rules
Wang and Jiang [136]	AHU	Expert rules
Wang and Xiao [137]	AHU	SPE (Squared Prediction Mean)
Wang and Xiao [138]	AHU	SPE, Expert rules
Qin and Wang [139]	AHU	T-statistics, SPE
Wang and Qin [140]	AHU	T-statistics, SPE, VRE (Variance Reconstruction Error)
Hou et al. [141]	AHU	Expert rules
Jin and Du [142]	AHU	SPE, JAA (Joint Angle Analysis)
Xiao et al. [143]	AHU	SPE contribution plot, expert rules
Du et al. [144]	AHU	SPE plots, JAA
Du et al. [145]	AHU	FDA (Fisher Discriminant Analysis)
Du and Jin [146]	AHU	SPE plots, JAA
Du and Jin [147]	AHU	SPE plots, JAA
Du and Jin [148]	AHU	FDA
Du et al. [149]	AHU	Wavelet analysis
Du et al. [150]	VAV	SPE plots, JAA
Li [78]	VAV	Wavelet analysis, pattern matching
Xiao et al. [151]	VAV	Expert-based multivariate decoupling
Fan et al. [152]	AHU	Wavelet analysis, ENN (Elmann Neural Network), FCM (Fuzzy C-Mean) clustering
Chen and Lan [153]	AHU	SPE, SVI (sensor validity index)
West and Guo [79]	AHU	Expert rules
Yang et al. [154]	AHU	Statistical residual
Yang et al. [155]	AHU	Statistical residual
Li and Wen [156]	AHU	Wavelet analysis
Li and Wen [157]	AHU	Pattern matching

Table 3.1: HVAC FDD studies with fault diagnosis methods

Among various software testing techniques, computational fault localization techniques provide suggestions of suspicious code that are responsible for causing a software fault observed from test cases execution results. The domain of software fault localization is best depicted by three recent survey papers [2, 159, 160]. Among these three survey papers, Wong et al. [2] provided the most comprehensive overview of different categories of fault localization techniques, in which software fault localization techniques were grouped into eight categories, as listed in Table 3.2. Among these 8 categories, spectrum-based fault localization techniques are selected over other techniques for evaluation in HVAC control logic fault diagnosis because:

1. Spectrum-based techniques are the most prevalent type of method used nowadays in the software engineering domain, account for 35% of the reviewed publications [2], and it is well-known due to its efficiency (easy to compute) and effectiveness [161] [6].

2. Spectrum-based techniques do not require ad-hoc modeling of the software under test into another form other than its source code. In other words, control logic program code is the only artifact of the software needed to apply spectrum-based techniques.

[6]The definition of effectiveness is discussed in detail in Section 3.3

Fault localization technique category	**Description**
Slice-based techniques	Program slicing abstracts a program into a reduced form by deleting irrelevant parts
Program spectrum-based techniques	A program spectrum details the execution information of a program and can be used to track program behavior
Statistics-based techniques	Isolating bugs in programs with instrumented predicates at particular points and rank the suspiciousness predicates
Program state-based techniques	A program state consists of variables and their values at a particular point during program execution, which can be a good indicator for locating faults
Machine learning-based techniques	In the context of fault localization, the program can be identified as trying to learn or deduce the location of a fault
Data mining-based techniques	The software fault localization problem is abstracted to a data mining problem – identify the pattern of statement execution
Model-based techniques	Assuming a correct model of each program being diagnosed is available and using the differences between the observed program model
Other techniques	Techniques do not below to above categories, many of which focus on specific programming languages or testing scenarios

Table 3.2: Software fault localization techniques categorization by Wong et al. [2]

Mutation-based fault localization techniques [129, 130] are a new type of software fault localization techniques that has gained traction very recently, and are not contained in the survey papers [2, 159, 160]. While spectrum-based techniques try to use the connections between test execution results (passed or failed) and the binary test execution profile (whether a statement is executed by a test case) to find the suspicious statements, mutation-based techniques try to gain more insights about the software through executing test cases on different versions of the software (mutants, generated by modifying the original program), instead of executing test cases only on the original software, which is the case for spectrum-based techniques. Thus, although mutation-based techniques are computationally expensive due to the needs of executing test suites under plenty of mutant programs, they are expected to provide better fault localization results than spectrum-based fault localization techniques [129, 130].

In this chapter, existing spectrum-based and mutation-based techniques are utilized and their ability to localize HVAC control logic faults are evaluated. These fault localization techniques, their outputs and evaluation metrics are discussed in detail in Section 3.3.

3.3 Fault localization methods

This section provides a detailed discussion about software fault localization techniques utilized in this chapter. The first two subsections provide the algorithms and explanations of the two types of fault localization techniques of our focus respectively. The third subsection focuses on the discussion of fault localization performance metrics.

3.3.1 Spectrum-based methods

The idea of spectrum-based fault localization techniques is as follows: program statements that are more likely to be executed by failed test cases than successful test cases are more suspicious to be the cause of the software fault. Plenty of spectrum-based techniques have been proposed and different techniques implement this same general idea with different suspiciousness computation formulas.

We recognize a list of existing spectrum-based fault localization techniques by complementing the survey conducted by Wong et al. [2] with three additional techniques (Op2, Barinel and D^2) evaluated by Pearson et al. [3]. Using the same notations used in [2] (Table 3.3), the full list of spectrum-based fault localization techniques we evaluated are detailed in Table 3.4.

Notation	**Description**
N_{CF}	number of failed test cases that executed the statement
N_{UF}	number of failed test cases that did not execute the statement
N_{CS}	number of successful test cases that executed the statement
N_{US}	number of successful test cases that did not execute the statement
N_C	total number of test cases that executed the statement
N_U	total number of test cases that did not execute the statement
N_S	total number of successful test cases
N_F	total number of failed test cases
$n = N_{CF} + N_{UF} + N_{CS} + N_{US}$	

Table 3.3: Notations used in Table 3.4

Table 3.4: Spectrum-based fault localization techniques [2, 3] evaluated in this study

No.	**Spectrum-based technique name**	**Statement suspiciousness calculation formula**
1	Tarantula	$\frac{N_{CF}/N_F}{N_{CF}/N_F+N_{CS}/N_S}$
2	Ochiai	$\frac{N_{CF}}{\sqrt{N_F \times (N_{CF}+N_{CS})}}$
3	Ochiai2	$\frac{N_{CF} \times N_{US}}{\sqrt{(N_{CF}+N_{CS}) \times (N_{US}+N_{UF}) \times (N_{CF}+N_{UF}) \times (N_{CS}+N_{US})}}$
4	Op2	$N_{CF} - \frac{N_{CS}}{(N_S+1)}$
5	Barinel	$1 - \frac{N_{CS}}{N_{CS}+N_{CF}}$
6	D^2 (DStar)	$\frac{N_{CF}^2}{N_{CS}+(N_F-N_{CF})}$

Table 3.4 continued from previous page

No.	Spectrum-based technique name	Statement suspiciousness calculation formula
7	Braun-Banquet	$\frac{N_{CF}}{max(N_{CF}+N_{CS},N_{CF}+N_{UF})}$
8	Dennis	$\frac{(N_{CF}\times N_{US})-(N_{CS}\times N_{UF})}{\sqrt{n\times(N_{CF}+N_{CS})\times(N_{CF}+N_{UF})}}$
9	Mountford	$\frac{N_{CF}}{0.5\times((N_{CF}\times N_{CS})+(N_{CF}\times N_{UF}))+(N_{CS}\times N_{UF})}$
10	Fossum	$\frac{n\times(N_{CF}-0.5)^2}{(N_{CF}+N_{CS})\times(N_{CF}+N_{UF})}$
11	Pearson	$\frac{n\times((N_{CF}\times N_{US})-(N_{CS}\times N_{UF}))^2}{N_C\times N_U\times N_S\times N_F}$
12	Gower	$\frac{N_{CF}+N_{US}}{\sqrt{N_F\times N_C\times N_U\times N_S}}$
13	Michael	$\frac{4\times((N_{CF}\times N_{US})-(N_{CS}\times N_{UF}))}{(N_{CF}+N_{US})^2+(N_{CS}+N_{UF})^2}$
14	Pierce	$\frac{(N_{CF}\times N_{UF})+(N_{UF}\times N_{CS})}{(N_{CF}\times N_{UF})+(2\times(N_{UF}\times N_{US}))+(N_{CS}\times N_{US})}$
15	Baroni-Urbani & Buser	$\frac{\sqrt{(N_{CF}\times N_{US})}+N_{CF}}{\sqrt{(N_{CF}\times N_{US})}+N_{CF}+N_{CS}+N_{UF}}$
16	Tarwid	$\frac{(n\times N_{CF})-(N_F\times N_C)}{(n\times N_{CF})+(N_F\times N_C)}$
17	Ample	$\left\vert\frac{N_{CF}}{N_{CF}+N_{UF}}-\frac{N_{CS}}{N_{CS}+N_{US}}\right\vert$
18	Phi (Geometric Mean)	$\frac{N_{CF}\times N_{US}-N_{UF}\times N_{CS}}{\sqrt{(N_{CF}+N_{CS})\times(N_{CF}+N_{UF})\times(N_{CS}+N_{US})\times(N_{UF}+N_{US})}}$
19	Arithmetic Mean	$\frac{2\times(N_{CF}\times N_{US}-N_{UF}\times N_{CS})}{(N_{CF}+N_{CS})\times(N_{US}+N_{UF})+(N_{CF}+N_{UF})\times(N_{CS}+N_{US})}$
20	Cohen	$\frac{2\times(N_{CF}\times N_{US}-N_{UF}\times N_{CS})}{(N_{CF}+N_{CS})\times(N_{US}+N_{CS})+(N_{CF}+N_{UF})\times(N_{UF}+N_{US})}$
21	Fleiss	$\frac{4\times(N_{CF}\times N_{US}-N_{UF}\times N_{CS})-(N_{UF}\times N_{CS})^2}{(2N_{CF}+N_{UF}+N_{CS})+(2N_{US}+N_{UF}+N_{CS})}$
22	Zoltar	$\frac{N_{CF}}{N_{CF}+N_{UF}+N_{CS}+\frac{10000\times N_{UF}\times N_{CS}}{N_{CF}}}$
23	Harmonic Mean	$\frac{(N_{CF}\times N_{US}-N_{UF}\times N_{CS})((N_{CF}+N_{CS})\times(N_{US}+N_{UF})+(N_{CF}+N_{UF})\times(N_{CS}+N_{US}))}{(N_{CF}+N_{CS})\times(N_{US}+N_{UF})\times(N_{CF}+N_{UF})\times(N_{CS}+N_{US})}$
24	Rogot2	$\frac{1}{4}(\frac{N_{CF}}{N_{CF}+N_{CS}}+\frac{N_{CF}}{N_{CF}+N_{UF}}+\frac{N_{US}}{N_{US}+N_{CS}}+\frac{N_{US}}{N_{US}+N_{UF}})$
25	Simple Matching	$\frac{N_{CF}+N_{US}}{N_{CF}+N_{CS}+N_{US}+N_{UF}}$

Table 3.4 continued from previous page

No.	Spectrum-based technique name	Statement suspiciousness calculation formula
26	Rogers & Tanimoto	$\frac{N_{CF}+N_{US}}{N_{CF}+N_{US}+2(N_{UF}+N_{CS})}$
27	Hamming	$N_{CF}+N_{US}$
28	Hamann	$\frac{N_{CF}+N_{US}-N_{UF}-N_{CS}}{N_{CF}+N_{UF}+N_{CS}+N_{US}}$
29	Sokal	$\frac{2(N_{CF}+N_{US})}{2(N_{CF}+N_{US})+N_{UF}+N_{CS}}$
30	Scott	$\frac{4(N_{CF}\times N_{US}-N_{UF}\times N_{CS})-(N_{UF}-N_{CS})^2}{(2N_{CF}+N_{UF}+N_{CS})(2N_{US}+N_{UF}+N_{CS})}$
31	Rogot1	$\frac{1}{2}(\frac{N_{CF}}{2N_{CF}+N_{UF}+N_{CS}}+\frac{N_{US}}{2N_{US}+N_{UF}+N_{CS}})$
32	Kulczynski	$\frac{N_{CF}}{N_{UF}+N_{CS}}$
33	Anderberg	$\frac{N_{CF}}{N_{CF}+2(N_{UF}+N_{CS})}$
34	Dice	$\frac{2N_{CF}}{N_{CF}+N_{UF}+N_{CS}}$
35	Goodman	$\frac{2N_{CF}-N_{UF}-N_{CS}}{2N_{CF}+N_{UF}+N_{CS}}$
36	Jaccard	$\frac{N_{CF}}{N_{CF}+N_{UF}+N_{CS}}$
37	Sorensen-Dice	$\frac{2N_{CF}}{2N_{CF}+N_{UF}+N_{CS}}$

3.3.2 Mutation-based methods

Two mutation-based techniques have been proposed, MUSE [129] and Metallaxis [130]. Both techniques leverage the power of mutants (different versions of the original software program, each has a single and unique difference from the original program), but their underlying ideas are different. MUSE utilizes the knowledge of how mutants can change the test results, i.e. making failed test cases pass or making passed test cases fail, assuming that mutating faulty statements will make more failed test cases pass than mutating correct statements, and mutating correct statements will make more passed test cases fail than mutating faulty statements.

In contrast, Metallaxis utilizes the knowledge of how mutants and original program output differently under the same test cases. In the software testing domain terminologies, a test case "kills" a mutant if executing the test case on the mutant yields a different test output than executing it on the original program. The intuition of Metallaxis is that mutants generated by mutating faulty statements are more likely to be killed by test cases that are failed on the original program because mutants and faults located on the same program statements frequently exhibit a similar behavior. The computations of MUSE and Metallaxis are detailed below.

MUSE

Moon et al. [129] proposed the MUSE suspiciousness calculation of statement (s) suspiciousness as:

$$\frac{1}{|mut(s)|}\sum_{m\in mut(s)}(\frac{|f_P(s)\cap p_m|}{|f_P|} - \alpha \times \frac{|p_P(s)\cap f_m|}{|p_P|}),\ \alpha = \frac{f2p}{|f_P|} \times \frac{|p_P|}{p2f}$$

where:

$f_P(s)$ - test cases executed s and failed on the original program

$p_P(s)$ - test cases executed s and passed on the original program

$mut(s) = m_1, ..., m_k$ - all mutants of statement s with observed changes in test results (i.e. killed by at least one test case)

f_m - test cases failed on mutant m

p_m - test cases passed on mutant m

f_P - test cases failed on the original program

p_P - test cases passed on the original program

$p2f$ - total number of instances when a test case passed on the original program and failed on a considered mutant

$f2p$ - total number of instances when a test case failed on the original program and passed on a considered mutant

As noted in [129], MUSE selects only a subset of statements of the original program to mutate. These statements are executed by at least one failed test case.

Metallaxis

Papadakis and Le Traon [130] proposed the Metallaxis suspiciousness calculation of mutant (e) as:

$$\frac{N_{KF}}{\sqrt{N_F \times (N_{KF} + N_{KS})}}$$

where:

N_{KF} - number of failed test cases that kill mutant e

N_{KS} - number of passed test cases that kill mutant e

N_F - total number of failed test cases

Metallaxis calculates a dedicated suspicious value for each mutant, instead of for each statement, like spectrum-based techniques and MUSE. In Metallaxis, the suspiciousness value of a statement is decided as the maximum of all suspiciousness values of mutants that mutated this statement in the original program.

3.3.3 Fault localization metrics

Both spectrum-based and mutation-based fault localization techniques compute a numerical suspiciousness value for each statement of the program source code. Then they output a ranked list of these statements according to the calculated suspiciousness values in descending order. These techniques suggest a programmer to manually examine the statements one by one according to the ranked list and assume that the programmer will identify the fault once the actual fault-causing statement is manually examined. This assumption is commonly referred as the "perfect bug understanding assumption" in existing software fault localization studies [162]. Although this is obviously an over-simplified assumption of how the programmer can debug the program, this assumption is widely adopted in software fault localization studies and facilitates a set of simple effectiveness metrics based on the ranked list to evaluate the effectiveness of fault localization techniques. The most popular example of these metrics is the $EXAM$ score [2], which represents the percentage of statements in a program that needs to be manually examined until the faulty statement is reached.

Various empirical studies [162, 163, 164, 165] have conducted user studies to investigate the usefulness of fault localization techniques' ranked list outputs to programmers in real-world. They found that programmers locate bugs significantly faster with the fault localization tool, and most programmers deemed the fault localization tool useful. Although these user studies mostly focused on investigating the usefulness of only spectrum-based fault localization techniques, mutation-based fault localization techniques gave the result in the same form (ranked list based on statement suspiciousness) with potentially better quality (faulty statement ranked higher in the list), so it can be inferred that mutation-based fault localization techniques should be as useful as spectrum-based fault localization techniques, if not more useful, to practitioners [164].

Before discussing the specific metric we used in this study to evaluate the fault localization performance for HVAC control logic fault diagnosis, we clarify three issues about the faulty statement in the ranked list: *multiple faulty statements*, *tie*, and *absolute ranking vs. percentage ranking.*

Multiple faulty statements

It is very common that more than one statements in the original program need to be modified, deleted, or added, in order to fix a software fault. Thus, a fault may have multiple faulty statement locations. For the purpose of evaluating fault localization methods, there have been metrics taking either the best result (highest ranking faulty statement), the worst result (lowest ranking faulty statement), or the average [166]. In our study, we use the best result in our evaluation metric for two reasons:

- Empirical studies of fault localization uses in practice [164, 165] indicated that while the perfect bug understanding assumption is false, fault localization results provide valuable hints to the programmer to guide them in the creation of fault fixing hypotheses. The focus is to help programmers find a good starting point to initiate the bug-fixing process rather than to provide the complete set of code that must be modified, deleted, or added [2]. Examining one faulty statement instead of all supports this use.
- Widely used metrics, such as the $EXAM$ score and T score, use the best result as well [2].

Tie

For both spectrum-based and mutation-based fault localization results, it is common to have multiple statements assigned the same suspiciousness value. Statements with the same suspiciousness value will result in ties in the ranking [2]. If the highest ranking faulty statement has tie statements with the same ranking, we adopt the average ranking of these tied statements in our evaluation metric, as the average is the expected number of statements the programmer should examine assuming that the programmer chooses these tie statements at random. This strategy is also adopted by other studies [3, 167].

Absolute ranking vs. percentage ranking

Although percentage ranking metrics, such as the $EXAM$ score, are widely used in existing fault localization studies, recently, more and more empirical studies have suggested the adoption of absolute ranking metrics. Parnin and Orso [162] suggested that fault localization techniques should focus on improving absolute rank rather than percentage rank for two reasons: 1) their collected data indicated that programmers will stop inspecting statements and transition to debugging without the fault localization tool's assistance if they do not get promising results within the first few statements they inspect; 2) the use of percentages underscores how difficult the problem becomes when moving to larger size programs. Among the participants of the user study conducted by Kochhar et al. [163], 74% of the participants think that the fault localization is successful if the faulty statement is among the top-5 ranking, and 98% of the participants think that inspecting more than 10 statements is beyond their acceptability level. Xia et al. [164] found that 75% of their study's participants visit the top-5 suspicious statements in sequence, and if the faulty statement is in the top-5 suspicious statements, the participants debug faster than if the faulty statement is ranked at a position between 6 and 10. Souza [165] found that developers navigate more frequently among the well-ranked statement of the ranked list and most developers did not inspect statements below the top-20 when using fault localization techniques. Based on this empirical evidence, we adopt the absolute ranking in our evaluation metric, and check if the faulty statements are within top-5, top-10 and top-20 ranked list in our fault localization results.

As a result, we evaluate our fault localization results with a single-value metric, referred to as *fl-effectiveness* hereafter, representing the expected absolute number of statements a programmer needs to manually examine following the fault localization ranked list, until he/she reaches the first actual faulty statement.

3.4 Control logic fault diagnosis framework and implementation

Following the software unit testing paradigm [28], we formulate the AHU control logic fault diagnosis process as a computation framework shown in Figure 3.1, in which spectrum-based and/or mutation-based fault localization techniques are utilized.

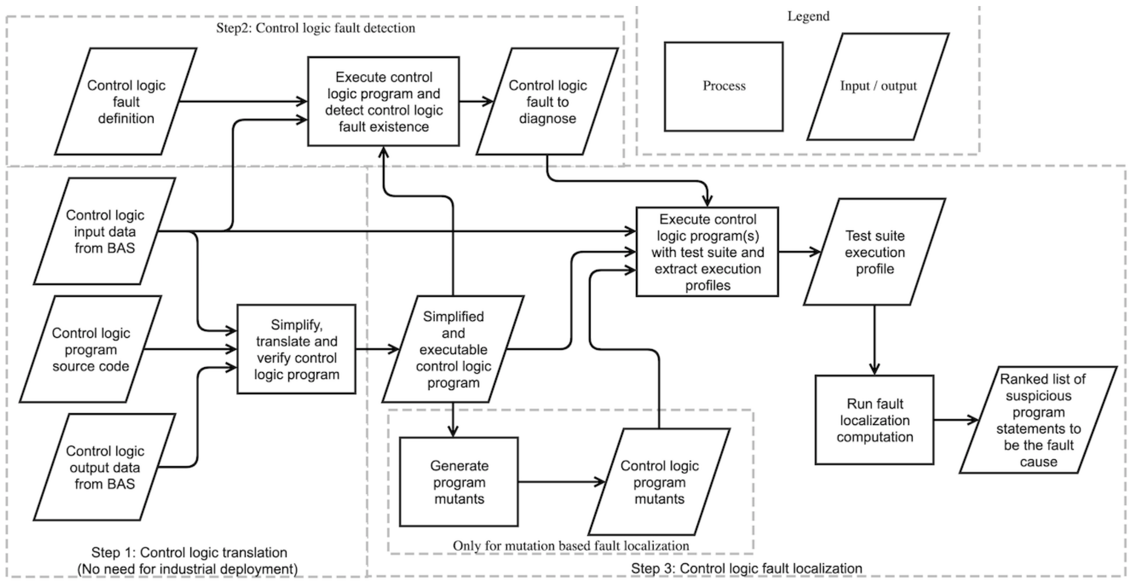
Step2: Control logic fault detection
Control logic fault definition
Execute control logic program and detect control logic fault existence
Control logic fault to diagnose
Legend
Process
Input / output
Control logic input data from BAS
Execute control logic program(s) with test suite and extract execution profiles
Test suite execution profile
Control logic program source code
Simplify, translate and verify control logic program
Simplified and executable control logic program
Run fault localization computation
Ranked list of suspicious program statements to be the fault cause
Control logic output data from BAS
Generate program mutants
Control logic program mutants
Only for mutation based fault localization
Step 1: Control logic translation
(No need for industrial deployment)
Step 3: Control logic fault localization

Figure 3.1: Control logic fault diagnosis framework

Under this framework, the artifacts needed from the HVAC system domain are AHU control logic program source code (typically in manufactures' specific format) and a BAS dataset including control logic input/output variables data points. The control logic fault definition can be acquired with the fault definition approach proposed in Chapter 2.

The control logic program of an AHU system asks for multiple input variable values about sensor readings and user settings, then it repeatedly computes multiple output variable values about actuator commands at a certain time interval. This is similar to placing the control logic program under an infinite loop condition. In order to utilize the fault localization algorithms, the control logic program cannot be executed with such infinite loops. Hence, the control logic program is simplified to eliminate such looping behavior by: 1) modifying all proportional-integral (PI) controllers to be proportional (P) controllers; 2) rearranging statements to avoid the instances of a variable being called before its value being assigned; 3) in some occasion when such an instance cannot be avoided by rearranging statements, adding this variable as a control logic input variable and reading its value from input data.

As discussed in Section 3.3, in order to conduct fault localization computation, the control logic program execution profiles need to be collected. This requires the execution of control logic program with instrumentation to extract execution information. In our framework, we assume that the original control logic program is written in manufactures' proprietary format and that its compiler and runtime environment are not disclosed to the public (this was the case in our test beds). In this situation, the control logic program needs to be translated [7] so that we can execute the control logic program and collect its execution information.

In this framework, the control logic output data from BAS is used to verify the correctness and accuracy of control logic program simplification and translation. It is worth noting that when the framework is being adopted in the industry, we envision that the control manufactures should be implementing this framework with their programming platform of choice and provide the fault localization functionality as a product or service to customers. Thus, the manual control logic simplification and translation should not be needed by then.

[7] rewrite the program source code into a different programming language syntax, with the same program semantics of the original program source code.

Control logic input data from the BAS dataset and its corresponding logic execution result is first used to detect the existence of control logic faults with the control logic fault definition list. Then the user of the framework shall select one existing fault based on the fault detection results for fault cause localization. For this selected fault, each input/output sample is marked as passed if it does not contain the symptom of the fault and failed otherwise.

After the dataset has been marked, it is treated as a test case suite and supplied to the fault localization algorithms together with the tests execution profiles. Finally, the fault localization algorithms compute the suspiciousness of each program statement of being the cause of the fault.

We implemented this framework in the Java programming language with the utilization of two existing software packages: Major framework [168] for generating program mutants and Tacoco [169] for generating JUnit tests execution profiles. [8]

3.5 Fault cases from real-world test beds

3.5.1 Test beds information

Two real-world AHU test beds, referred to as *PAM* and *DELTA* hereafter, are used in this study. We picked two AHUs from different buildings, with different system configurations, and from different manufactures, in order to acquire diverse control logic programs and control logic fault cases. Located in an academic building, the PAM AHU has steam heating coil, chilled water cooling coil, supply and return variable frequency drives (VFD) fans, and a mixing box with linked three dampers (outdoor air damper, mixed air damper, and exhaust air damper). Located in an office building, the DELTA AHU has hot water heating coil, direct expansion (DX)

[8]The implementation source code will be available at `https://github.com/leijerry888/` once the review process is over.

cooling coil, supply and relief VFD fans, a mixing box with linked two dampers (outdoor air damper and mixed air damper), and an independent exhaust air damper for space pressure control.

As shown in Figure 3.1, in order to apply software fault localization techniques on control logic programs, the control logic programs need to be executable and their execution information needs to be extractable from test executions. The PI controllers also need to be simplified as P controllers to avoid running the control logic with dynamic (looping) behavior. Each of the 2 test beds control logic programs is written in its manufacture's proprietary software. We manually translated and simplified the two control logic programs into Java methods, with the help of control programmers from the manufactures. One minute interval BAS datasets of normal operation containing all control logic input and output variables are collected for both test beds for the purpose of verifying control logic translation as well as facilitating the application of fault localization techniques (as indicated in Figure 3.1), i.e., each sample of BAS data is developed into a test case of the control logic program during software fault localization. We verified the accuracy of the control logic translation and simplification by simulating the translated programs with control logic input from the BAS dataset and verifying the simulation outputs with actual control logic output data from the BAS dataset. Basic information of these two test beds is provided in Table 3.5.

3.5.2 Fault cases information

In order to detect control logic faults, a control logic fault definition is needed. We utilized the control logic fault definition approach we proposed in Chapter 2 to derive a set of control logic fault definition for each of the two test beds according to their specific component, control and service requirement information. For the

Test bed AHU	**PAM**	**DELTA**
Control Manufacture	American Automatrix	Delta Controls
Original control logic format	BACnet objects with manufacture added properties	textual code
Original control logic access	American Automatrix NBPro software	Delta Controls web interface
Translated control logic program length	383 LOC	431 LOC
Main sequencing logic program length	189 LOC	215 LOC
# Mutants for main sequencing logic	336	693
# Control logic input variables	29	38
# Control logic output variables	13	21
Available BAS dataset size	28,800 Samples	5,951 Samples
Time span of BAS dataset	24 hour operation for 20 days	9 hour operation for 11 days

Table 3.5: Two AHU test beds information

PAM AHU, 16 control logic faults are defined, and for the DELTA AHU, 21 control logic faults are defined. All defined faults are associated with mathematical expressions about the control logic input/output variables that can be used as JUnit test assertions [170] during fault detection process. The BAS datasets and the program simulation outputs are used to detect the existence of control logic faults in the test beds, as indicated in Figure 3.1. Control logic faults existed in the two test beds that will be used in the following fault localization tasks are listed in Table 3.6.

3.5.3 Faulty statements identification

In order to identify the faulty statements (i.e. the coding errors causing control logic faults), we manually went through and interpreted the control logic programs, identified the fault causes, and fixed the faults. When generating the fault fixes, we follow the idea of finding the smallest change (i.e., add/delete/modify source code statements) of the program that represents the isolated fault fix [3, 171], while still complying with the control objectives of the AHU systems. After fixing each fault, we executed the fixed control logic program with all control logic input data from the BAS datasets to verify that all test cases passed for that fault. Then, for each AHU test bed, we merged together all the fault fixes to generate a control logic program and verify it with the execution of all test inputs again to make sure that

Fault ID	**Fault detection results**		**Fault definition concise description**
	# Failed test case	**# Passed test case**	
PAMfault2	27,989	811	Mixing box cooling not fully utilized
PAMfault7	17,834	10,966	Preheat coil valve active when there is continuously no heating need
PAMfault9	163	28,637	Supply air fan speed larger than minimum when discharge air static pressure is continously above service set point
PAMfault10	25,505	3,295	Return air fan speed larger than minimum when return air flow is continuously above service set point
PAMfault15	17,589	11,211	Simultaneous economizer cooling and mechanical heating
DELTAfault1	3,199	2,752	Mixing box heating not fully utilized
DELTAfault2	169	5,782	Economizer cooling not fully utilized
DELTAfault3	3,345	2,606	Preheat coil heating not fully utilized
DELTAfault4	1,259	4,692	Mechanical cooling not fully utilized
DELTAfault11	2,679	3,272	Mechanical cooling active when there is continuously no cooling need
DELTAfault16	293	5,658	Mechanical cooling active before economizer cooling is fully utilized

Table 3.6: Fault cases used in this study

the control logic program is completely fault-free with regard to all defined control logic faults. Finally, we confirmed with HVAC control professionals the legitimacy of our control logic fault findings and fixes. Note that there can be more than one ways to fix a fault at different program locations, even following the same fault fixing ideas. In this case, we also identified alternate fault fixes directly linked them to the original fix. A fault shall be deemed fixed when either the original or one alternate fix is performed [3].

In the next two subsections, we characterize the 11 fault cases of our case study for two purposes: 1) benchmark the diversity of the faults considered in our evaluation, and 2) analyze the relationships between fault characteristics and fault localization performance. We conducted fault characterization in two different aspects: 1) causes of control logic faults, and 2) software code statements to fix the faults. The characterization methods and results are discussed below and fault characteristics' impact on fault localization performance is discussed in detail in Section 3.6.

3.5.4 Fault causes categorization

The 11 fault cases used for fault localization evaluation have various causes. To semantically characterize the control logic fault causes, we first discuss the semantic characterization of HVAC control logic. There are three levels of controls in AHU systems, namely, local level control, supervisory control, and optimal control [95, 172]. The optimal control level is out of our scope because the decisions of service set points are not questioned in the control logic fault definition, and thus not detected and diagnosed. The semantics of AHU control logic programs can be built based on the perspective of two levels of controls and the idea of hierarchical control loops. Two existing studies have been identified that developed semantic models of HVAC control logic [172, 92]. Chen [172] specified three levels of controls, i.e. building layer control module, system layer control module, and local layer control module. The meaning of these three levels of controls corresponds to optimal, supervisory, and local control. The three levels of controls contain "Control Modules" connected hierarchically through module inputs and outputs. Schneider [92] proposed an ontology for the semantic modeling of control logic programs, of which the central building blocks are "ControlActors" connected to each other through inputs and outputs. The ideas of "Control Module" and "ControlActor" associated with inputs and outputs is based on the theory of open/closed loop control through the Sense-Process-Actuate cycles [92].

We categorize the control logic fault causes according to which parts of the aforementioned control logic semantic model they are affecting, and how they are affecting them (e.g. missing certain elements, or incorrect value of certain elements). At local control level, the local controllers of components adjust their control output variables to track set points. Each local controller can be semantically modeled by further elaborating with elements that define a closed-loop feedback controller [173]: set point, output (actuation range and direction), feedback (only for closed-loop control). The supervisory control contains logic that coordinates the sequencing of different components. The supervisory controls are added on top of local controllers through constraints, such as conditional code that lockout/override/select local controllers under certain conditions. Based on this perspective, the fault causes of our 11 fault cases are categorized in Table 3.7. It is worth noting that some faults are caused by multiple different types of fault causes, thus they are being categorized into multiple categories.

TestBed	**PAM**					**DELTA**					
FaultID	**Fault2**	**Fault7**	**Fault9**	**Fault10**	**Fault15**	**Fault1**	**Fault2**	**Fault3**	**Fault4**	**Fault11**	**Fault16**
Fault cause at local control level											
Incorrect local controller set point	✓								✓	✓	
Incorrect local controller feedback	✓		✓		✓						
Incorrect local controller output mapping									✓	✓	
Missing local controller						✓					
Fault cause at supervisory control level											
Incorrect constraint expression(s)	✓	✓		✓	✓		✓	✓	✓	✓	
Missing constraint(s)											✓

Table 3.7: Fault cause categorization of 11 fault cases

3.5.5 Fault fixing statements categorization

As discussed in Section 3.5.3, one or more control logic program statements need to be changed (modified, inserted, or deleted) in order to fix one control logic fault, and there may be more than one ways of fixing each fault. For the 11 control logic faults we used in this study, a total of 68 changes of statements (i.e. fault fixing statements) are identified. In this subsection, we discuss the categorization of these 68 fault fixing statements.

Plenty of studies about software fault categorization have been published since the 1970s [174]. Although several fault categorization schemes exist, such as the Orthogonal Defect Classification (ODC) [175] and the fault taxonomy proposed in [176], there is no widely accepted, comprehensive and consistently used software fault characterization scheme [177], because different studies conducted fault categorization with a wide range of focus and motivations [178]. For example, 8 software defect types were defined in ODC [175], with focuses including general design issues and documentation problems, while 27 software bug fix types were defined by Pan et al. [179], all of which focused specifically on Java statement level code changes.

In this study, we leverage the software bug fix categorization scheme proposed by Zhao et al. [180], because of the following features of this scheme:

- It focuses on categorizing bug fixes at code statement level, which aligns with the focus of our faulty statement characterization.
- It is programming language independent [180], so the control logic fault fixing statements can be categorized accordingly even if they are written in other programming languages.
- It provides unambiguous description for classifying bug fixes, so we can categorize control logic fault fixing statements without confusions and uncertainties. Actually, Zhao et al. [180] developed a tool to automatically categorize fault fixing code based on this scheme for existing open source software projects.
- It provides bug fix frequencies of different categories from multiple software systems. We can compare this information with our control logic fault fixes

to identify the diversity of the control logic fault fixes in our 11 fault cases.

More specifically, we utilize the following 5 out of the 9 non-exclusive [9] bug fixing code change types:

1. Changes on data declaration/initialization
2. Changes on assignment statements
3. Changes on function call
4. Changes on branch statements
5. Moving statements (specified in the "Others" category in [180])

The other 4 code change types are about loops, function declaration/definition, return/goto statements, and preprocessor directives [180]. They do not apply to AHU control logic programs because the control logic programs do not contain these four types of statements.

Zhao et al. [180] classified more than 2,000 bug fixing statements in existing software projects and found that the first four categories listed above are the four most prevalent bug fix types.

Among the 68 fault fixing statements we identified, 12 are about adding new statements, 8 are about deleting statements, and the rest 48 are about modifying statements. The categorization of these 68 bug fixing statements are shown in Table 3.8, in which the diversity of our fault fix types is demonstrated, i.e., each of the four most prevalent bug fix types identified in [180] exists in our fault fixing statements.

The diversity of the fault fixing statements illustrated by Table 3.8 helps to make sure that our fault localization performance evaluation results, which will be

[9]One statement level code change can be classified into multiple categories

Faulty statement type	Data declaration / initialization	Assignment statements	Function call	Branch statements	Move statements	Total
Number of statement	18	42	26	19	8	68
Frequency over all bug fixing statements	26.5%	61.8%	38.2%	27.9%	11.8%	100.0%

Table 3.8: Fault fixing statements categorization results

shown in Section 3.6, do not overfit to a specific fault fixing statement type. We will also study the relationships between the fault fixing types and the fault localization performance in more detail in Section 3.6.

3.6 Evaluation of existing techniques

In Section 3.3, we discussed two types of fault localization techniques of our focus, namely spectrum-based and mutation-based techniques, and presented the computations of 39 existing algorithms of these two types. In order to evaluate the performance of these spectrum-based and mutation-based fault localization techniques for the purpose of AHU control logic fault diagnosis, we applied and evaluated these 39 fault localization algorithms on 11 real-world control logic fault cases from two AHU test beds, using our control logic fault diagnosis framework implementation. The evaluation results are shown in Table 3.9 with the metric, *fl-effectiveness*, articulated earlier. In Table 3.9, the first two rows are fault localization results of the 2 mutation-based techniques, and the following rows are results of 37 spectrum-based techniques, ordered according to Table 3.4.

3.6.1 Fault localization performance of considered algorithms

Fault localization results from Table 3.9 are visualized as box plots shown in Figure 3.2. For each evaluated fault localization algorithm, its median of the 11 fault cases' *fl-effectiveness* is plotted in Figure 3.3. The medians are used instead of means to summarize the results because they are less affected by outliers. It can be ob-

Fault localization	**PAM**					**DELTA**					
algorithm	**Fault2**	**Fault7**	**Fault9**	**Fault10**	**Fault15**	**Fault1**	**Fault2**	**Fault3**	**Fault4**	**Fault11**	**Fault16**
MUSE-LOC	2.5	1.5	95	95	9.5	108	1	108	108	4	114
Metallaxis	3.5	3.5	19	11	1	18	1	18	1	1	128.5
Tarantula	56	56	56	56	56	40	1	40	43	3.5	40
Ochiai	53	54	51	52	54	39	1	38	43	2.5	39
Ochiai2	98	99.5	100	100	99.5	12	1	114.5	9	2.5	111.5
Op2	52	52	51	51	52	35	36	35	36	2.5	38
Barinel	56	56	56	56	56	40	1	40	43	3.5	40
DStar	53	54	51	52	54	39	1	37	43	2.5	39
Braun-Banquet	53	54	56	53	54	40	1	40	43	3.5	40
Dennis	95	95	95	95	95	106.5	1	106.5	109.5	3.5	106.5
Mountfod	53	54	51	52	54	39	1	39	43	2.5	40
Fossum	53	54	51	52	54	39	1	38	43	2.5	39
Pearson	100	100	100	100	100	1.5	1.5	114.5	4	6	114.5
Gower	100	100	100	100	100	13	8	114.5	1	4.5	114.5
Michael	95	95	95	95	95	106.5	1	106.5	109.5	2.5	106.5
Pierce	99.5	99.5	100	100	99.5	1	12	114.5	1.5	113	113
Baroni-Urbani & Buser	53	54	61	53	54	40	6	40	43	2.5	42
Tarwid	95	95	95	95	95	106.5	1	106.5	109.5	3.5	106.5
Ample	100	100	100	100	100	2	1.5	114.5	4	6	114.5
Phi (Geometric Mean)	95	95	95	95	95	106.5	1	106.5	109.5	2.5	106.5
Arithmetic Mean	95	95	95	95	95	106.5	1	106.5	109.5	2.5	106.5
Cohen	95	95	95	95	95	106.5	1	106.5	109.5	2.5	106.5
Fleiss	94	91	90	90	91	101.5	102.5	101.5	102.5	2.5	104.5
Zoltar	52	52	51	51	52	35	36	35	36	2.5	38
Harmonic Mean	95	95	95	95	95	106.5	1	106.5	109.5	2.5	106.5
Rogot2	100	100	100	100	100	13	1	114.5	9	2.5	114.5
Simple Matching	53	54	139	53	54	40	67	40	141	2.5	67
Rogers & Tanimoto	53	54	139	53	54	40	67	40	141	2.5	67
Hamming	53	54	139	53	54	40	67	40	141	2.5	67
Hamann	53	54	139	53	54	40	67	40	141	2.5	67
Sokal	53	54	139	53	54	40	67	40	141	2.5	67
Scott	53	56	139	53	56	40	67	40	141	3.5	68
Rogot1	53	56	139	53	56	40	67	40	141	3.5	68
Kulczynski	53	54	55	52	54	40	1	40	43	2.5	40
Anderberg	53	54	55	52	54	40	1	40	43	2.5	40
Dice	53	54	55	52	54	40	1	40	43	2.5	40
Goodman	53	54	55	52	54	40	1	40	43	2.5	40
Jaccard	53	54	55	52	54	40	1	40	43	2.5	40
Sorensen-Dice	53	54	55	52	54	40	1	40	43	2.5	40

Table 3.9: Fault localization evaluation on 11 fault cases

served that Metallaxis outperforms all other evaluated fault localization algorithms considerably. The median of 11 fault cases' fl-effectiveness values for Metallaxis is 3.5 LOC, while the second best algorithm is Op2/Zoltar, with a median of 38 LOC. Additionally, as shown in Figures 3.4 and 3.5, out of 11 fault cases, Metallaxis ranked faulty statements at the 1st position (highest suspiciousness) 4 times and within Top-5 positions 6 times, both counts are the highest over other algorithms. As revealed by empirical studies discussed in Section 3.3.3, the fault localization technique's ability of pinpointing the faulty statement in the top-5 positions is very important [163, 164].

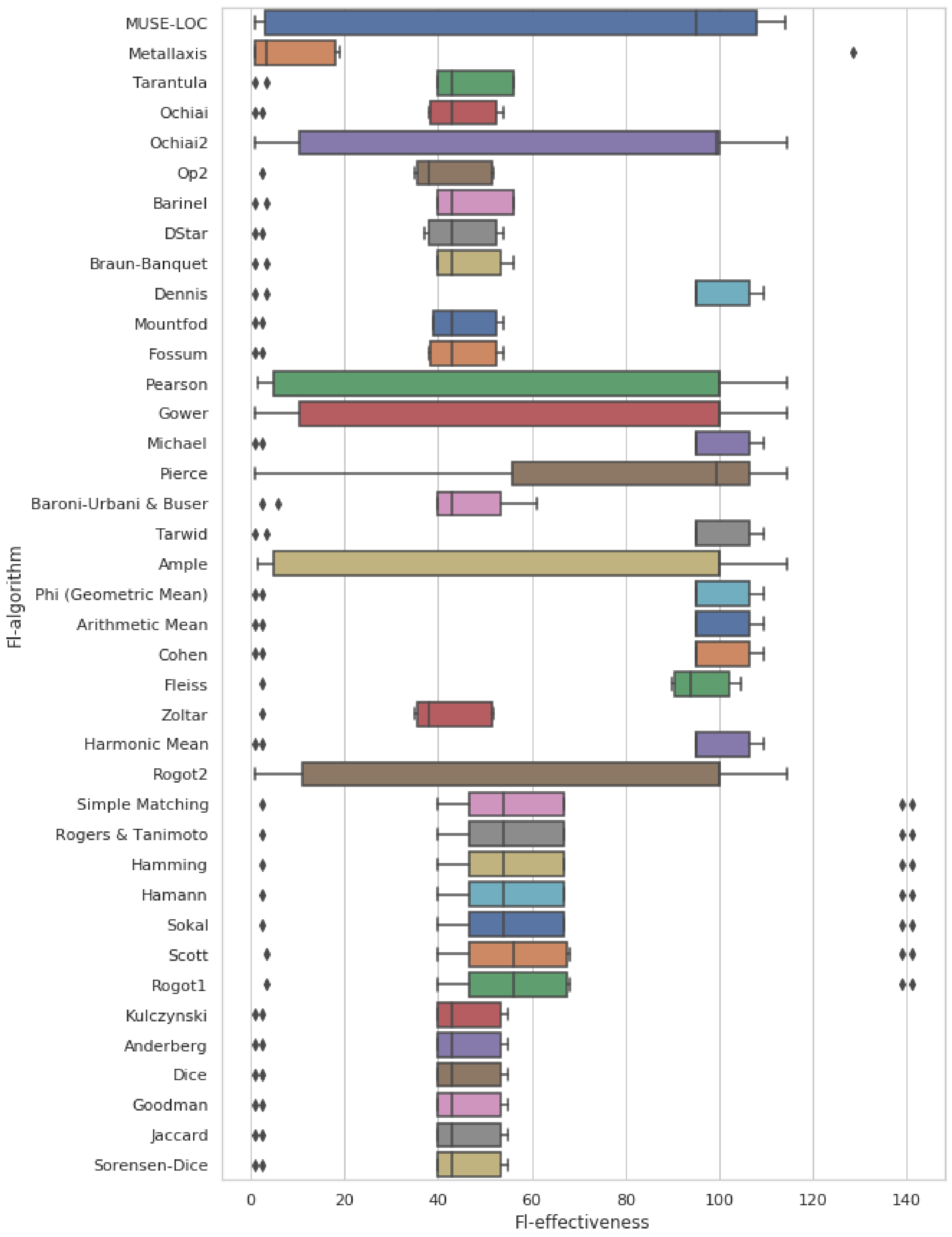

Figure 3.2: Box plot of fault localization evaluation results

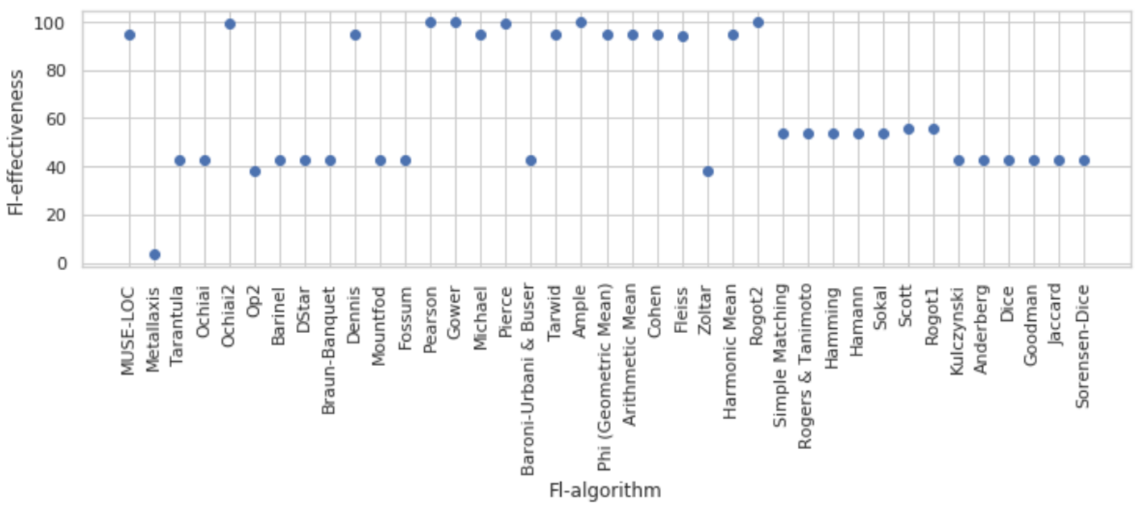

Figure 3.3: Medians of fault localization evaluation results

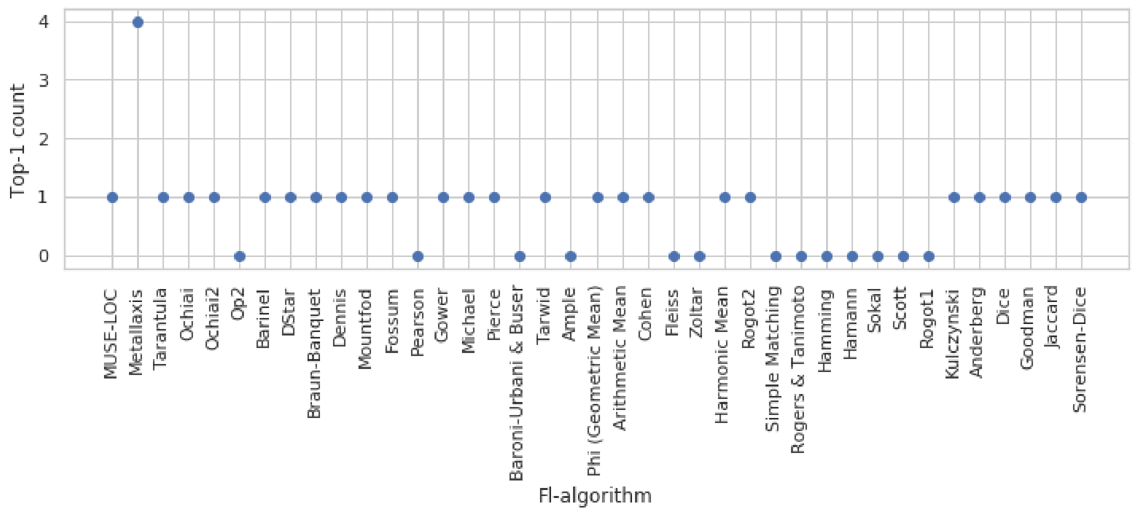

Figure 3.4: Top-1 case counts of fault localization evaluation results

The underlying assumption of spectrum-based fault localization is that statements that are more likely to be executed by failed test cases are more suspicious to be causes of the failures. One limitation when applying spectrum-based fault localization techniques on control logic fault localization is that in many cases, the root cause of the fault in the control logic program is not in any conditional branch (i.e., the faulty statements, together with many other statements, are executed by

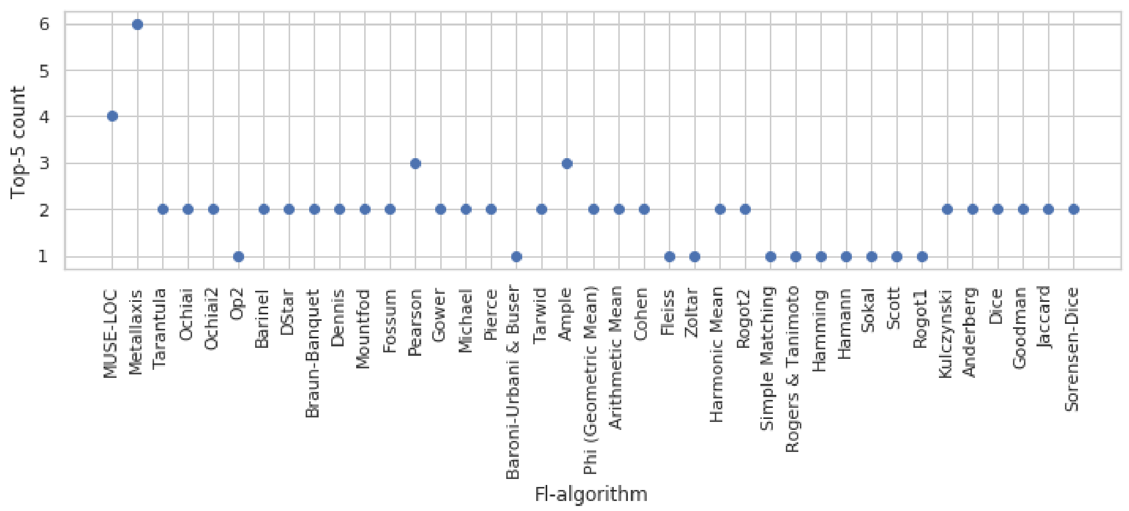

Figure 3.5: Top-5 case counts of fault localization evaluation results

every test case). In this situation, these statements do not show their proneness of being executed by failed or passed test cases, thus spectrum-based fault localization becomes ineffective. All considered spectrum-based algorithms suffered from this issue and there is no clear winner within the evaluated spectrum-based fault localization techniques, as can be observed from Figure 3.2. Mutation-based techniques overcome this limitation because they can scrutinize program statements with additional information by evaluating how mutating the original program will affect outcomes of each test case execution.

While Metallaxis outperforms all spectrum-based fault localization by a large margin, MUSE, the other mutation-based fault localization technique, performs very poorly. More specifically, for fault cases *PAM Fault9*, *Fault10*, and *DELTA Fault1*, *Fault3*, *Fault4*, and *Fault16*, it ranked the faulty statements worse than many spectrum-based fault localization techniques, as shown in Table 3.9. A closer look at the MUSE computations revealed the following two main causes of poor effectiveness: 1) considering program mutants on service requirement related logic, and 2) invalid MUSE computation. These two issues are elaborated in the following two paragraphs.

MUSE relies on the mutants' ability of turning originally failed test cases to passed ones, or the other way around, to infer the suspiciousness of statements. For a specific statement in the control logic program, if no mutant of it has this ability, then the suspiciousness of this statement is 0 [10]. A closer look at *DELTA Fault1*, *Fault3*, and *Fault4* revealed that in these three fault cases, the only mutants that have this ability of switching passed/failed test cases to failed/passed test cases are on the statements that implemented the resetting of supply air temperature service requirements. We use an example to explain why these statements are marked suspicious by MUSE and why they are not the actual faulty statements: if a control logic fault is found causing the cooling coil to be active when the supply air temperature is well below the service requirement set point, a fix that lowers the service requirement set point will make the fault symptom disappear, but this fix is invalid. In other words, if the system's behavior does not meet the service requirement, to fix the problem, one should focus on troubleshooting the incorrect behavior rather than modifying the requirement and compromising the service.

If none of the mutants for a control logic program can switch any passed test cases to fail and/or switching any failed test cases to pass, the MUSE computation (formula discussed in Section 3.3.2) becomes invalid as α is either 0 (no failed to passed instances) or infinite (no passed to failed cases), or undefined (no failed to passed and no passed to failed cases). This scenario happened to *PAM Fault9* and *Fault10*, the two PAM fault cases where MUSE provided very bad results (*fl-*

[10]0 is not the minimum suspiciousness value as there might be negative suspiciousness value in MUSE. Furthermore, suspiciousness is a relative value and it only intends to be used for ranking and does not mean probability

effectiveness was 95). This scenario will happen to *DELTA Fault 1 - 4* as well if no mutants on statements implementing service requirement resetting logic are included in the computation, as discussed in the previous paragraph.

To summarize, both Metallaxis and MUSE leverage the execution profiles from program mutants to infer fault cause locations. Metallaxis collects two types of information: 1) if the mutated program outputs differently from the original program, and 2) whether the corresponding test case is failed or passed on the original program. On the other hand, MUSE collects the information about whether the mutated program can change the test result (pass or fail) of the original program. For AHU control logic programs, while it is easy to have mutants changing program outputs, thus providing insights to Metallaxis, it is much harder to have mutants changing test results in both directions to support MUSE computation. This is largely caused by the feature of control logic fault symptom as multiple expressions of control logic input/output variables inequality expressions being satisfied at the same time [13]. Due to this feature, in many cases, it is rare to have a 1st order mutant[11] switching failed test cases to passed, and it is even rarer to have such a mutant switching passed test cases to failed. This characteristic is unique for control logic fault localization when comparing it with fault localization of typical software engineering domain applications, assuming it is easier to have a mutant "break" a correct program than to have a mutant "fix" a faulty program [129].

It is worth noticing that Pearson et al. [3] evaluated the fault localization performance of 5 spectrum-based techniques, namely Tarantula, Ochiai, Op2, Barinel and D^2, and the same 2 mutation-based techniques as we evaluated. They found that

[11]only have 1 difference from the original program, due to the computational expensiveness of mutation-based fault localization, considering higher than 1st order mutants will introduce too many mutants and make the computation intractable

when used for localizing real-world software faults, Metallaxis and MUSE perform worse than any of the five evaluated spectrum-based techniques. Our evaluation showed contrary results as we found that when diagnosing real-world control logic faults, Metallaxis performs better than spectrum-based fault techniques with a large margin.

3.6.2 Fault localization performance v.s. control logic fault cause semantics

Comparing the semantic characteristics of the cause of each fault shown in Table 3.7 to the Metallaxis fault localization fl-effectiveness of each fault shown in Table 3.9, we have the following observations:

- If a fault is caused by one or more semantic logic elements being incorrect, i.e. fix is done by modifying existing program statements, Metallaxis will perform very well, ranking faulty statements within top-5 positions, as long as there are considered mutants that can mimic at least one incorrect element. For example, one cause of PAM Fault 2, is an incorrect value assigned to the set point variable for the mixing box local controller. Although no mutant fixes this issue exactly, there are mutants changing the assignment of this set point variable to other incorrect values. Using these mutants is enough to make Metallaxis rank the corresponding faulty statement to the third most suspicious statement of all program statements. Other fault cases that fall under this observation are PAM Fault 7, Fault 15, Delta Fault 4 and Fault 11.
- If a fault is caused by one or more semantic logic elements being incorrect, but no mutants can mimic the fix, then it is likely that Metallaxis will be able to provide a ranking of top-20 position, for the fault causing statement. For

example, PAM Fault 9 is solely caused by a supply air fan local controller using an incorrect variable as the feedback. There are considered mutants changing other settings of this local controller, but no mutant changes the incorrect feedback variable name to something else. As a result, Metallaxis provided an fl-effectiveness value of 19 for this fault case. Other fault cases that fall under this observation are PAM Fault 10 and Delta Fault 3.

- When a fault is caused only by missing semantic elements, i.e. fixing the fault requires solely adding new program statements, Metallaxis performs poorly because no mutants will be able to mimic the fault's behavior at all. Fault cases of this scenario are Delta Fault 1 and Fault 16. [12]

The above observations of the relationships between Metallaxis fault localization performance and the fault causes are in line with Metallaxis' underlying idea of computing suspiciousness according to behavioral similarities between faults and mutants.

3.6.3 Fault localization performance v.s. fault fixing statements semantics

We summarize the semantic characteristics of the 68 fault fixing statements and their Metallaxis fault localization rankings into segmented stacked bar graphs in Figure 3.6 and Figure 3.7. The fault fixing statements are characterized in terms of their statement semantic types (the five types in Table 3.8) in Figure 3.6, and are characterized in terms of their fault fixing code change types (modifying/deleting/adding statements) in Figure 3.7.

[12] Although Metallaxis provides an fl-effectiveness of 18 for Delta Fault 1, after a closer look at related mutants provide this ranking, we found that this top-20 ranking comes by coincidence rather than links of mutants to fault cause.

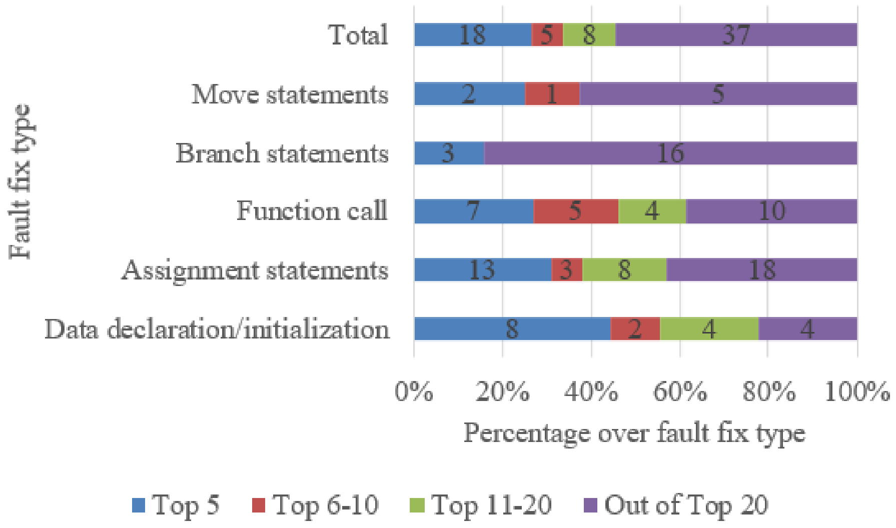

Figure 3.6: Fault fix statements' type v.s. Metallaxis ranking

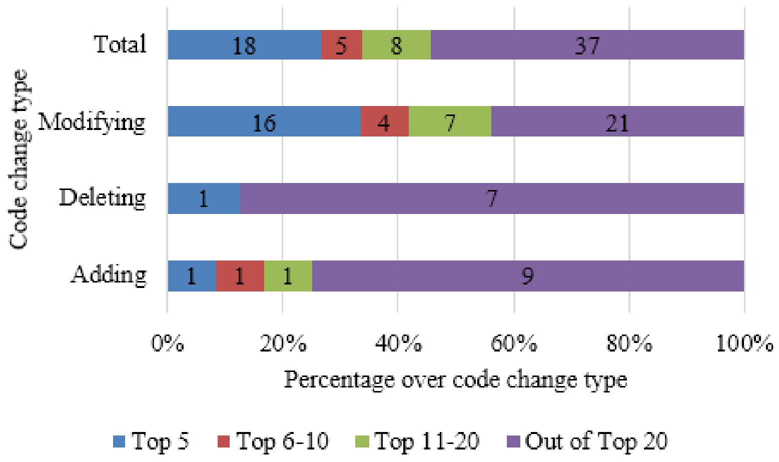

Figure 3.7: Fault fix code change type v.s. Metallaxis ranking

From Figure 3.6 and Figure 3.7, we have the following observations:

- Metallaxis is good at picking up fault fixing statements about data declaration and initialization, and is bad at picking up fault fixes about branch statements.
- Metallaxis is good at picking up fault fixes in terms of modifying existing statements, and is bad at picking up fault fixes about adding new statements or deleting existing statements.

3.7 Mutation operators evaluation and suggestion

As discussed earlier, the predominant factor that affects Metallaxis fault localization performance is whether the considered program mutants are similar to the actual faults. In another word, if the mutation operators utilized to generate program mutants match the fault fixing patterns of control logic faults, then Metallaxis is expected to provide good fault localization results.

In this section, we first summarize the mutants considered in our Metallaxis fault localization evaluation (generated by the Major framework [168]) that provide positive suspiciousness scores for the fault fixing statements about code modification and deletion. Then we provide suggestions of mutation operators based on the categorization of the actual fixes of control logic faults.

3.7.1 Summary of considered mutants

The Major framework [168] is utilized in this work for mutant generation to support mutation-based fault localization computation. We used all Major mutation operators to generate mutants for the two test bed AHU control logic programs. As shown in Table 3.5, 336 mutants are utilized for the PAM test bed, and 693 mutants are utilized for the DELTA test bed.

In order to identify what types of Major mutants help the most for Metallaxis to provide good performance, for each of the 56 faulty statements that corresponding fault fix is performed through code modification or deletion, we select mutants that:

- mutate this statement
- provide positive suspiciousness value
- provide highest suspiciousness value among all mutants of this statement

A total of 82 mutants are selected from the Metallaxis computation of 9 faults. Note that fault fixes through adding new code are not considered because no mutation that modifies/deletes existing code can mimic the behavior of adding new code. As a result, we exclude two faults (DELTA Fault 1 and Fault 16) that are fixed solely through adding new code.

We leverage the mutation operator terminologies used by Major to classify these mutants and summarize their Metallaxis fault localization calculation results in Table 3.10.

Major mutation operator	**Number of involved**			**Metallaxis fl-effectiveness**		**Metallaxis susp. value**	
	Mutants	**Faulty LOC**	**Faults**	**Average**	**Median**	**Average**	**Median**
Literal Value Replacement (LVR)	46	22	9	13.86	6.75	0.61	0.79
Expression Value Replacement (EVR)	12	12	6	17.38	9	0.76	0.99
Relational Operator Replacement (ROR)	10	5	2	24.35	37	0.37	0.1
Conditional Operator Replacement (COR)	5	3	2	30.8	37	0.24	0.1
STatement Deletion (STD)	4	4	4	41.25	49.5	0.35	0.25
Arithmetic Operator Replacement (AOR)	4	1	1	11	11	0.94	0.94
Operator Replacement Unary (ORU)	1	1	1	39	39	0.1	0.1

Table 3.10: Summary of effective mutants for Metallaxis

In Table 3.10, we first ignore the last two rows, namely AOR and ORU, because each of these two appeared only for one faulty statement, thus the fault localization performance is not representative. Moreover, as will be shown in the next subsection, none of the actual fault fixes can be attributed as AOR/ORU mutation operations. Among the rest of the mutation operators, we discuss them in three categories as follows:

- LVR and EVR: these two mutation operators are about replacing literal values with other literal values (LVR), or replacing expression values, such as variable identifiers (names) and method calls, with literal values (EVR). Among the selected mutants, they are more popular and more effective for Metallaxis computation than other mutation operators.

- ROR and COR: these two mutation operators focus on relational and conditional operators in logical expressions, which appear mostly in conditional branch statements. They are not as effective as LVR and EVR mutation operators for Metallaxis. This echos the observation discussed earlier in the previous section: Metallaxis is not very good at picking up control logic faults located in branch statements.

- STD: as the name suggests, this mutation operator is about deleting single statement completely. This mutation operator provides similar performance as the ROR and COR operators.

The above mutation operators will be discussed in more detail in the next subsection when being compared with the actual fixes of control logic faults.

3.7.2 Mutation operator suggestion

In order to discover what mutation operators, if utilized to generate mutants of control logic programs, can provide the most benefits for Metallaxis to localize control logic fault causes, we summarize the actual fixes of the faulty code statements in terms of "mutation operations" conducted. The summary is shown in Table 3.11 (A detailed list is provided in Appendix B). As stated earlier, only fault fixes about modifying/deleting code are considered, which correspond to 56 out of the 68 fault fixing statements. Because 2 of the 56 fault fixes can be attributed to 3 mutation operations each, a total of 60 mutation operations are summarized in the table.

Some notations in Table 3.11 that we borrow from Major are clarified below:

MutOp type	#	**Mutate from**	#	**Mutate to**	#	**Covered by Major**	**Covered by related operator in Major**	**ID for ref.**
LVR	13	POS	9	POS	5	No	Yes	LVR1
				NEG	2	Yes	\	LVR2
				0	2	Yes	\	LVR3
		0	4	POS	4	Yes	\	LVR4
EVR	27	<IDENTIFIER>	7	POS	1	No	Yes (0)	EVR1
				NEG	1	No	Yes (0)	EVR2
				<IDENTIFIER>	4	No	No	EVR3
				<METHOD_INVOCATION>	1	No	No	EVR4
		<IDENTIFIER>as Method Argument	9	<IDENTIFIER>	5	No	No	EVR5
				<METHOD_INVOCATION>	2	No	No	EVR6
				POS(1)	2	No	Yes (0)	EVR7
		<METHOD_INVOCATION>	10	POS(1)	7	No	Yes (0)	EVR8
				NEG	1	No	Yes (0)	EVR9
				0	1	Yes	\	EVR10
				<IDENTIFIER>	1	No	No	EVR11
		Arithmetic Expression	1	<IDENTIFIER>	1	No	No	EVR12
STD	8	Assignment Statement	2	<NO-OP>	3	Yes	\	STD1
		Condition Block	6	<NO-OP>	6	No	No	STD2
COR/ROR	12	Add logical expression(s)			4	No	No	CR1
		Delete multiple logical expressions			2	No	No	CR2
		Delete one logical expression			6	No	No	CR3

Table 3.11: Actual fault fixes summarized as mutations

- POS: positive number
- NEG: negative number
- <IDENTIFIER>: variable name
- <METHOD_INVOCATION>: method call
- <NO-OP>: No code (statements deleted)

From Table 3.11, the first observation is that Expression Value Replacement (EVR) is the most popular mutation operator for actually fixing control logic faults, corresponding to 27 of the 60 fault fix instances, while the second most popular mutation operator, Literal Value Replacement (LVR), corresponds to 13 of the 60 instances. These two operators together cover 2/3 of the actual fault fixes.

While LVR mutations are largely covered by Major, EVR mutations are not, as indicated in Table 3.11. This is especially true for mutation operators that change some variable name / method call / arithmetic expression into a variable name or method call. As a result, although EVR mutations are twice more popular than LVR mutations for fixing faults, among effective mutants generated by Major that support Metallaxis computation (shown in Table 3.10), number of effective LVR mutants are almost four times of the number for effective EVR mutants. Major only supports EVR operations that change into literals, more specifically, default values (e.g. 0 for integer). These non-supported mutations are actually responsible for half of the fault fixes under the EVR category. Based on this observation, we suggest the inclusion of mutation operators that mutate code of aforementioned types into names of variables that have been assigned values before the mutation LOC and names of control logic input variables, filtered by data types. Comparatively, mutation operations that change code into method calls might be difficult to include as the search space is, in most cases, infinite. In the meantime, it is worth noting that replacing code with method calls are rarer than with variable names, i.e. 3 versus 11 among actual fault fixes of our evaluated faults.

The second observation is that in 10% of the cases, fixing control logic faults involves deleting statements about conditional blocks, but this is not supported by Major. Major only supports deleting single statement of assignment, method call, etc., while deleting conditional blocks may require deleting multiple statements (e.g. conditional branch statement together with the bracket pair).

The third observation is that 20% of the fault fixes are about adding/deleting logical expressions in conditional branch statements. While Major supports modifying conditional/relational operators for expressions in conditional branch statements, it does not support adding/deleting logic expression(s) directly. As a result, very limited mutants can be provided to effectively support Metallaxis computation to locate faults about branch statements, as shown in both Figure 3.6 and Table 3.10. Although adding new logical expressions can be a difficult task for a mutation operator, deleting expressions are actually straightforward, and deleting logical expressions are responsible for 8 out of the 12 actual fault fixes about logical expression addition/deletion.

Based on the above discussion, the mutation operators we suggest for supporting Metallaxis computation to diagnose AHU control logic faults are summarized into the following three categories:

1. Changing literal values with other literal values of same data type.

2. Changing variable names (whether they are called directly by assignment statements, or in arithmetic expressions, or as method call arguments) with other variable names that have assigned values, and with literal values of the same data type as the variable to be replaced.

3. Deleting assignment statements, conditional blocks, and logical expression(s) of conditional branches.

Through the link of the 60 actual fault fixes, the mutations listed in Table 3.11 can be grouped by fault cause categories discussed in Section 3.5.4, with which we discuss the linkage between the aforementioned three categories of suggested mutation operators and the control logic fault cause characteristics as follows:

- The fixes of control logic faults caused by incorrect setpoint values and incorrect output mappings typically involve changing the incorrect setpoint values or output mapping boundaries. Changing literal values of related assignment statements or method call arguments are likely to generate mutants that lead to the localization of these faulty statements.

- Control logic faults caused by incorrect local controller feedback typically require fixes replacing the incorrect feedback variable name with the correct one. In this case, mutation operators replacing variable names with other variable names have the chance to provide mutants that are effective in helping localize this type of faults.

- Because supervisory control constraints have different typical types, such as lockout, override, and select, constraints of different types can be implemented in different types of statements. For example, a select constraint can be implemented as an assignment statement with the method call of the select function on the right hand side, while a lockout constraint can be implemented as a conditional branch constraint over the local control function. As a result, control logic faults caused by incorrect constraint expressions of supervisory control may have their faults rooted in different types of statements. All three aforementioned categories of mutation operators should be utilized together for a broad coverage of effective mutations for different statement types. This will maximize the chance to localize faulty code about supervisory control sequencing.

The above fault causes are popular in our 11 fault cases, as indicated in Table 3.7. Moreover, their popularity is also revealed by existing case studies of control logic faults, summarized by Ardehali and Smith [4], in which control logic program faults

Fault cause category	**Actual fault fix mutation operations (represented with ref. ID in Table 3.11)**
Fault cause at local control level	
Incorrect local controller set point	LVR1, EVR1
Incorrect local controller feedback	EVR4, EVR5
Incorrect local controller output mapping	LVR1, CR1, CR3
Fault cause at supervisory control level	
Incorrect constraint expression(s)	LVR2-4, EVR2, EVR3, EVR5-12, STD1, STD2, CR1-3

Table 3.12: Actual fault fix mutation operation grouped by fault causes

such as improper setpoint of different local control functions and using incorrect feedback in local controllers frequently appeared.

3.8 Fault localization setup exploration

Spectrum-based fault localization techniques require the execution profiles of a suite of both failed and passed test cases on the original program under evaluation. Mutation-based fault localization techniques take a step further and require additional execution profiles of this test suite on mutated programs as well. In our evaluation of fault localization techniques in Section 3.6, fault cases of two different AHUs have different test suite size and each fault case has a different ratio of failed test cases in its test suite. It is unknown what is the optimal setting for test suite size and failed test cases ratio.

To investigate the optimal setup options of fault localization computations, we design and conduct experiments to explore the setup option space. A total of 22 different setup scenarios were explored, as listed in Table 3.13. All Major [168] supported mutation operators are utilized to generate mutants for mutation-based fault localization. While Exp0 is the same scenario of the evaluation in Section 3.6 and is used as a baseline reference, the other scenarios are assembled to explore two different dimensions of setup options:

- Exp0 to Exp8 explored test suite sizes at three different orders of magnitude (100, 1000, and original (28800 or 5951)) and four different down sampling approaches to acquire test suites with reduced sizes.
- Exp9 to Exp 21 explored different failed test cases ratios within the test suite, ranging from 1% to 99%.

The following two sub-sections discuss approaches and findings of each dimension's exploration in detail.

ExpID	**# Test cases**	**Fail rate %**	**Down sampling method**	**Mutation Operation**
0	28800 / 5951	original	None	ALL
1	100	original	Random	ALL
2	1000	original	Random	ALL
3	100	original	KMeans	ALL
4	1000	original	KMeans	ALL
5	100	original	AHC	ALL
6	1000	original	AHC	ALL
7	100	original	Sequential	ALL
8	1000	original	Sequential	ALL
9	100	10%	KMeans	ALL
10	100	20%	KMeans	ALL
11	100	30%	KMeans	ALL
12	100	40%	KMeans	ALL
13	100	50%	KMeans	ALL
14	100	60%	KMeans	ALL
15	100	70%	KMeans	ALL
16	100	80%	KMeans	ALL
17	100	90%	KMeans	ALL
18	100	95%	KMeans	ALL
19	100	99%	KMeans	ALL
20	100	5%	KMeans	ALL
21	100	1%	KMeans	ALL

Table 3.13: Fault localization setup experiments information

3.8.1 Sample size and down sampling methods

Our available BAS datasets of the two test beds provide us 28,800 test cases for the PAM test bed and 5,951 test cases for the DELTA test bed. These test cases helped us to detect the existence of the 11 faults detailed in Table 3.6. The computational time of fault localization is linearly proportional to the number of test cases. In this subsection, we explore options of reducing the size of test cases, since using all available test cases for fault localization computation increases the computation time, especially for mutation-based fault localization techniques, which is computationally very expensive. By applying different down sampling methods to reduce the test suite size for fault localization, we explore the possibility of obtaining fault localization performance at the same level as using all test cases. We aim at reducing the test suite size by orders of magnitude, thus, besides evaluating the fault localization performance using all available test cases, i.e. thousands or tens of thousands of test cases, we also evaluate test suite sizes of 1000 and 100.

When selecting a subset of test cases, we conjecture that selecting test cases that are distinct will benefit the fault localization performance, because similar test cases provide similar, if not the same, test results and execution profiles, which leads to redundant information being given to fault localization computations. This conjecture is in line with the heuristic of similarity-based test case selection in software testing studies, which state that "the greater the dissimilarity between test cases, the easier it becomes to detect faults" [181]. Among test case selection methods that are proposed in the software testing domain, two popular methods utilized to select test cases before test execution are, random selection and clustering methods [181]. An effective choice of clustering algorithm used in the test case selection studies [182, 183] is Agglometrative Hierarchical Clustering (AHC) [184]. Complementing random selection and AHC with K-Means [185] (another popular clustering algorithm) and a baseline reference, we adopted four methods of down sampling in our experiments to select k (including f failed test cases and p passed test cases, $f + p = k$) test cases from the original 28,800 or 5,951 test cases:

- Random: Select f test cases randomly from all available failed test cases, and select p test cases randomly from all available passed test cases.
- AHC: Use the AHC algorithm to cluster all available failed test cases into f clusters, then select 1 test case from each cluster to assemble a total of f failed test cases. Use the same approach to select p passed test cases from all available passed test cases.
- K-Means: Use the K-Means algorithm instead of AHC algorithm and follow the same process under the AHC method.
- Sequential: Select the first f test cases from all available failed test cases, and select p test cases randomly from all available passed test cases.

We conducted the experiments for all 39 considered fault localization algorithms. From the results of the experiments, we have the following observations:

- The performance of considered four down sampling methods can be ranked as K-Means $\approx$ AHC $>$ Random $>$ Sequential. This echoed our conjecture discussed above.
- For the three different levels of sample sizes we explored (original, 100, and 1000), while a bigger sample size provides better fault localization results in general, the effect is only marginal if a clustering algorithm (K-Means or AHC) is used for down sampling. For example, the 6 fault cases that had their faulty

statements ranked within top-5 positions in Exp0 (as discussed in Section 3.6 and shown in Table 3.9) are still having their faulty statements ranked within top-5 positions in Exp3. The Metallaxis *fl-effectiveness* median value of 3.5 in Exp0 becomes 4.0 in Exp3.

3.8.2 Failed test cases ratio

The 11 fault cases originally have different failed test cases ratio in their test suite, which can be computed based on the information in Table 3.6. We conducted Exp9 to Exp21 to investigate if there exist an optimal failed test cases ratio to achieve best fault localization performance. As discussed in the previous subsection, a test suite size of 100 using one of the clustering methods achieves almost the same fault localization performance as using all test cases. For computational efficiency, all experiments discussed in this subsection are conducted with a test suite size of 100, down sampled from the original test suite using the K-Means clustering method.

From these experiments, we observe three typical patterns between *fl-effectiveness* and failed test cases ratio. The three typical patterns can be represented by the results of Metallaxis, MUSE, and Op2 respectively, as shown in Figure 3.8, in which number of fault cases that got *fl-effectiveness* less than or equal to 5 is plotted:

- Pattern A: fault localization performs better when the failed test cases ratio increases at lower values and remains the same after some mid-range threshold. The Metallaxis plot in Figure 3.8 represents this pattern.
- Pattern B: fault localization performs better when the failed test cases ratio increases at lower values, then the performance remains the same after some mid-range threshold, finally the performance drops after failed test cases ratio reaches 90% or 95%. The MUSE plot in Figure 3.8 represents this pattern.

- Pattern C: fault localization performance does not change much under different failed test cases ratios. The Op2 plot in Figure 3.8 represents this pattern.

Overall, with a test suite size of 100, our findings suggest that the optimal failed test cases ratio range is from 60% to 90%.

It is worth noticing that Abreu et al. [186] evaluated test suite options for spectrum-based fault localization. Knowing some key differences in the study of Abreu et al. that they only used Ochiai for evaluation and their fault cases most involve a single faulty location in the program, our findings are different from theirs. Abreu et al. found that adding failed test cases improve the fault localization performance, but the benefit of having more than around 10 failed test cases is marginal on average. However, we found that in many cases, there is much room to improve the fault localization performance by adding more failed test cases.

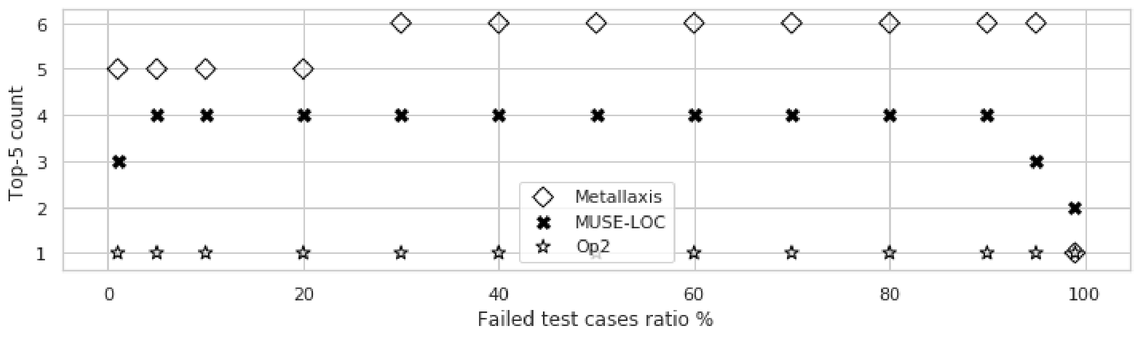

Figure 3.8: Top-5 case counts of different failed test cases ratio

3.9 Conclusions and discussions

As the first attempt to leverage software fault localization techniques in the HVAC domain, in this study, we proposed a framework of adopting spectrum-based and mutation-based fault localization techniques for AHU control logic program fault diagnosis.

We evaluated the performance of 39 existing mutation-based and spectrum-based fault localization techniques on 11 real-world AHU control logic fault cases from 2 AHU test beds and identified that, for AHU control logic fault localization, the mutation-based Metallaxis method significantly outperformed all other methods, while the computations of the other mutation-based method, MUSE, were invalid in many cases. Due to the lack of existing control logic fault cases, we have to develop our own test beds of 2 AHUs, from which a total of 11 fault cases are leveraged. With only 11 fault cases, we do not compute and claim the evaluation results with statistical significance. However, the large effects demonstrated in Figure 3.2 and the diversity of the fault causes and fault fixing statements illustrated in Table 3.7 and Table 3.8 lead us to believe that Metallaxis' superior performance on localizing control logic faults when compared with other algorithms is unlikely to be coincidental. In the meantime, we recognize the limited size of the evaluation case set and suggest future work of evaluating fault localization techniques on additional fault cases to obtain results with statistical significance.

Based on our characterization of the control logic faults and the observations of fault characteristics' relationship to Metallaxis fault localization performance, we would like to provide the following practical suggestions:

- If the control logic program under evaluation is relatively complete, i.e., it contains all control logic semantic elements and the fixes to existing faults are likely to be only about modifying existing control logic code, then one should use Metallaxis as it is likely to provide very good fault localization results
- If the control logic program under evaluation is relatively incomplete, i.e. it misses some control logic semantic elements (e.g. misses a local control loop for a specific component) and the fault fixes likely involve adding new code for

the missing logic, then one should rely more on manual troubleshooting and only uses Metallaxis loosely

The differences between our evaluation of fault localization algorithms to those done by others [3, 129] imply that there are some underlying differences between HVAC control logic faults and the software faults considered by them [3, 129]. Our work discussed in this chapter cannot answer what these differences are, and this calls for future research on code analysis [187] of HVAC control logic programs.

We took a close look at what mutants we considered are most effective in supporting Metallaxis computation for control logic fault localization. We also make suggestions of three categories of mutation operators based on the categorization of control logic fault fixes in terms of mutation operations and discussed why these suggested mutation operators would lead Metallaxis to AHU control logic fault causes.

In this chapter, we also conducted sensitivity analysis to explore the setup options of fault localization computations in two aspects: 1) test suite size and down sampling approaches, and 2) test suite failed test case ratio. The exploration is conducted with 22 controlled experiments and the results suggest: 1) using clustering-based down sampling methods to acquire a 100 test cases suite provides as good fault localization performance as using tens of thousands of test cases; 2) a safe choice of failed test cases ratio within test suite is ranging from 60% to 90%.

Postamble

In this chapter, I present the research work addressing the second research question discussed in Chapter 1, structured as a self-contained journal paper draft.

This research mainly targets the identification of a software fault localization technique among various existing software fault localization algorithms for AHU control logic fault diagnosis, through evaluation of considered software fault localization algorithms against real-world AHU control logic fault cases. I was able to develop 11 fault cases (with their fault symptoms and causes verified by BAS datasets and control logic program simulation) to be used in the evaluation. My objective is not to make claims with statistical significance based on fault localization evaluation on these 11 fault cases. Instead, I developed these fault cases with the aim of them being diverse in terms of fault types (symptoms), fault causes, and fault fixing code types. According to the evaluation results, I found that among the considered fault localization algorithms that suit for the task scenario of AHU control logic fault cause diagnosis, the mutation-based Metallaxis method [130] outperformed all other evaluated algorithms with a large margin.

The contributions of this research is summarized into the following points:

- A framework of casting AHU control logic fault diagnosis problem into a software fault localization task
- Performance evaluation of spectrum-based and mutation-based fault localization algorithms for the purpose of locating AHU control logic fault causes with analysis of characteristics of control logic faults and their relationships to fault localization performance
- Evaluate the mutation operators' impact on fault localization and make mutation operators suggestions based on analysis of actual fault fixes patterns
- Exploration of setup options for AHU control logic fault localization computation

Based on the evaluation results shown in this research, the potential industrial deployment of software fault localization algorithm, more specifically, Metallaxis, with the setup option guideline provided in this research, will aid the manual control logic fault diagnosis to improve the efficiency of troubleshooting control logic fault causes.

Chapter 4

An Integrated Use Case of Control Logic Fault Identification and Diagnosis

4.1 Introduction

To concretely show the application scenario and value of the proposed research in a practical setting, in this chapter, I detail the usage of the proposed research with a complete use case demo.

Two software applications, each implementing one research approach discussed in Chapter 2 and 3, are developed and used in this demo. Details of these applications are provided in Section 4.2. Leveraging the developed software applications, this demo shows the processes of troubleshooting the control logic program of a real-world AHU (the "PAM" AHU test bed discussed in Chapter 3) through the following sequential activities:

1. Information collection of the AHU under evaluation

2. Automatic control logic fault definition using the control logic fault definition software implementation

3. Automatic control logic fault detection of BAS historical data with regard to the defined control logic faults

4. Computer-aided fault diagnosis with the fault localization software implementation

These four activities are detailed in dedicated sections after the details of system implementation.

4.2 System Implementation

4.2.1 AHU control logic fault definition system

As discussed in Chapter 2, the fault definition system consists of a inference engine and a HVAC ontology.

The developed HVAC ontology, shown in Chapter 2 (Figure 2.2), is implemented as an XML Schema Definition (XSD) and is used to specify the requirements and encoding of the AHU information. Information specified by the ontology is the only information needed from the user (i.e. the only information needed about the specific AHU system) as the input to the control logic fault definition inference engine.

The inference engine has two layers of reasoning: the lower level reasoning mechanism encodes the reasoning mechanism responsible of deriving control logic fault definition according to high level control objectives; the higher level reasoning mechanism merges the control logic fault definition results provided by the lower level reasoning mechanism and resolves control logic fault conflicts if there exists any. The inference engine is implemented as a Java application and the implementation of the two layers of reasoning mechanisms are completely decoupled as shown in Figure 2.3 so that extending the system with additional lower level reasoning mechanisms can be done without modifying higher level reasoning implementation. In my implementation, the inference engine outputs the control logic fault definition in two forms: textual control logic fault definition list and executable Python script that can be directly imported for control logic fault definition (additional information about the BAS data set is needed, as will be showcased later in this chapter).

4.2.2 AHU control logic fault localization system

The control logic fault localization framework, shown in Chapter 3 (Figure 3.1), is implemented in the Java programming language. The implementation contains the following modules that shall be run sequentially:

1. Test Code Generation: generate control logic program JUnit test code according to BAS data and control logic fault definition.

2. Mutant Generation: generate mutants using the Major framework [168]. This is only needed for running mutation-based fault localization algorithms.

3. Test Execution: execute the JUnit tests with Tacoco [169] execution profile analyzer.

4. Test Results Extraction: extract and summarize the test case results and execution profiles.

5. Test Case Selection: select a subset of test cases for fault localization consideration. This is only needed if not all test cases are considered in fault localization.

6. Fault Localization Calculation: calculate the fault localization results with spectrum-based and / or mutation-based fault localization algorithms.

As shown in Figure 3.1, this implementation asks for the following inputs about the AHU under evaluation:

- Control logic program source code encoded as a Java class, provided to the "Mutant Generation" module.
- Control logic test suite from BAS data set containing all control logic input data points as a csv file, provided to the "Test Code Generation" module.
- Control logic fault to be diagnosed as a Java mathematical expression with Boolean result, provided to the "Test Code Generation" module as well.

The implemented fault localization framework outputs a CSV file containing the calculated statement-wise suspiciousness values of the control logic program with regard to the provided control logic fault expression, for each of the considered fault localization algorithms. This result is further transformed into suspiciousness ranking, analyzed and visualized with Python scripts.

4.3 Use case information collection

4.3.1 Specific AHU configuration information

For control logic fault definition, the information needed to specify the configuration of the PAM AHU is encoded in an XML file listed below.

Listing 4.1: PAM AHU XML file

```
    <?xml version="1.0" encoding="UTF-8"?>

<Project name="PAM DOH AHU-1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
```

```
<AHUComponent name="Linked OAD-MAD-EAD" id="1"
    compType="MixingBox_ModLinkOAMAEA">
    <Function name="MBHeating" id="2" funcType="SAHeating">
        <EnergyEfficiency name="MBEfficiency" id="3" efficiency="99" />
        <FunctioningCondition name="MBHeatingCondition" id="4"
            opAId="41" expOperator="GT" opBId="43" />
        <Command name="MAO" id="5" varName="DOH_AHU-1_Economizer Damper
            Output">
            <SaturationPoint name="MAOMax" id="6" satType="Max"
                satValue="100" />
            <SaturationPoint name="MAOMin" id="7" satType="Min"
                satValue="20" />
        </Command>
    </Function>
    <Function name="MBCooling" id="8" funcType="SACooling">
        <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
        <FunctioningCondition name="MBCoolingCondition" id="10"
            opAId="41" expOperator="LT" opBId="43" />
        <Command name="MAO" id="11" varName="DOH_AHU-1_Economizer
            Damper Output">
            <SaturationPoint name="MAOMax" id="12" satType="Max"
                satValue="100" />
            <SaturationPoint name="MAOMin" id="13" satType="Min"
                satValue="20" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="PHTCoil" id="20" compType="PreheatCoil">
    <Function name="PCHeating" id="21" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="22" efficiency="1.0"
            />
        <Command name="PHO" id="23" varName="DOH_AHU-1_Preheat Valve
            Output">
            <SaturationPoint name="PHOMax" id="24" satType="Max"
                satValue="100" />
            <SaturationPoint name="PHOMin" id="25" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="CHWCoil" id="26" compType="CoolingCoil">
    <Function name="CCCooling" id="27" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="28" efficiency="1.0"
            />
        <Command name="CCO" id="29" varName="DOH_AHU-1_CHW Valve
            Output">
            <SaturationPoint name="CCOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="CCOMin" id="31" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="32" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="33"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="SFO" id="35" varName="DOH_AHU-1_SA Fan VFD
            Speed">
            <SaturationPoint name="SFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="37" satType="Min"
                satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDRF" id="38" compType="ReturnFan_VFD">
        <Function name="RAFlowIncrease" id="133" funcType="RAFlowIncrease">
            <EnergyEfficiency name="RFEfficiency" id="134" efficiency="1.0"
                />
            <Command name="RFO" id="135" varName="DOH_AHU-1_RA Fan VFD
                Speed">
                <SaturationPoint name="RFOMax" id="136" satType="Max"
                    satValue="100" />
                <SaturationPoint name="RFOMin" id="137" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="41" varName="DOH_AHU-2_OA Temperature"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="42" varName="DOH_AHU-1_MAT"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="43" varName="DOH_AHU-1_RAT"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="44" varName="DOH_AHU-1_SAT"
        sensorType="SupplyAirTemperature" />
    <Sensor name="OAF" id="45" varName="DOH_AHU-1_OA Airflow"
        sensorType="OutdoorAirFlow" />
    <Sensor name="SSP" id="46" varName="DOH_AHU-1_Remote Static Pressure"
        sensorType="SupplyAirPressure" />
    <Sensor name="SAF" id="47" varName="DOH_AHU-1_SA Airflow"
        sensorType="SupplyAirFlow" />
    <Sensor name="RAF" id="48" varName="DOH_AHU-1_RA Airflow"
        sensorType="ReturnAirFlow" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="49" spValue="1.25" spType="Tracking" />
    <SetPoint name="SAHeatingSP" id="50" spValue="45" spType="LowerBound"
        />
    <SetPoint name="SACoolingSP" id="51" spValue="50" spType="UpperBound"
        />
    <SetPoint name="RAFlowSP" id="58" spValue="DOH_AHU-1_SA Airflow SP"
        spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOHeating" id="52" setPointId="50"
        feedBackId="44" outputId="5" pidType="Reverse" />
    <ClosedLoopControl name="MAOCooling" id="53" setPointId="51"
        feedBackId="44" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="55" setPointId="50"
        feedBackId="44" outputId="23" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="56" setPointId="51"
        feedBackId="44" outputId="29" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="57" setPointId="49"
        feedBackId="46" outputId="35" pidType="Reverse" />
    <ClosedLoopControl name="RFOFlowIncrease" id="59" setPointId="58"
        feedBackId="48" outputId="135" pidType="Reverse" />
</Project>
```

The AHU information contained in the above XML is the end product after all user inputs. That is, the XML file shown above is the XML provided directly to the inference engine for control logic fault definition.

4.3.2 BAS data set

A BAS historical dataset is used to detect defined control logic faults, and later to provide test cases for control logic fault diagnosis. For the PAM AHU, the system is in normal operation mode 24 hours a day and we acquired 20 consecutive days of data in November/December for our use case. The collected raw BAS dataset is not interpolated and each data point has its own time stamps. This data set is then interpolated into one-minute interval data, with a total of 28,800 samples. The data set contains all the input and output variables of the control logic program.

4.3.3 Control logic program

The control manufacture of the PAM AHU is American Auto-matrix [188]. The original control logic program of this AHU is implemented as different objects referencing each other in American Auto-matrix's NBPro software. We have access to this software and the control logic program is manually translated into Java code with the help of the NBPro software documentation. The building manager and the original control programmer of this AHU also offered help during the translation.

The translated Java program is verified by simulating it with control logic input provided by the BAS dataset and comparing simulation output with actual control logic outputs in the BAS dataset. Simulated data and actual BAS interpolated data agrees well for most of the points, except for the cooling coil command. The BAS data of the cooling coil command has erroneous readings. Due to this issue, all cooling coil command related control logic problems are excluded in our study for the PAM AHU.

4.4 Control logic fault definition

Running the control logic fault definition application with the input XML file shown in Section 4.3, a total of 16 control logic faults are defined for the PAM AHU, as shown below. (Note that it is a coincidence that the number of the control logic faults defined by the control logic fault definition system here and the number of those defined manually in the motivating case study are both 16)

Listing 4.2: PAM AHU control logic fault definition

```
1,[DOH_AHU-1_SAT < 45 (continuously), DOH_AHU-2_OA Temperature >
   DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper Output < 100]
2,[DOH_AHU-1_SAT > 50 (continuously), DOH_AHU-2_OA Temperature <
   DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper Output < 100]
3,[DOH_AHU-1_SAT < 45 (continuously), DOH_AHU-1_Preheat Valve Output < 100]
4,[DOH_AHU-1_SAT > 50 (continuously), DOH_AHU-1_CHW Valve Output < 100]
5,[DOH_AHU-1_Remote Static Pressure < 1.25 (continuously), DOH_AHU-1_SA
   Fan VFD Speed < 100]
6,[DOH_AHU-1_RA Airflow < DOH_AHU-1_SA Airflow SP (continuously),
   DOH_AHU-1_RA Fan VFD Speed < 100]
7,[DOH_AHU-1_SAT > 45 (continuously), DOH_AHU-1_Preheat Valve Output > 0]
8,[DOH_AHU-1_SAT < 50 (continuously), DOH_AHU-1_CHW Valve Output > 0]
9,[DOH_AHU-1_Remote Static Pressure > 1.25 (continuously), DOH_AHU-1_SA
   Fan VFD Speed > 0]
10,[DOH_AHU-1_RA Airflow > DOH_AHU-1_SA Airflow SP (continuously),
   DOH_AHU-1_RA Fan VFD Speed > 0]
11,[DOH_AHU-2_OA Temperature > DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper
   Output < 100, DOH_AHU-1_Preheat Valve Output > 0]
12,[DOH_AHU-2_OA Temperature < DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper
```

```
    Output < 100, DOH_AHU-1_CHW Valve Output > 0]
13,[DOH_AHU-2_OA Temperature > DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper
    Output > 20, DOH_AHU-1_CHW Valve Output > 0]
14,[DOH_AHU-2_OA Temperature > DOH_AHU-1_RAT, DOH_AHU-1_Preheat Valve
    Output > 0, DOH_AHU-1_CHW Valve Output > 0]
15,[DOH_AHU-2_OA Temperature < DOH_AHU-1_RAT, DOH_AHU-1_Economizer Damper
    Output > 20, DOH_AHU-1_Preheat Valve Output > 0]
16,[DOH_AHU-2_OA Temperature < DOH_AHU-1_RAT, DOH_AHU-1_Preheat Valve
    Output > 0, DOH_AHU-1_CHW Valve Output > 0]
```

4.5 Control logic fault detection

When detecting the existence of the 16 defined control logic faults, the following thresholds are used.

- Temperature threshold: 2 °F
- Pressure threshold: 0.2 inch of water
- Air flow rate threshold: 200 CFM

For example, the first defined fault can be implemented as the following Python code for fault definition:

Listing 4.3: PAM AHU control logic fault definition

```
temp_thresh = 2
if (pData['DOH_AHU-1_SAT'] < 45 - temp_thresh) and \
(pData['DOH_AHU-2_OA Temperature'] > pData['DOH_AHU-1_RAT']) and \
(pData['DOH_AHU-1_Economizer Damper Output'] < 100): faultCount += 1
```

Using thresholds in this manner to add margin to the expression that specifies unsatisfied service requirements corresponds to the PID controller behavior (P controller if being simplified in the control logic program simulation) in control logic programs. The selection of the thresholds value shall be decided heuristically while considering the values used as proportional gains in local component controllers. For example, if the economizer PID/P controller has a proportional gain of 50, then when the deviation between the supply air temperature and the set point is larger than 2, then economizer control output should be saturated to 100 (computation of PID controller is described in Chapter 2).

Note that troubleshooting the PID gain settings are out of the scope of the proposed control logic program fault identification and localization framework. As long as the P gains used in control logic program simulation corresponds to the thresholds used in the control logic fault definition as described above, the framework will be able to troubleshoot the control logic faults of focus.

Using these thresholds, the fault detection results from the 28,800 samples are shown below, and visualized in Figure 4.1.

Listing 4.4: PAM AHU control logic fault definition

```
Fault_1        0
Fault_2    27989
Fault_3        0
Fault_4    16702
Fault_5       46
Fault_6        1
Fault_7    17834
Fault_8        4
```

```
Fault_9      163

Fault_10   25505

Fault_11       0

Fault_12   11307

Fault_13       0

Fault_14       0

Fault_15   17589

Fault_16    1159
```

4.6 Control logic fault localization

In this use case, we show the diagnosis of "Fault 2" for the PAM AHU. This fault is observed in 27,989 out of the 28,800 samples.

As discussed in Section 4.2, to conduct fault localization, the following steps are taken based on our implementation.

1. The 28,800 samples from the BAS dataset and the focusing control logic fault definition are provided to the test case generation module to generate explicit non-parameterized JUnit test classes. Parameterized JUnit tests are not supported in the current implementation due to limitation of the Tacoco execution profile analyzed utilized.

2. All mutation operators of the Major framework is leveraged to generate mutants of the control logic program. A total of 336 mutants are generated for the main logic section of the control logic program (mutants generated for statement in helper functions and for control logic input assignments are discarded).

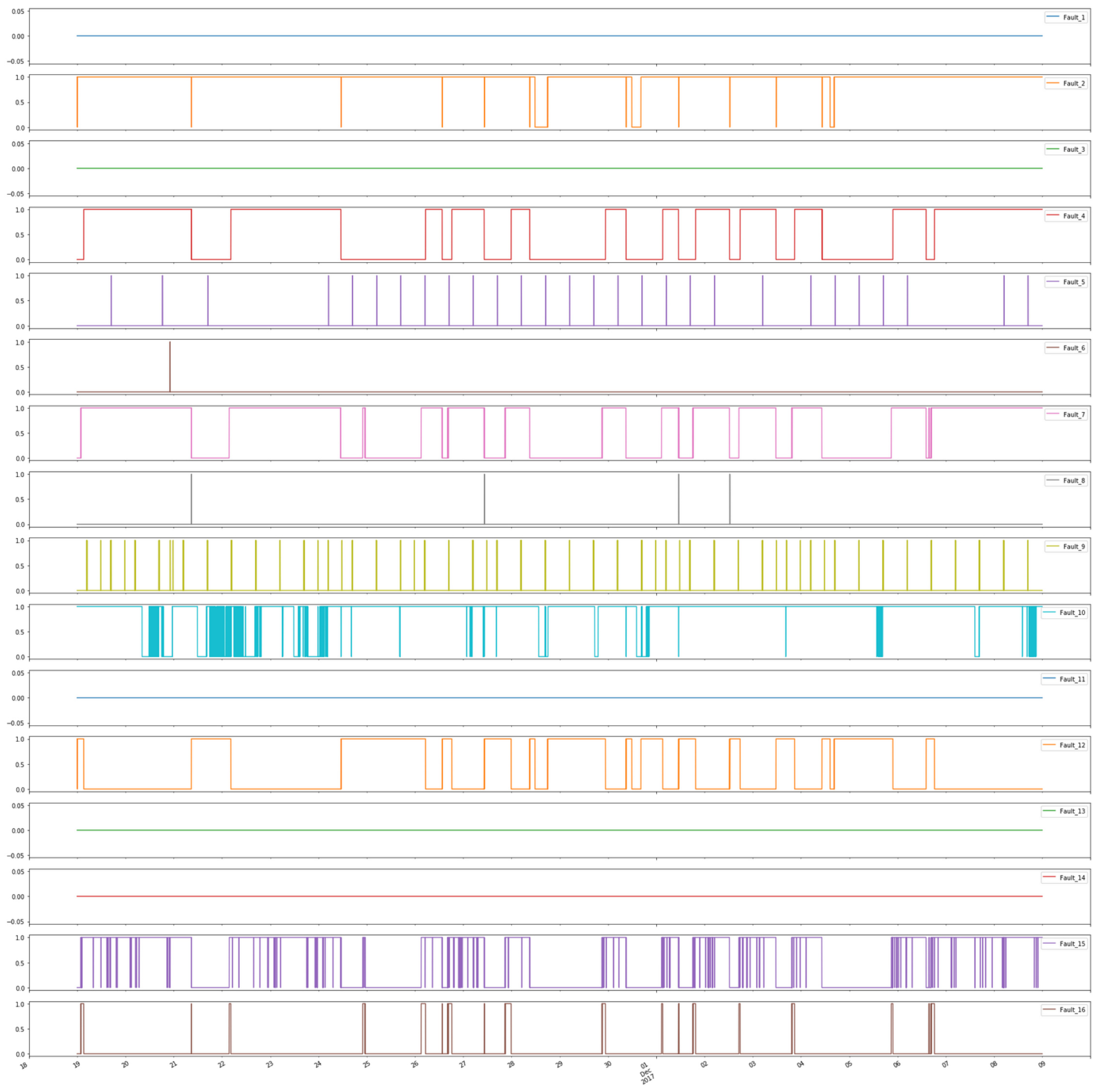

Figure 4.1: 1-minute-trigger fault flags for PAM AHU 1 with thresholds

3. The generated mutants, together with the original control logic program is executed with the generated JUnit tests by Tacoco to provide execution profiles. While the previous steps took little time to conduct (within seconds), this step is computationally expensive. Running 337 versions of the program with regard to 28,800 tests means the control logic program needs to be executed almost 10 million times. The tasks of mutant test executions can be con-

ducted in a paralleled mannered because test executions for different mutant are independent from each other. In my experiment, a Linux workstation with Intel Xeon E5-2623 v4 CPU (8 logical cores @ 2.60 GHz) and 32 GB RAM is utilized and this step can be finished in around 8 hours with paralleled tasks.

4. The Tacoco execution profiles and test execution results (pass/fail, kill/alive) of original control logic program and mutants are summarized and extracted by the test results extraction module. This step is done within 30 minutes.

5. The extracted test results are provided to the fault localization calculation module, and the fault localization calculation is done in around 15 minutes.

Based on the fault localization algorithm evaluation results in Chapter 3, the mutation-based Metallaxis algorithm outperforms other evaluated techniques considerably. Thus, I use the Metallaxis fault localization suspiciousness results to direct my manual search of the control logic fault cause. The Metallaxis calculation results is visualized as a scatter plot in Figure 4.2.

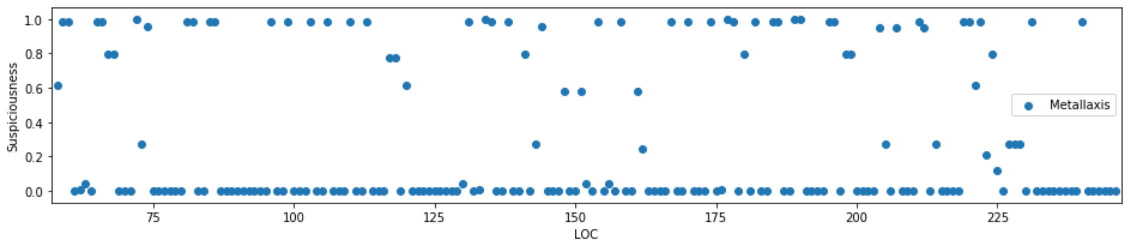

Figure 4.2: PAM Fault 2 Metallaxis calculated statement suspiciousness

Following the list of control logic program statements ranked according the descending order of the Metallaxis suspiciousness value, the control logic program is manually examined. As an illustration, the top-10 ranked statements are shown in Table 4.1.

Line No.	Suspiciousness	Java control logic program statements
189	0.999535855	if (EnthalpySelect <MATLowLimPID) {
190	0.999446489	MnMxAvg1 = EnthalpySelect;
72	0.998641833	double MATStpt = 55;
134	0.998641833	double MATStptRemap = MATStpt;
177	0.998552387	double MAT_Control = PID_Normal(Mixed_Air_Temp, MATStptRemap, 30, FanCommand, 2);
154	0.985853829	double ElecRm_Space_Temp_Setpoint_Remap = ElecRm_Space_Temp;
59	0.985819597	double Static_Pressure_Setpoint = 1.25;
60	0.985819597	double Speed_Offset = 5;
81	0.985819597	int Control_Slect2 = 0;

Note: statements with same Metallaxis suspiciousness value are listed according to their line number sequence

Table 4.1: Metallaxis top-10 ranked suspicious statements of PAM AHU Fault 2

Sequentially checking the ranked statement list shown in Table 4.1, a fault causing statement is soon examined: MATStpt should be 50 °F instead of 55 °F. Context of the fault cause and fixes are provided below.

Key logic details of the control logic program related to this fault are:

1. The normal economizer control is implemented as a direct acting PID controller (LOC 177) with a set point of 55 °F (LOC 72).

2. The economizer control has a temperature bound switch, i.e., the economizer control is on when -10 °F < OAT < 100 °F.

3. The economizer control has a enthalpy switch, i.e., the economizer control is on when outdoor air enthalpy > 20.

This logic has the following issues causing the focusing fault:

1. The cooling function requirement for this unit is a set point of 50 °F (as specified in the xml instance), not 55 °F (LOC 72).

2. The enthalpy switch of the economizer should be comparing the outdoor air enthalpy with the return air enthalpy, rather than comparing outdoor air enthalpy with a static enthalpy value of 20 BTU/lb.

3. The economizer closed-loop control (LOC 177) uses mixed air temperature as feedback rather than discharge air temperature. (This is a problem, although it is not the cause of any failed test cases in this test suite, verified by simulation)

With the help of Metallaxis ranked statements, we are able to locate one of the three logic problems causing the focusing fault at the early statement of manual examination and using it as the start point to troubleshoot the program.

4.7 Summary of the integrated use case

In previous sections of this chapter, we demonstrate the usage and value of the research approaches proposed in Chapter 2 and Chapter 3 with a concrete and real-world use case.

The prototype implementation of the proposed control logic fault identification and localization framework was able to first formally derive the definition of 16 control logic faults for the PAM AHU under evaluation. The control logic fault definition can be directly utilized for fault detection. Then control logic localization framework casted the fault diagnosis problem into a software fault localization task and leveraged the Metallaxis algorithm to provide guidance for manual control logic examination. With the help of the fault localization results, we can discover the program statements causing the fault at very early stage of the manual examination process.

4.8 Envisioned industry use cases

We envision four typical use cases of our proposed approach once deployed in the industry as an software application provided by the control manufacture. Each envisioned use case is discussed with one subsection below.

4.8.1 Use case 1: AHU control logic fault identification and localization during functional test in commissioning

Brief description

This use case describes how the implementation of the proposed approach (hereafter referred to as application) helps control logic fault identification, detection and diagnosis during HVAC control functional test during commissioning / recommissioning / retro-commissioning.

Actor

The primary actor is the user of this application, who is in charge of control logic verification of the AHU systems during commissioning.

Preconditions

1. AHU system design intents are collected in terms of sequence of operations and control drawings (showing system component availability and data points).
2. BAS data is collected from functional tests or normal operation.
3. Access to control logic program is available.

Basic flow of events

1. The use case begins when the user starts to verify the control logic program of the AHU system during commissioning.
2. The user instantiates the AHU component and control ontology based on the AHU control design intents.
3. The user provides the instantiated ontology instance to the application.

4. The application returns derived control logic fault definition back to the user in terms of textual document and executable code.
5. The application checks the BAS data for existence of control logic faults.
6. The application shows the visual and summary of control logic fault detection results.
7. The user selects an existing control logic fault for fault diagnosis.
8. The application executes the fault localization task with the selected fault.
9. The application returns ranked list of control logic program statements based on suspiciousness values calculated by the fault localization algorithm.
10. The user finds and fixes the control logic fault causes with the assistance of the ranked list from the previous step.
11. The application executes the modified control logic program with control logic inputs from the BAS data and verifies the symptom of the fault disappears.
12. Return to 7 if the user wants to diagnose another existing fault.

Post-conditions

Existing control logic faults in the AHU systems are fixed.

4.8.2 Use case 2: AHU control logic program development fault identification and localization

Brief description

This use case describes how the application helps the control logic program developer to test and troubleshoot the control logic during program development.

Actor

The primary actor is the control logic programmer who is in charge of generating the control logic program code for the AHU system.

Preconditions

1. AHU system design intents are collected in terms of sequence of operations and control drawings (showing system component availability and data points).
2. A suite of control logic test cases are available, either manually generated or by test case generation techniques mentioned in Chapter 1.
3. Access to control logic program is available.

Basic flow of events

1. The use case begins when the control logic programmer finishes developing the control logic program for the AHU and wants to verify the control logic program implementation.
2. The control logic programmer instantiates the AHU component and control ontology based on the AHU control design intents.
3. The control logic programmer provides the instantiated ontology instance to the application.
4. The application returns derived control logic fault definition back to the user in terms of textual document and executable code.
5. The application executes the control logic program with the test cases and collects test outputs

6. The application checks the control logic test cases input and outputs for existence of control logic faults.
7. The application shows the visual and summary of control logic fault detection results.
8. The control logic programmer selects an existing control logic fault for troubleshooting.
9. The application executes the fault localization task with the selected fault.
10. The application returns ranked list of control logic program statements based on suspiciousness values calculated by the fault localization algorithm.
11. The control logic programmer debugs control logic program with the assistance of the ranked list from the previous step.
12. The application executes the modified control logic program with the test cases and verifies the symptom of the fault disappears.
13. Return to 8 if more faults need to be fixed.

Post-conditions

The updated control logic program is free of faults defined by the application.

4.8.3 Use case 3: AHU control logic fault identification and continuous detection during normal operation

Brief description

This use case describes how the control logic fault definition function of the application facilitates the implementation of control logic fault detection for AHU system continuous monitoring.

Actor

The primary actor is the implementer of fault detection rules for AHU continuous monitoring / fault detection.

Preconditions

1. Access to AHU system control input/output real time data is available.
2. A rule-based fault detection platform is available for continuous monitoring /fault detection of the HVAC systems.
3. AHU system design intents are collected in terms of sequence of operations and control drawings (showing system component availability and data points).

Basic flow of events

1. The use case begins when the fault detection rules implementer starts to work on implementing rules for continuous monitoring / fault detection for the AHU system.
2. The implementer instantiates the AHU component and control ontology based on the AHU control design intents.
3. The implementer provides the instantiated ontology instance to the application.
4. The application returns derived control logic fault definition back to the user in terms of textual document and executable code.
5. The implementer implements the fault definition from the previous step as rules in the rule-based fault detection platform.

Post-conditions

Continuous monitoring and fault detection of the AHU system control is in operation with the rules based on control logic fault definition provided by the application.

4.8.4 Use case 4: Reactive AHU control logic fault diagnosis

Brief description

This use case describes how control logic fault is diagnosed and fixed with the assistance of the application after someone (e.g. building manager) reports the existence of a control logic fault.

Actor

The primary actor is the person (user of the application) who is in charge of troubleshooting the control logic program after someone reports the existence of a control logic fault.

Preconditions

1. Access to BAS data containing control logic input/output data points
2. The existence and symptom of control logic fault is reported and available
3. Access to control logic program is available

Basic flow of events

1. The use case begins when the user starts to troubleshoot the control logic program for a reported control logic fault.
2. The user specifies the symptom of the control logic fault to troubleshoot in terms of control logic input/output variable expressions and supply it to the application.

3. The application checks the BAS data to verify the existence of control logic fault and flag the BAS data as test cases.
4. The application shows the summary of the focusing fault.
5. The application executes the fault localization task for the focusing fault with the flagged BAS data as test cases.
6. The application returns ranked list of control logic program statements based on suspiciousness values calculated by the fault localization algorithm.
7. The user finds and fixes the control logic fault causes with the assistance of the ranked list from the previous step.
8. The application executes the modified control logic program with control logic inputs from the BAS data and verifies the symptom of the fault disappears.

Post-conditions

The reported control logic fault is diagnosed and fixed by the user.

Chapter 5

Conclusions

In this chapter, I summarize the contributions of the research work presented in this thesis, echoing the research objectives discussed in Chapter 1, and discuss the practical implications and future research directions.

5.1 Contributions

In this thesis, the presented research focuses on two of the four core activities of the HVAC control logic fault identification and localization framework discussed in Section 1.4:

- Generate control logic fault definition
- Locate the cause of identified faults

The contributions made by this research can be summarized into the following five points, among which the first two are about generating control logic fault definition and the latter three are about control logic fault localization:

1. A formalism of defining applicable AHU control logic faults for specific systems

2. An AHU component and control ontology that specifies the information requirements for defining control logic faults

3. A framework of casting AHU control logic fault diagnosis problem into a software fault localization task

4. Performance evaluation of spectrum-based and mutation-based fault localization algorithms for the purpose of locating AHU control logic fault causes

5. Identification of effective mutation operators and sensitivity analysis of setup options for AHU control logic fault localization computation

Each of these research contributions is discussed in the following subsections.

A formalism of defining applicable AHU control logic faults based on system-specific information

In order to conduct AHU systems control logic programs fault detection and diagnosis, the first step is to specify a set of potential control logic faults to be checked against the behavior of the control logic. In current practice, HVAC commissioners have to subjectively adapt "rules of thumb" instructions from existing HVAC control guidelines with their own interpretations and brainstorm what the applicable types of control logic faults are for the specific AHU.

In this research, I proposed a formalized approach of deriving control logic fault definition for AHUs that is customized according to the specific AHU system information and is unambiguous in the sense that the fault is defined in terms of control logic program input/output variable mathematical expressions, and can be directly adopted by fault detection tools, as shown in the use case in Chapter 4. From the high-level objectives of occupancy comfort and energy efficiency maximization, I identified four general control goals of AHU systems, and developed corresponding reasoning algorithms to identify potential violations of these goals, i.e. to derive control logic fault definition, based on AHU system-specific information. I also developed an approach of resolving control logic fault definition conflict when merging faults defined under different goals. The proposed formalism is implemented as a prototype in this research, and the validation results shows that the prototype is able to provide customized control logic fault definition for 27 different AHUs specified by ASHRAE [1] with an average precision of 95.4%.

This contribution corresponds to research objective 1.1 discussed in Chapter 1. The developed formalism is a systematic approach to specify customized control logic fault definition. With this, the HVAC commissioners no longer need to conduct ad-hoc adaptation of general narrative instructions to identify potential control logic faults for a specific AHU. In this research, I focus on AHU systems, however, I envision that the proposed formalism can be extended to be used in other types of HVAC equipment such as variable air volume boxes, boilers, and chiller, assuming that they are also being controlled via Direct Digital Controls (DDC), and their operations also aim at certain general control objectives such as energy efficiency maximization.

An AHU component and control ontology that specifies the information requirements for defining control logic faults

In order to specify customized control logic fault definition, certain AHU system-specific information needs to be collected. These information requirements are not explicitly identified previously. Without these information requirements being explicitly identified and organized, a systematic approach of deriving control logic fault definition based on system-specific information cannot be developed.

To address this limitation, I identified the information requirements of the fault definition formalism alongside the development of the formalism. In general, These information requirements consists of three aspects of information about AHUs: 1) AHU component and function information, 2) AHU component control information, and 3) AHU service requirement information. Targeting research objective 1.3 discussed in Chapter 1, I developed an HVAC information ontology to specify these requirements in detail by integrating and extending existing HVAC BIM standards. Targeting research objective 1.2 discussed in Chapter 1, I also identified the information source of the developed ontology from BIM standards by generating a mapping table between essential information elements specified in the developed ontology and existing HVAC BIM standards, namely Brick, IFC 4 and gbXML 6.01. The developed object-oriented ontology is easily extensible to include new AHU components and functions. The ontology's ability to specify information about an AHU and support the fault definition reasoning is validated together with the fault definition formalism discussed in the previous subsection.

This contribution, together with the previous contribution, form a formalized approach that extracts information about AHU system into an ontology instance and systematically derives control logic fault definition through ontological reasoning. The ontology developed in this research focuses on information about AHU systems. In order to use the formalism for fault definition of other types of HVAC equipment, additional work needs to be conducted to identify the information requirements, develop the ontology and the goal-based reasoning algorithms, following the strategy similar to the one discussed in Chapter 2.

A framework of casting AHU control logic fault diagnosis problem into a software fault localization task

In current practice, the diagnosis of control logic faults, i.e. localization of the code in the control logic programs that causes the control logic faults, requires the HVAC control programmers/commissioners to manually read the whole control logic program to understand the logic, which is very time-consuming [11] and error-prone due to human cognition limitations.

To alleviate this problem, in this research, I aim at leveraging software fault localization techniques in the HVAC domain, to provide a computer-aided approach for AHU control logic fault diagnosis. In order to adopt software fault localization techniques, I formulated the AHU control logic fault diagnosis process into a computation framework (Figure 3.1) according to the software unit testing paradigm and the computation procedures of spectrum-based and mutation-based software fault localization techniques. In this framework, the inputs of HVAC domain artifacts are AHU control logic program source code and BAS dataset with control logic input/output variables. Based on these inputs, the framework specifies a sequence of activities to perform software fault localization computation for diagnosing AHU control logic fault causes, including 1) control logic simplification, translation and verification, 2) control logic fault detection through simulation, 3) control logic mutants generation (only needed if running mutation-based fault localization algorithms), 4) control logic program testing and execution profile extraction, and 5) fault localization algorithms execution. The framework also specifies the input and output artifacts of each activity. In this research, the implementation of this framework successfully provide fault localization algorithms computation results for 11 AHU control logic faults from 2 real-world AHU systems.

Performance evaluation of spectrum-based and mutation-based fault localization algorithms for the purpose of locating AHU control logic fault causes

The framework discussed in the previous contribution support the computations of various software fault localization algorithms. Plenty of software fault localization algorithms have been proposed in the software engineering domain, and without evaluation, it is unknown which algorithm(s) works best for the fault diagnosis of AHU control logic programs.

In this research, in order to identify a fault localization algorithm that effectively locate AHU control logic fault causes, I evaluated the performance of 39 existing mutation-based and spectrum-based fault localization techniques on 11 real-world AHU control logic fault cases from 2 AHU test beds. I identified that, for AHU control logic fault localization, the mutation-based Metallaxis method significantly outperforms all other evaluated methods. My evaluation results contradicts with existing evaluation of these techniques in the software engineering domain. The reasons of why other evaluated techniques are not as good as Metallaxis are identified.

Multiple existing user studies have shown that when a fault localization tool is able to rank a faulty statement within the top-5 position, it will help the programmer to debug much faster. The Metallaxis algorithm's performance in my evaluation reached a median value of 3.5, meaning that the programmer will reach the faulty statement within 4 statements if he/she follows the Metallaxis ranking list to diagnose AHU control logic faults. Thus, it is expected that Metallaxis can help HVAC control programmers/commissioners to diagnose control logic faults much faster than diagnosing without its help.

I also characterized the control logic fault cases in the evaluation set in terms of

1) causes of control logic faults, and 2) software code statements to fix the faults. These characteristics showed the diversity of the 11 fault cases. The relationships observed between these characteristics and the Metallaxis fault localization performance showed that Metallaxis works best when 1) the control logic program is relatively complete in terms of its logic semantic elements, and 2) when the fault can be fixed by modifying existing code rather than adding/deleting code.

Identification of effective mutation operators and sensitivity analysis of setup options for AHU control logic fault localization computation

Spectrum-based fault localization algorithms perform computation based on a test suite of passed and failed test cases. In addition to a test suite, the computation of mutation-based fault localization algorithms also requires the generation of program mutants. The setup options with regard to the test suite and mutant generation for these fault localization algorithms have direct impacts on fault localization performance. What's more, mutation-based fault localization algorithms require the execution of each test case against each program mutant, which can be computationally very expensive when the considered test cases size and mutants size are large.

In order to identify what mutation operators work best for diagnosing AHU control logic faults, I summarized effective mutants considered in the Metallaxis fault localization evaluation. I also categorized actual control logic fault fixes in terms of mutation operations. These results lead to the suggestions of three categories of mutation operators that will boost Metallaxis fault localization performance for localizing AHU control logic faults: 1) Changing literal values with other literal values of same data type; 2) Changing variable names with other variable names that have assigned values, and with literal values of the same data type as the variable to be replaced; 3) Deleting assignment statements, conditional blocks, and logical expression(s) of conditional branches.

In this research, in order to identify strategies to select test cases that provide peak fault localization performance with efficient computation, I also conducted sensitivity analysis to explore two aspects of setup options for AHU control logic fault diagnosis with software fault localization techniques. These two aspects are: 1) test suite size and down sampling methods, and 2) failed test cases ratio. The same 11 fault cases used in the fault localization evaluation are utilized in this work to evaluate the performance of a total of 29 different fault localization setup options. From the experiments results, I identified that: 1) using clustering-based down sampling methods to acquire a 100 test cases suite provides as good fault localization performance as using tens of thousands of test cases (fault localization computation time is linearly proportional to the size of test suite); 2) a safe choice of failed test cases ratio within test suite is a range from 60% to 90%.

Summary

With the aforementioned contributions, I address two major research objectives in the computation framework of HVAC control logic fault identification and diagnosis (Figure 1.3).

To facilitate the systematic derivation of customized control logic fault definition, I developed a formalism of conducting ontological reasoning about AHU system-specific information to specify applicable control logic faults for the specific AHU. The AHU information ontology used for the reasoning was developed together with the identification of its information sources from existing BIM standards. With these contributions, the HVAC professionals can specify customized control logic fault definition for AHU systems under evaluation without ad-hoc reasoning. They only need to provide factual information about the AHU system that is clearly specified in the ontology.

To help the control logic programmers/commissioners to locate the control logic fault causes inside the control logic programs more effectively, I developed a framework that facilitates the adoption of software fault localization techniques in diagnosing control logic faults. I evaluated existing spectrum-based and mutation-based fault localization techniques and identified that the mutation-based Metallaxis method outperforms others when used for diagnosing control logic faults defined by the formalism developed for the first research objective. I also explored AHU control logic fault localization setup options to identify strategies to improve computational efficiency. With these contributions, the control logic programmers/commissioners are expect to diagnose control logic faults much faster, through the fault localization tool that can perform automatic computation for each fault within tens of minutes.

These contributions address the challenges of control logic fault verification identified in Chapter 1, with limitations that will be discussed in the Section 5.3 together with future research direction.

5.2 Practical implications

The outcome of this research has the following practical implications on practitioners in the domain of HVAC system control implementation and commissioning.

Providing a complete framework to deploy systematic HVAC control logic fault verification

The HVAC control logic fault verification framework (Figure 1.3) presented in this research details a complete verification process of HVAC control logic programs. This is showcased with the use case presented in Chapter 4. The applicability of this framework makes it practical for HVAC system control practitioners, especially HVAC control manufactures, to adopt the framework and develop corresponding products/service to improve the HVAC system performance and alleviate the problem of HVAC control logic faults, which have an estimated impact of 12 trillion BTU energy wasted every year in the United States.

Providing a formalism to automatically specify control logic fault definition for HVAC FDD

I developed a formalism that can automatically derive control logic fault definition for HVAC systems based on the information of specific systems. The formalism requires factual information that can be collected from sequence of operations, control drawing submittals, and existing BIMs (e.g. IFC, gbXML) if available. The formalism provides control logic fault definition that can be directly adopted by rule-based fault detection systems. The formalism liberated commissioners from manual ad-hoc reasoning of what faults should be checked for specific AHUs under evaluation. The formalism's practicability motivates the wide adoption of control logic FDD in existing HVAC DDC systems.

Identifying a computational fault localization approach that has the potential to greatly improve control logic fault diagnosis efficiency

A framework of utilizing spectrum-based and mutation-based software fault localization algorithms to help locate control logic fault causes is proposed in the research presented in Chapter 3. My evaluation of the fault localization performance showed that, in terms of median value, the mutation-based fault localization algorithm can rank AHU control logic fault causing statements within top-5 positions. User studies showed that, when faulty statements are ranked within top-5 positions, the fault localization results can greatly help the programmer to fix the fault faster. This research is expected to motivate control manufactures to implement the proposed fault localization framework in the same software platform they provide to control programmers/commissioners to develop control logic programs. When the programmers/commissioners use the software to troubleshoot control logic programs, the fault localization functionality can help them to troubleshoot control logic faults faster.

5.3 Future work directions

Based on the work conducted in this research and its outcome, I present the following future work directions.

Studying automatic inference of HVAC information to support the reasoning of control logic fault definition

In Chapter 2, I presented the research of developing a formalism of deriving AHU control logic fault definition based on system-specific information. This system-specific information requirements are specified in the HVAC information ontology developed in the research. Although a considerate amount of information contained in the ontology can be mapped from existing BIM standards, such as IFC, gbXML, and Brick, information about AHU component functionalities, which is essential to the deriving of control logic fault definition, is not contained in these BIM standards. To use the developed fault definition approach, the user needs to specify this component functionality related information based on the sequence of operation (SOO) of the AHU system. However, this requires the user's knowledge to understand the SOO and supply ontology required information.

Ideally, every component and sensor installed in the AHU systems has a functional purpose, so the component functionality related information required by the fault definition reasoning may be inferred by reasoning about the information and arrangements of the available components and sensors installed in the AHU systems. For example, if an outdoor air flow sensor is installed in the supply air duct after the outdoor air damper, then it is very likely that the outdoor air damper will be used to provide the function of maintaining an outdoor air flow rate set point. On the other hand, if there is a separate two-stage (open/close) outdoor air damper installed in addition to the modulating outdoor air damper, then it is very likely that the modulating outdoor air damper will not have functions about supply air quality control. Based on this opinion, aiming at avoiding the need of human heuristic knowledge and manual ad-hoc work, I propose the future research direction of studying the automatic inference of AHU component functionality information based on factual information, such as the topological configuration of components and sensors inside an AHU system.

Expanding and formalizing control logic fault definition reasoning algorithms

Reasoning algorithms of violations of four general control goals are identified and developed in Chapter 2. Although these four control goals cover the majority of the control logic faults discussed in existing case studies [4], during the validation study of the fault definition prototype, I found that certain requirements of the AHUs specified in the sequence of operations are not covered by any of the four reasoning algorithms. A typical example of such requirements is the low temperature mixing box protection. Observations like this call for research to identify additional control goals and develop additional reasoning algorithms to be included in the fault definition formalism.

The needs of additional control goals and reasoning algorithms to be considered for deriving control logic fault definition also call for a more formalized understanding of the constitutes of control goals and/or fault definition reasoning algorithms to provide a search space of reasoning algorithms and a principled description of what elements form control goals and/or fault definition reasoning algorithms.

Investigating expanding the fault definition derivation formalism to domains outside of HVAC systems

The fault definition formalism developed in Chapter 2 focuses on AHU control logic fault definition. The formal classification approach adopted in the research has its generality and there is no constraint to limit its use within AHU systems.

As discussed in Chapter 1, the developed control logic fault definition derivation approach is a formalization of the reasoning procedures of people to derive system-specific requirements according to general control objectives. The problem I dealt with is the requirement specification of a cyber-physical system, in which the requirements to the control (cyber) system is based on high level objectives about the operation of the physical system, such as achieving optimal energy efficiency and providing required services with maximum capacity. Although I have not recognized similar cases in other domains, it is expected that this scenario of having requirements of a specific system coming from the application of high-level general rules is not unique. In my opinion, it is valuable to research on the generalization of the developed formalism so that similar cases in other domains, such as other cyber-physical systems, can potentially benefit from the proposed approach by adapting the approach to their use.

Expanding the control logic fault cases for fault localization evaluation

In Chapter 3, I evaluated the performance of software fault localization algorithms for diagnosing AHU control logic faults. Test beds for this evaluation work need to contain: 1) control logic programs, 2) control logic faults and corresponding fault fix (i.e. faulty statements) locations, and 3) control logic program test cases. After an intensive search on the Internet, I did not find any publicly available test beds that can be utilized in the evaluation. Thus I developed my own test beds of 2 AHUs with 11 real-world control logic faults for research work conducted in Chapter 3. Nonetheless, hardly any statistical significant conclusions can be drawn with only 11 fault cases. In order to make more convincing evaluation conclusions, more control logic fault cases are needed. Moreover, a pool of well organized real-world control logic fault cases can be very valuable to many other research topics, such as building energy simulation, control logic code analysis, etc.

Conducting code analysis of HVAC control logic and faults

In Chapter 3, my evaluation of fault localization performances of mutation-based and spectrum-based algorithms show that mutation-based Metallaxis technique outperformed other techniques considerably. This contradicts with previous studies in the software engineering domain, which showed that mutation-based techniques performed poorly on real faults of software. The contradictory results imply that there may be some fundamental differences between HVAC control logic programs and typical software applications. Research presented in this thesis cannot answer this question, which calls for future research on code analysis of HVAC control logic programs and faults.

Appendix A

AHU ontology instances of 27 ASHRAE AHUs

Listing A.1: ASHRAE AHU 1

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU1: CV1A1-XSX21"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent compType="MixingBox_ModOA" id="1" name="OAD">
        <Function funcType="SpaceCO2Decrease" id="2" name="MBCO2Decrease"
            funcRank="1">
            <EnergyEfficiency name="MBEfficiency" id="3"
                efficiency="99"></EnergyEfficiency>
            <Command name="MAO" id="56" varName="AHU1oad">
                <SaturationPoint name="MAOMax" id="4" satType="Max"
                    satValue="100"></SaturationPoint>
                <SaturationPoint name="MAOMin" id="5" satType="Min"
                    satValue="10"></SaturationPoint>
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent compType="PreheatCoil" id="18" name="PHTCoil">
        <Function funcType="SAHeating" id="19" name="PCHeating">
            <EnergyEfficiency name="PCEfficiency" id="20"
                efficiency="1.0"></EnergyEfficiency>
            <Command name="PHO" id="21" varName="AHU1pho">
                <SaturationPoint name="PHOMax" id="22" satType="Max"
                    satValue="100"></SaturationPoint>
                <SaturationPoint name="PHOMin" id="23" satType="Min"
                    satValue="0"></SaturationPoint>
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent compType="CoolingCoil" id="24" name="CHWCoil">
        <Function funcType="SACooling" id="25" name="CCCooling">
```

```
            <EnergyEfficiency name="CCEfficiency" id="26"
                efficiency="1.0"></EnergyEfficiency>
            <Command name="CCO" id="27" varName="AHU1cco">
                <SaturationPoint name="CCOMax" id="28" satType="Max"
                    satValue="100"></SaturationPoint>
                <SaturationPoint name="CCOMin" id="29" satType="Min"
                    satValue="0"></SaturationPoint>
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent compType="SupplyFan_CAV" id="30"
        name="CAVSF"></AHUComponent>
    <AHUComponent compType="Humidifier" id="31" name="Humidifier">
        <Function funcType="SAHumidification" id="32"
            name="HMHumidification">
            <EnergyEfficiency name="HMEfficiency" id="33"
                efficiency="1.0"></EnergyEfficiency>
            <Command name="HMO" id="34" varName="AHU1hmo">
                <SaturationPoint name="HMOMax" id="34" satType="Max"
                    satValue="100"></SaturationPoint>
                <SaturationPoint name="HMOMin" id="35" satType="Min"
                    satValue="0"></SaturationPoint>
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor sensorType="OutdoorAirTemperature" id="38" name="OAT"
        varName="AHU1oat"></Sensor>
    <Sensor sensorType="SupplyAirTemperature" id="39" name="SAT"
        varName="AHU1sat"></Sensor>
    <Sensor sensorType="MixedAirTemperature" id="40" name="MAT"
        varName="AHU1mat"></Sensor>
    <Sensor sensorType="ReturnAirTemperature" id="41" name="RAT"
        varName="AHU1rat"></Sensor>
    <Sensor id="42" name="SAH" sensorType="SupplyAirHumidity"
        varName="AHU1sah"></Sensor>
    <Sensor id="43" name="SpaceTemp" sensorType="SpaceTemperature"
        varName="AHU1spaceTemp"></Sensor>
    <Sensor id="44" name="SpaceCO2" sensorType="SpaceCO2"
        varName="AHU1spaceCO2"></Sensor>
    <Sensor id="45" name="RAH" sensorType="ReturnAirHumidity"
        varName="AHU1rah"></Sensor>
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="46"
        spType="UpperBound"></SetPoint>
    <SetPoint name="HeatingSP" spValue="53" id="47"
        spType="LowerBound"></SetPoint>
    <SetPoint name="SAHSP" spValue="30" id="48"
        spType="Tracking"></SetPoint>
    <SetPoint name="SACO2" spValue="1000" id="49"
        spType="UpperBound"></SetPoint>
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCO2Decrease" id="50" setPointId="49"
        feedBackId="44" outputId="56" pidType="Direct"></ClosedLoopControl>
    <ClosedLoopControl name="PHOHeating" id="53" setPointId="47"
        feedBackId="39" outputId="21" pidType="Reverse"></ClosedLoopControl>
    <ClosedLoopControl name="CCOCooling" id="54" setPointId="46"
        feedBackId="39" outputId="27" pidType="Direct"></ClosedLoopControl>
    <ClosedLoopControl name="HMOHumidification" id="55" setPointId="48"
        feedBackId="42" outputId="34" pidType="Reverse"></ClosedLoopControl>
</Project>
```

Listing A.2: ASHRAE AHU 2

```xml
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU2: CV1B1-XSX12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingEfficiency" id="10"
                opAId="29" expOperator="LT" opBId="31" />
            <Command name="MAO" id="11" varName="AHU2oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="10" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="2-Stage PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU2pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="2" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCOCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU2cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="100" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CAVSF" id="26"
        compType="SupplyFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="29" varName="AHU2oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="30" varName="AHU2mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="31" varName="AHU2rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="32" varName="AHU2sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="33" varName="AHU2spaceTemp"
        sensorType="SpaceTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="34" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="35" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="37" setPointId="34"
        feedBackId="32" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="38" setPointId="35"
        feedBackId="32" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="39" setPointId="34"
        feedBackId="32" outputId="23" pidType="Direct" />
</Project>
```

Listing A.3: ASHRAE AHU 3

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU3: CV1C1-XSX12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="29" expOperator="LT" opBId="31" />
            <Command name="MAO" id="11" varName="AHU3oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="10" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU3pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="2-Stage CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU3cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="2" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CAVSF" id="26"
        compType="SupplyFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="29" varName="AHU3oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="30" varName="AHU3mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="31" varName="AHU3rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="32" varName="AHU3sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="33" varName="AHU3spaceTemp"
```

```
        sensorType="SpaceTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="34" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="35" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="37" setPointId="34"
        feedBackId="32" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="38" setPointId="35"
        feedBackId="32" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="39" setPointId="34"
        feedBackId="32" outputId="23" pidType="Direct" />
</Project>
```

Listing A.4: ASHRAE AHU 4

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU4: CV1D1-XSX11"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="2-Stage OAD" id="1"
        compType="MixingBox_2StageOA"></AHUComponent>
    <AHUComponent name="2-Stage PHTCoil" id="2" compType="PreheatCoil">
        <Function name="PCHeating" id="3" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="4" efficiency="1.0" />
            <Command name="PHO" id="25" varName="AHU4pho">
                <SaturationPoint name="PHOMax" id="5" satType="Max"
                    satValue="2" />
                <SaturationPoint name="PHOMin" id="6" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="2-Stage CHWCoil" id="7" compType="CoolingCoil">
        <Function name="CCCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="9" efficiency="1.0" />
            <Command name="CCO" id="10" varName="AHU4cco">
                <SaturationPoint name="CCOMax" id="11" satType="Max"
                    satValue="2" />
                <SaturationPoint name="CCOMin" id="12" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CAVSF" id="13"
        compType="SupplyFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="16" varName="AHU4oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="17" varName="AHU4mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="18" varName="AHU4rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="19" varName="AHU4sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="20" varName="AHU4spaceTemp"
        sensorType="SpaceTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="21" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="22" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="PHOHeating" id="23" setPointId="22"
```

```
      feedBackId="19" outputId="25" pidType="Reverse" />
   <ClosedLoopControl name="CCOCooling" id="24" setPointId="21"
      feedBackId="19" outputId="10" pidType="Direct" />
</Project>
```

Listing A.5: ASHRAE AHU 5

```
   <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU5: CV2A1-XSX32"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
   <!-- Components -->
   <AHUComponent name="Linked OAD-MAD-EAD" id="1"
      compType="MixingBox_ModLinkOAMAEA">
      <Function name="MBCooling" id="8" funcType="SACooling">
         <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
         <FunctioningCondition name="MBCoolingCondition" id="10"
            opAId="36" expOperator="LT" opBId="38" />
         <Command name="MAO" id="11" varName="AHU5oad">
            <SaturationPoint name="MAOMax" id="12" satType="Max"
               satValue="100" />
            <SaturationPoint name="MAOMin" id="13" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="MinOAD" id="14" compType="MixingBox_ModMinOA">
      <Function name="MinOADOAFlowIncrease" id="15"
         funcType="OAFlowIncrease">
         <EnergyEfficiency name="MinOADEfficiency" id="16"
            efficiency="99" />
         <Command name="MinOAO" id="17" varName="AHU5minOad">
            <SaturationPoint name="MinOADMax" id="18" satType="Max"
               satValue="100" />
            <SaturationPoint name="MinOADMin" id="19" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="PHTCoil" id="20" compType="PreheatCoil">
      <Function name="PCHeating" id="21" funcType="SAHeating">
         <EnergyEfficiency name="PCEfficiency" id="22" efficiency="1.0"
            />
         <Command name="PHO" id="23" varName="AHU5pho">
            <SaturationPoint name="PHOMax" id="24" satType="Max"
               satValue="100" />
            <SaturationPoint name="PHOMin" id="25" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="CHWCoil" id="26" compType="CoolingCoil">
      <Function name="CCCooling" id="27" funcType="SACooling">
         <EnergyEfficiency name="CCEfficiency" id="28" efficiency="1.0"
            />
         <Command name="CCO" id="29" varName="AHU5cco">
            <SaturationPoint name="CCOMax" id="30" satType="Max"
               satValue="100" />
            <SaturationPoint name="CCOMin" id="31" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
```

```
    <AHUComponent name="CAVSF" id="32"
        compType="SupplyFan_CAV"></AHUComponent>
    <AHUComponent name="CAVRF" id="33"
        compType="ReturnFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="36" varName="AHU5oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="37" varName="AHU5mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="38" varName="AHU5rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="39" varName="AHU5sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="40" varName="AHU5spaceTemp"
        sensorType="SpaceTemperature" />
    <Sensor name="MinOAFlow" id="41" varName="AHU5minOAFlow"
        sensorType="OutdoorAirFlow" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="42" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="43" spType="LowerBound" />
    <SetPoint name="MinOAFlowSP" spValue="1500" id="49"
        spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="45" setPointId="42"
        feedBackId="39" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="MinOADOAIncrease" id="46" setPointId="49"
        feedBackId="41" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="PHOHeating" id="47" setPointId="43"
        feedBackId="39" outputId="23" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="48" setPointId="42"
        feedBackId="39" outputId="29" pidType="Direct" />
</Project>
```

Listing A.6: ASHRAE AHU 6

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU6: CV2B1-XSX12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="30" expOperator="LT" opBId="32" />
            <Command name="MAO" id="11" varName="AHU6oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="10" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="2-Stage PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU6pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="2" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU6cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="100" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CAVSF" id="26"
        compType="SupplyFan_CAV"></AHUComponent>
    <AHUComponent name="CAVRF" id="27"
        compType="ReturnFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="30" varName="AHU6oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="31" varName="AHU6mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="32" varName="AHU6rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="33" varName="AHU6sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="34" varName="AHU6spaceTemp"
        sensorType="SpaceTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="35" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="36" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="38" setPointId="35"
        feedBackId="33" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="39" setPointId="36"
        feedBackId="33" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="40" setPointId="35"
        feedBackId="33" outputId="23" pidType="Direct" />
</Project>
```

Listing A.7: ASHRAE AHU 7

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU7: CV2C1-XSX12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="30" expOperator="LT" opBId="32" />
            <Command name="MAO" id="11" varName="AHU7oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="10" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU7pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="2-Stage CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU7cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="2" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CAVSF" id="26"
        compType="SupplyFan_CAV"></AHUComponent>
    <AHUComponent name="CAVRF" id="27"
        compType="ReturnFan_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="30" varName="AHU7oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="31" varName="AHU7mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="32" varName="AHU7rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="33" varName="AHU7sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceTemp" id="34" varName="AHU7spaceTemp"
        sensorType="SpaceTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="CoolingSP" spValue="58" id="35" spType="UpperBound" />
    <SetPoint name="HeatingSP" spValue="53" id="36" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="38" setPointId="35"
        feedBackId="33" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="39" setPointId="36"
        feedBackId="33" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="40" setPointId="35"
        feedBackId="33" outputId="23" pidType="Direct" />
</Project>
```

Listing A.8: ASHRAE AHU 8

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU8: CV2D1-XSX12"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
```

```
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
               opAId="30" expOperator="LT" opBId="32" />
            <Command name="MAO" id="11" varName="AHU8oad">
               <SaturationPoint name="MAOMax" id="12" satType="Max"
                  satValue="100" />
               <SaturationPoint name="MAOMin" id="13" satType="Min"
                  satValue="10" />
            </Command>
         </Function>
      </AHUComponent>
      <AHUComponent name="2-Stage PHTCoil" id="14" compType="PreheatCoil">
         <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
               />
            <Command name="PHO" id="17" varName="AHU8pho">
               <SaturationPoint name="PHOMax" id="18" satType="Max"
                  satValue="100" />
               <SaturationPoint name="PHOMin" id="19" satType="Min"
                  satValue="0" />
            </Command>
         </Function>
      </AHUComponent>
      <AHUComponent name="2-Stage CHWCoil" id="20" compType="CoolingCoil">
         <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
               />
            <Command name="CCO" id="23" varName="AHU8cco">
               <SaturationPoint name="CCOMax" id="24" satType="Max"
                  satValue="2" />
               <SaturationPoint name="CCOMin" id="25" satType="Min"
                  satValue="0" />
            </Command>
         </Function>
      </AHUComponent>
      <AHUComponent name="CAVSF" id="26"
         compType="SupplyFan_CAV"></AHUComponent>
      <AHUComponent name="CAVRF" id="27"
         compType="ReturnFan_CAV"></AHUComponent>
      <!-- Sensors -->
      <Sensor name="OAT" id="30" varName="AHU8oat"
         sensorType="OutdoorAirTemperature" />
      <Sensor name="MAT" id="31" varName="AHU8mat"
         sensorType="MixedAirTemperature" />
      <Sensor name="RAT" id="32" varName="AHU8rat"
         sensorType="ReturnAirTemperature" />
      <Sensor name="SAT" id="33" varName="AHU8sat"
         sensorType="SupplyAirTemperature" />
      <Sensor name="SpaceTemp" id="34" varName="AHU8spaceTemp"
         sensorType="SpaceTemperature" />
      <!-- Service setpoint -->
      <SetPoint name="CoolingSP" spValue="58" id="35" spType="UpperBound" />
      <SetPoint name="HeatingSP" spValue="53" id="36" spType="LowerBound" />
      <!-- Controllers -->
      <ClosedLoopControl name="MAOCooling" id="38" setPointId="35"
         feedBackId="33" outputId="11" pidType="Direct" />
      <ClosedLoopControl name="PHOHeating" id="39" setPointId="36"
         feedBackId="33" outputId="17" pidType="Reverse" />
      <ClosedLoopControl name="CCOCooling" id="40" setPointId="35"
         feedBackId="33" outputId="23" pidType="Direct" />
   </Project>
```

Listing A.9: ASHRAE AHU 9

```xml
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU9: VAV1A1-25022"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBOAIncrease" id="2" funcType="OAFlowIncrease"
            funcRank="1">
            <EnergyEfficiency name="MBEfficiency" id="3" efficiency="99" />
            <Command name="MAO" id="4" varName="AHU9oad">
                <SaturationPoint name="MAOMax" id="5" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="6" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
        <Function name="MBCooling" id="13" funcType="SACooling"
            funcRank="2">
            <EnergyEfficiency name="MBEfficiency" id="14" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="15"
                opAId="45" expOperator="LT" opBId="47" />
            <Command name="MAO" id="16" varName="AHU9oad">
                <SaturationPoint name="MAOMax" id="17" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="18" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="19" compType="PreheatCoil">
        <Function name="PCHeating" id="20" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="21" efficiency="1.0"
                />
            <Command name="PHO" id="22" varName="AHU9pho">
                <SaturationPoint name="PHOMax" id="23" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="24" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="25" compType="CoolingCoil">
        <Function name="CCCooling" id="26" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="27" efficiency="1.0"
                />
            <Command name="CCO" id="28" varName="AHU9cco">
                <SaturationPoint name="CCOMax" id="29" satType="Max"
                    satValue="100" />
                <SaturationPoint name="CCOMin" id="30" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="31" compType="SupplyFan_VFD">
        <Function name="SFPressurization" id="32"
            funcType="SAPressurization">
            <EnergyEfficiency name="SFEfficiency" id="33" efficiency="1.0"
                />
            <Command name="SFO" id="34" varName="AHU9sfo">
                <SaturationPoint name="SFOMax" id="35" satType="Max"
                    satValue="100" />
                <SaturationPoint name="SFOMin" id="36" satType="Min"
                    satValue="30" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="Humidifier" id="37" compType="Humidifier">
        <Function name="HMHumidification" id="38"
            funcType="SAHumidification">
            <EnergyEfficiency name="HMEfficiency" id="39" efficiency="1.0"
                />
            <Command name="HMO" id="40" varName="AHU9hmo">
                <SaturationPoint name="HMOMax" id="41" satType="Max"
                    satValue="100" />
                <SaturationPoint name="HMOMin" id="42" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="45" varName="AHU9oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="46" varName="AHU9mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="47" varName="AHU9rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="48" varName="AHU9sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SAH" id="49" varName="AHU9sah"
        sensorType="SupplyAirHumidity" />
    <Sensor name="RAH" id="50" varName="AHU9rah"
        sensorType="ReturnAirHumidity" />
    <Sensor name="SpaceCO2" id="51" varName="AHU9spaceCO2"
        sensorType="SpaceCO2" />
    <Sensor name="MinOAF" id="52" varName="AHU9oaf"
        sensorType="OutdoorAirFlow" />
    <Sensor name="SSP" id="53" varName="AHU9ssp"
        sensorType="SupplyAirPressure" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="54" spValue="0.75" spType="Tracking" />
    <SetPoint name="SATSP" id="55" spValue="55" spType="Tracking" />
    <SetPoint name="RAHSP" id="56" spValue="30" spType="Tracking" />
    <SetPoint name="OAFSP" id="64" spValue="1500" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOOAFIncrease" id="57" setPointId="64"
        feedBackId="52" outputId="4" pidType="Reverse" />
    <ClosedLoopControl name="MAOCooling" id="59" setPointId="55"
        feedBackId="48" outputId="16" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="60" setPointId="55"
        feedBackId="48" outputId="22" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="61" setPointId="55"
        feedBackId="48" outputId="28" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="62" setPointId="54"
        feedBackId="53" outputId="34" pidType="Reverse" />
    <ClosedLoopControl name="HMOHumidification" id="63" setPointId="56"
        feedBackId="50" outputId="40" pidType="Reverse" />
    </Project>
```

Listing A.10: ASHRAE AHU 10

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU10: VAV1D1-11031"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
```

```
<!-- Components -->
<AHUComponent name="Linked OAD-MAD" id="1"
   compType="MixingBox_ModLinkOAMA">
   <Function name="OAFlowIncrease" id="101" funcType="OAFlowIncrease">
      <EnergyEfficiency name="MBEfficiency" id="102" efficiency="99"
         />
      <Command name="OAD" id="103" varName="AHU10oad">
         <SaturationPoint name="OADMax" id="104" satType="Max"
            satValue="100" />
         <SaturationPoint name="OADMin" id="105" satType="Min"
            satValue="10" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="2-Stage PHTCoil" id="2" compType="PreheatCoil">
   <Function name="PCHeating" id="3" funcType="SAHeating">
      <EnergyEfficiency name="PCEfficiency" id="4" efficiency="1.0" />
      <Command name="PHO" id="5" varName="AHU10pho">
         <SaturationPoint name="PHOMax" id="6" satType="Max"
            satValue="2" />
         <SaturationPoint name="PHOMin" id="7" satType="Min"
            satValue="0" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="2-Stage CHWCoil" id="8" compType="CoolingCoil">
   <Function name="CCCooling" id="9" funcType="SACooling">
      <EnergyEfficiency name="CCEfficiency" id="10" efficiency="1.0"
         />
      <Command name="CCO" id="11" varName="AHU10cco">
         <SaturationPoint name="CCOMax" id="12" satType="Max"
            satValue="2" />
         <SaturationPoint name="CCOMin" id="13" satType="Min"
            satValue="0" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="14" compType="SupplyFan_VFD">
   <Function name="SFPressurization" id="15"
      funcType="SAPressurization">
      <EnergyEfficiency name="SFEfficiency" id="16" efficiency="1.0"
         />
      <Command name="SFO" id="17" varName="AHU10sfo">
         <SaturationPoint name="SFOMax" id="18" satType="Max"
            satValue="100" />
         <SaturationPoint name="SFOMin" id="19" satType="Min"
            satValue="30" />
      </Command>
   </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="22" varName="AHU10oat"
   sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="23" varName="AHU10mat"
   sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="24" varName="AHU10rat"
   sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="25" varName="AHU10sat"
   sensorType="SupplyAirTemperature" />
<Sensor name="MinOAF" id="26" varName="AHU10minOAF"
   sensorType="OutdoorAirFlow" />
<Sensor name="SSP" id="27" varName="AHU10ssp"
   sensorType="SupplyAirPressure" />
<!-- Service setpoint -->
```

```
    <SetPoint name="SSPSP" id="28" spValue="0.75" spType="Tracking" />
    <SetPoint name="SATSP" id="29" spValue="55" spType="Tracking" />
    <SetPoint name="OAFSP" id="291" spValue="1000" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="PHOHeating" id="30" setPointId="29"
        feedBackId="25" outputId="5" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="31" setPointId="29"
        feedBackId="25" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="32" setPointId="28"
        feedBackId="27" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="OADFlowIncrease" id="321" setPointId="291"
        feedBackId="26" outputId="103" pidType="Reverse" />
</Project>
```

Listing A.11: ASHRAE AHU 11

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU11: VAV2A1-25124"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBOAIncrease" id="2" funcType="OAFlowIncrease"
            funcRank="1">
            <EnergyEfficiency name="MBEfficiency" id="3" efficiency="99" />
            <Command name="MAO" id="4" varName="AHU11oad">
                <SaturationPoint name="MAOMax" id="5" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="6" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
        <Function name="MBCooling" id="13" funcType="SACooling"
            funcRank="2">
            <EnergyEfficiency name="MBEfficiency" id="14" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="15"
                opAId="58" expOperator="LT" opBId="57" />
            <Command name="MAO" id="16" varName="AHU11oad">
                <SaturationPoint name="MAOMax" id="17" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="18" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="19" compType="PreheatCoil">
        <Function name="PCHeating" id="20" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="21" efficiency="1.0"
                />
            <Command name="PHO" id="22" varName="AHU11pho">
                <SaturationPoint name="PHOMax" id="23" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="24" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="25" compType="CoolingCoil">
        <Function name="CCCooling" id="26" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="27" efficiency="1.0"
                />
            <Command name="CCO" id="28" varName="AHU11cco">
            <SaturationPoint name="CCOMax" id="29" satType="Max"
               satValue="100" />
            <SaturationPoint name="CCOMin" id="30" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="VFDSF" id="31" compType="SupplyFan_VFD">
      <Function name="SFPressurization" id="32"
         funcType="SAPressurization">
         <EnergyEfficiency name="SFEfficiency" id="33" efficiency="1.0"
            />
         <Command name="SFO" id="34" varName="AHU11sfo">
            <SaturationPoint name="SFOMax" id="35" satType="Max"
               satValue="100" />
            <SaturationPoint name="SFOMin" id="36" satType="Min"
               satValue="30" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="VFDRF" id="37" compType="ReturnFan_VFD">
      <Function name="RFO" id="371" funcType="MAPressurization">
         <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
         <Command name="RFO" id="373" varName="AHU11rfo">
            <SaturationPoint name="RFOMax" id="374" satType="Max"
               satValue="100" />
            <SaturationPoint name="RFOMin" id="375" satType="Min"
               satValue="30" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
      <Function name="SpaceDepressurization" id="39"
         funcType="SpaceDepressurization">
         <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
         <Command name="EAO" id="41" varName="AHU11ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
               satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <!-- Sensors -->
   <Sensor name="OAT" id="46" varName="AHU11oat"
      sensorType="OutdoorAirTemperature" />
   <Sensor name="MAT" id="47" varName="AHU11mat"
      sensorType="MixedAirTemperature" />
   <Sensor name="RAT" id="48" varName="AHU11rat"
      sensorType="ReturnAirTemperature" />
   <Sensor name="SAT" id="49" varName="AHU11sat"
      sensorType="SupplyAirTemperature" />
   <Sensor name="RAH" id="50" varName="AHU11rah"
      sensorType="ReturnAirHumidity" />
   <Sensor name="OAH" id="51" varName="AHU11oah"
      sensorType="OutdoorAirHumidity" />
   <Sensor name="SpaceCO2" id="52" varName="AHU11co2"
      sensorType="SpaceCO2" />
   <Sensor name="MinOAF" id="53" varName="AHU11minoaf"
      sensorType="OutdoorAirFlow" />
   <Sensor name="SSP" id="54" varName="AHU11ssp"
      sensorType="SupplyAirPressure" />
   <Sensor name="SpaceAP" id="55" varName="AHU11spaceap"
```

```
        sensorType="SpacePressure" />
    <Sensor name="RAP" id="56" varName="AHU11rap"
        sensorType="ReturnAirPressure" />
    <Sensor name="RAE" id="57" varName="AHU11rae"
        sensorType="ReturnAirEnthalpy" />
    <Sensor name="OAE" id="58" varName="AHU11oae"
        sensorType="OutdoorAirEnthalpy" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="59" spValue="0.75" spType="Tracking" />
    <SetPoint name="SpaceAPSP" id="60" spValue="0.02" spType="Tracking" />
    <SetPoint name="SATSP" id="61" spValue="55" spType="Tracking" />
    <SetPoint name="MinOAFSP" id="62" spValue="1500" spType="LowerBound" />
    <SetPoint name="MAPSP" id="621" spValue="0.1" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOOAIncrease" id="63" setPointId="62"
        feedBackId="53" outputId="4" pidType="Reverse" />
    <ClosedLoopControl name="MAOCooling" id="65" setPointId="61"
        feedBackId="49" outputId="16" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="66" setPointId="61"
        feedBackId="49" outputId="22" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="67" setPointId="61"
        feedBackId="49" outputId="28" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="68" setPointId="59"
        feedBackId="54" outputId="34" pidType="Reverse" />
    <ClosedLoopControl name="EAOSpaceDepressurization" id="69"
        setPointId="60" feedBackId="55" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="56" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.12: ASHRAE AHU 12

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU12: VAV2A2-21232"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="41" expOperator="LT" opBId="43" />
            <Command name="MAO" id="11" varName="AHU12oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="10" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="MinOAD" id="14" compType="MixingBox_ModMinOA">
        <Function name="MinOADOAIncrease" id="15" funcType="OAFlowIncrease">
            <EnergyEfficiency name="MinOADEfficiency" id="16"
                efficiency="99" />
            <Command name="MinOAD" id="17" varName="AHU12minoad">
                <SaturationPoint name="MinOADMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MinOADMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
```

```
</AHUComponent>
<AHUComponent name="PHTCoil" id="20" compType="PreheatCoil">
    <Function name="PCHeating" id="21" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="22" efficiency="1.0"
            />
        <Command name="PHO" id="23" varName="AHU12pho">
            <SaturationPoint name="PHOMax" id="24" satType="Max"
                satValue="100" />
            <SaturationPoint name="PHOMin" id="25" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="CHWCoil" id="26" compType="CoolingCoil">
    <Function name="CCCooling" id="27" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="28" efficiency="1.0"
            />
        <Command name="CCO" id="29" varName="AHU12cco">
            <SaturationPoint name="CCOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="CCOMin" id="31" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="32" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="33"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="SFO" id="35" varName="AHU12sfo">
            <SaturationPoint name="SFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="38" compType="ReturnFan_VFD">
    <Function name="RFRAFlowIncrease" id="381"
        funcType="RAFlowIncrease">
        <EnergyEfficiency name="RFEfficiency" id="382" efficiency="1.0"
            />
        <Command name="RFO" id="383" varName="AHU12rfo">
            <SaturationPoint name="RFOMax" id="384" satType="Max"
                satValue="100" />
            <SaturationPoint name="RFOMin" id="385" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="41" varName="AHU12oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="42" varName="AHU12mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="43" varName="AHU12rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="44" varName="AHU12sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="MinOAF" id="45" varName="AHU12minOAF"
    sensorType="OutdoorAirFlow" />
<Sensor name="SSP" id="46" varName="AHU12ssp"
    sensorType="SupplyAirPressure" />
    <Sensor name="SAF" id="47" varName="AHU12saf"
       sensorType="SupplyAirFlow" />
    <Sensor name="RAF" id="48" varName="AHU12raf"
       sensorType="ReturnAirFlow" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="49" spValue="0.75" spType="Tracking" />
    <SetPoint name="SATSP" id="50" spValue="55" spType="Tracking" />
    <SetPoint name="MinOAFlowSP" id="51" spValue="1500"
       spType="LowerBound" />
    <SetPoint name="RAFSP" id="511" spValue="1200" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="53" setPointId="50"
       feedBackId="44" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="MinOADOAIncrease" id="54" setPointId="51"
       feedBackId="45" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="PHOHeating" id="55" setPointId="50"
       feedBackId="44" outputId="23" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="56" setPointId="50"
       feedBackId="44" outputId="29" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="57" setPointId="49"
       feedBackId="46" outputId="35" pidType="Reverse" />
    <ClosedLoopControl name="RFORAIncrease" id="571" setPointId="511"
       feedBackId="48" outputId="383" pidType="Reverse" />
</Project>
```

Listing A.13: ASHRAE AHU 13

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU13: VAV2B1-15132"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
       compType="MixingBox_ModLinkOAMA">
       <Function name="MBCooling" id="8" funcType="SACooling">
          <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
          <FunctioningCondition name="MBCoolingCondition" id="10"
             opAId="41" expOperator="LT" opBId="43" />
          <Command name="MAO" id="11" varName="AHU13oad">
             <SaturationPoint name="MAOMax" id="12" satType="Max"
                satValue="100" />
             <SaturationPoint name="MAOMin" id="13" satType="Min"
                satValue="10" />
          </Command>
       </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage PHTCoil" id="14" compType="PreheatCoil">
       <Function name="PCHeating" id="15" funcType="SAHeating">
          <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
             />
          <Command name="PHO" id="17" varName="AHU13pho">
             <SaturationPoint name="PHOMax" id="18" satType="Max"
                satValue="4" />
             <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
          </Command>
       </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
       <Function name="CCCooling" id="21" funcType="SACooling">
          <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
             />
```

```
      <Command name="CCO" id="23" varName="AHU13cco">
         <SaturationPoint name="CCOMax" id="24" satType="Max"
            satValue="100" />
         <SaturationPoint name="CCOMin" id="25" satType="Min"
            satValue="0" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
   <Function name="SFPressurization" id="27"
      funcType="SAPressurization">
      <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
         />
      <Command name="SFO" id="29" varName="AHU13sfo">
         <SaturationPoint name="SFOMax" id="30" satType="Max"
            satValue="100" />
         <SaturationPoint name="SFOMin" id="31" satType="Min"
            satValue="30" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="32" compType="ReturnFan_VFD">
   <Function name="RFO" id="371" funcType="MAPressurization">
      <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
      <Command name="RFO" id="373" varName="AHU13rfo">
         <SaturationPoint name="RFOMax" id="374" satType="Max"
            satValue="100" />
         <SaturationPoint name="RFOMin" id="375" satType="Min"
            satValue="30" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="EAD" id="33" compType="MixingBox_ModEA">
   <Function name="SpaceDepressurization" id="34"
      funcType="SpaceDepressurization">
      <EnergyEfficiency name="EADEfficiency" id="35" efficiency="99"
         />
      <Command name="EAO" id="36" varName="AHU13ead">
         <SaturationPoint name="EAOMax" id="37" satType="Max"
            satValue="100" />
         <SaturationPoint name="EAOMin" id="38" satType="Min"
            satValue="0" />
      </Command>
   </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="41" varName="AHU13oat"
   sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="42" varName="AHU13mat"
   sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="43" varName="AHU13rat"
   sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="44" varName="AHU13sat"
   sensorType="SupplyAirTemperature" />
<Sensor name="SSP" id="45" varName="AHU13ssp"
   sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="46" varName="AHU13spaceap"
   sensorType="SpacePressure" />
<Sensor name="RAP" id="47" varName="AHU13rap"
   sensorType="ReturnAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="48" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="49" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="50" spValue="55" spType="Tracking" />
```

```
    <SetPoint name="MAPSP" id="621" spValue="0.1" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="52" setPointId="50"
       feedBackId="44" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="53" setPointId="50"
       feedBackId="44" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="54" setPointId="50"
       feedBackId="44" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="55" setPointId="48"
       feedBackId="45" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="SpaceDepressurization" id="56"
       setPointId="49" feedBackId="46" outputId="36" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
       setPointId="621" feedBackId="47" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.14: ASHRAE AHU 14

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU14: VAV2B2-25142"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
       compType="MixingBox_ModLinkOAMA">
       <Function name="MBCooling" id="8" funcType="SACooling">
          <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
          <FunctioningCondition name="MBCoolingCondition" id="10"
             opAId="42" expOperator="LT" opBId="44" />
          <Command name="MAO" id="11" varName="AHU14oad">
             <SaturationPoint name="MAOMax" id="12" satType="Max"
                satValue="100" />
             <SaturationPoint name="MAOMin" id="13" satType="Min"
                satValue="10" />
          </Command>
       </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage PHTCoil" id="14" compType="PreheatCoil">
       <Function name="PCHeating" id="15" funcType="SAHeating">
          <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
             />
          <Command name="PHO" id="17" varName="AHU14pho">
             <SaturationPoint name="PHOMax" id="18" satType="Max"
                satValue="4" />
             <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
          </Command>
       </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
       <Function name="CCCooling" id="21" funcType="SACooling">
          <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
             />
          <Command name="CCO" id="23" varName="AHU14cco">
             <SaturationPoint name="CCOMax" id="24" satType="Max"
                satValue="100" />
             <SaturationPoint name="CCOMin" id="25" satType="Min"
                satValue="0" />
          </Command>
       </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
       <Function name="SFPressurization" id="27"
```

```
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
           />
        <Command name="SFO" id="29" varName="AHU14sfo">
           <SaturationPoint name="SFOMax" id="30" satType="Max"
              satValue="100" />
           <SaturationPoint name="SFOMin" id="31" satType="Min"
              satValue="30" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="VFDRF" id="32" compType="ReturnFan_VFD">
     <Function name="RFO" id="371" funcType="MAPressurization">
        <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
        <Command name="RFO" id="373" varName="AHU14rfo">
           <SaturationPoint name="RFOMax" id="374" satType="Max"
              satValue="100" />
           <SaturationPoint name="RFOMin" id="375" satType="Min"
              satValue="30" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="EAD" id="33" compType="MixingBox_ModEA">
     <Function name="SpaceDepressurization" id="34"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="35" efficiency="99"
           />
        <Command name="EAO" id="36" varName="AHU14ead">
           <SaturationPoint name="EAOMax" id="37" satType="Max"
              satValue="100" />
           <SaturationPoint name="EAOMin" id="38" satType="Min"
              satValue="0" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="MinOAD" id="39"
     compType="MixingBox_ModMinOA"></AHUComponent>
  <!-- Sensors -->
  <Sensor name="OAT" id="42" varName="AHU14oat"
     sensorType="OutdoorAirTemperature" />
  <Sensor name="MAT" id="43" varName="AHU14mat"
     sensorType="MixedAirTemperature" />
  <Sensor name="RAT" id="44" varName="AHU14rat"
     sensorType="ReturnAirTemperature" />
  <Sensor name="SAT" id="45" varName="AHU14sat"
     sensorType="SupplyAirTemperature" />
  <Sensor name="SSP" id="46" varName="AHU14ssp"
     sensorType="SupplyAirPressure" />
  <Sensor name="SpaceAP" id="47" varName="AHU14spaceap"
     sensorType="SpacePressure" />
  <Sensor name="RAP" id="48" varName="AHU14rap"
     sensorType="ReturnAirPressure" />
  <Sensor name="MAP" id="49" varName="AHU14map"
     sensorType="MixedAirPressure" />
  <!-- Service setpoint -->
  <SetPoint name="SSPSP" id="50" spValue="0.75" spType="Tracking" />
  <SetPoint name="SpaceAPSP" id="51" spValue="0.02" spType="Tracking" />
  <SetPoint name="SATSP" id="52" spValue="55" spType="Tracking" />
  <SetPoint name="MAPSP" id="621" spValue="0.1" spType="Tracking" />
  <!-- Controllers -->
  <ClosedLoopControl name="MAOCooling" id="54" setPointId="52"
     feedBackId="45" outputId="11" pidType="Direct" />
  <ClosedLoopControl name="PHOHeating" id="55" setPointId="52"
```

```
        feedBackId="45" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="56" setPointId="52"
        feedBackId="45" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="57" setPointId="50"
        feedBackId="46" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="SpaceDepressurization" id="58"
        setPointId="51" feedBackId="47" outputId="36" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="48" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.15: ASHRAE AHU 15

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU15: VAV2C1-25133"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBOAIncrease" id="2" funcType="OAFlowIncrease"
            funcRank="1">
            <EnergyEfficiency name="MBEfficiency" id="3" efficiency="99" />
            <Command name="MAO" id="4" varName="AHU15oad">
                <SaturationPoint name="MAOMax" id="5" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="6" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
        <Function name="MBCooling" id="13" funcType="SACooling"
            funcRank="2">
            <EnergyEfficiency name="MBEfficiency" id="14" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="15"
                opAId="46" expOperator="LT" opBId="48" />
            <Command name="MAO" id="16" varName="AHU15oad">
                <SaturationPoint name="MAOMax" id="17" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="18" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="19" compType="PreheatCoil">
        <Function name="PCHeating" id="20" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="21" efficiency="1.0"
                />
            <Command name="PHO" id="22" varName="AHU15pho">
                <SaturationPoint name="PHOMax" id="23" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="24" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage CHWCoil" id="25" compType="CoolingCoil">
        <Function name="CCCooling" id="26" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="27" efficiency="1.0"
                />
            <Command name="CCO" id="28" varName="AHU15cco">
                <SaturationPoint name="CCOMax" id="29" satType="Max"
                    satValue="4" />
                <SaturationPoint name="CCOMin" id="30" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="31" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="32"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="33" efficiency="1.0"
            />
        <Command name="SFO" id="34" varName="AHU15sfo">
            <SaturationPoint name="SFOMax" id="35" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="36" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="37" compType="ReturnFan_VFD">
    <Function name="RFO" id="371" funcType="MAPressurization">
        <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
        <Command name="RFO" id="373" varName="AHU15rfo">
            <SaturationPoint name="RFOMax" id="374" satType="Max"
                satValue="100" />
            <SaturationPoint name="RFOMin" id="375" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
    <Function name="SpaceDepressurization" id="39"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
        <Command name="EAO" id="41" varName="AHU15ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="46" varName="AHU15oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="47" varName="AHU15mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="48" varName="AHU15rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="49" varName="AHU15sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="MinOAF" id="50" varName="AHU15minoaf"
    sensorType="OutdoorAirFlow" />
<Sensor name="SSP" id="51" varName="AHU15ssp"
    sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="52" varName="AHU15spaceap"
    sensorType="SpacePressure" />
<Sensor name="RAP" id="53" varName="AHU15rap"
    sensorType="ReturnAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="54" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="55" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="56" spValue="55" spType="Tracking" />
<SetPoint name="MinOAFSP" id="57" spValue="1500" spType="LowerBound" />
```

```
    <SetPoint name="MAPSP" id="621" spValue="0.1" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOOAIncrease" id="58" setPointId="57"
        feedBackId="50" outputId="4" pidType="Reverse" />
    <ClosedLoopControl name="MAOCooling" id="60" setPointId="56"
        feedBackId="49" outputId="16" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="61" setPointId="56"
        feedBackId="49" outputId="22" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="62" setPointId="56"
        feedBackId="49" outputId="28" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="63" setPointId="54"
        feedBackId="51" outputId="34" pidType="Reverse" />
    <ClosedLoopControl name="EAOSpaceDepressurization" id="64"
        setPointId="55" feedBackId="52" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="53" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.16: ASHRAE AHU 16

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU16: VAV2C2-11142"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="42" expOperator="LT" opBId="44" />
            <Command name="MAO" id="11" varName="AHU16oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU16pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU16cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="4" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
```

```
</AHUComponent>
<AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
   <Function name="SFPressurization" id="27"
      funcType="SAPressurization">
      <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
         />
      <Command name="SFO" id="29" varName="AHU16sfo">
         <SaturationPoint name="SFOMax" id="30" satType="Max"
            satValue="100" />
         <SaturationPoint name="SFOMin" id="31" satType="Min"
            satValue="30" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="32" compType="ReturnFan_VFD">
   <Function name="RFO" id="371" funcType="MAPressurization">
      <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
      <Command name="RFO" id="373" varName="AHU16rfo">
         <SaturationPoint name="RFOMax" id="374" satType="Max"
            satValue="100" />
         <SaturationPoint name="RFOMin" id="375" satType="Min"
            satValue="30" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="EAD" id="33" compType="MixingBox_ModEA">
   <Function name="SpaceDepressurization" id="34"
      funcType="SpaceDepressurization">
      <EnergyEfficiency name="EADEfficiency" id="35" efficiency="99"
         />
      <Command name="EAO" id="36" varName="AHU16ead">
         <SaturationPoint name="EAOMax" id="37" satType="Max"
            satValue="100" />
         <SaturationPoint name="EAOMin" id="38" satType="Min"
            satValue="0" />
      </Command>
   </Function>
</AHUComponent>
<AHUComponent name="MinOAD" id="39"
   compType="MixingBox_ModMinOA"></AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="42" varName="AHU16oat"
   sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="43" varName="AHU16mat"
   sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="44" varName="AHU16rat"
   sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="45" varName="AHU16sat"
   sensorType="SupplyAirTemperature" />
<Sensor name="SSP" id="46" varName="AHU16ssp"
   sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="47" varName="AHU16spaceap"
   sensorType="SpacePressure" />
<Sensor name="RAP" id="48" varName="AHU16rap"
   sensorType="ReturnAirPressure" />
<Sensor name="MAP" id="49" varName="AHU16map"
   sensorType="MixedAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="50" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="51" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="52" spValue="55" spType="Tracking" />
<SetPoint name="MAPSP" id="621" spValue="0.1" spType="Tracking" />
<!-- Controllers -->
<ClosedLoopControl name="MAOCooling" id="54" setPointId="52"
```

```
        feedBackId="45" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="55" setPointId="52"
        feedBackId="45" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="56" setPointId="52"
        feedBackId="45" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="57" setPointId="50"
        feedBackId="46" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="SpaceDepressurization" id="58"
        setPointId="51" feedBackId="47" outputId="36" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="48" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.17: ASHRAE AHU 17

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU17: VAV2D1-11132"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBOAIncrease" id="2" funcType="OAFlowIncrease"
            funcRank="1">
            <EnergyEfficiency name="MBEfficiency" id="3" efficiency="99" />
            <Command name="MAO" id="4" varName="AHU17oad">
                <SaturationPoint name="MAOMax" id="5" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="6" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
        <Function name="MBCooling" id="13" funcType="SACooling"
            funcRank="2">
            <EnergyEfficiency name="MBEfficiency" id="14" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="15"
                opAId="46" expOperator="LT" opBId="48" />
            <Command name="MAO" id="16" varName="AHU17oad">
                <SaturationPoint name="MAOMax" id="17" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="18" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage PHTCoil" id="19" compType="PreheatCoil">
        <Function name="PCHeating" id="20" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="21" efficiency="1.0"
                />
            <Command name="PHO" id="22" varName="AHU17pho">
                <SaturationPoint name="PHOMax" id="23" satType="Max"
                    satValue="4" />
                <SaturationPoint name="PHOMin" id="24" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage CHWCoil" id="25" compType="CoolingCoil">
        <Function name="CCCooling" id="26" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="27" efficiency="1.0"
                />
            <Command name="CCO" id="28" varName="AHU17cco">
                <SaturationPoint name="CCOMax" id="29" satType="Max"
                satValue="4" />
            <SaturationPoint name="CCOMin" id="30" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="31" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="32"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="33" efficiency="1.0"
            />
        <Command name="SFO" id="34" varName="AHU17sfo">
            <SaturationPoint name="SFOMax" id="35" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="36" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="37" compType="ReturnFan_VFD">
    <Function name="RFO" id="371" funcType="MAPressurization">
        <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
        <Command name="RFO" id="373" varName="AHU17rfo">
            <SaturationPoint name="RFOMax" id="374" satType="Max"
                satValue="100" />
            <SaturationPoint name="RFOMin" id="375" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
    <Function name="SpaceDepressurization" id="39"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
        <Command name="EAO" id="41" varName="AHU17ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="46" varName="AHU17oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="47" varName="AHU17mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="48" varName="AHU17rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="49" varName="AHU17sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="MinOAF" id="50" varName="AHU17minoaf"
    sensorType="OutdoorAirFlow" />
<Sensor name="SSP" id="51" varName="AHU17ssp"
    sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="52" varName="AHU17spaceap"
    sensorType="SpacePressure" />
<Sensor name="RAP" id="53" varName="AHU17rap"
    sensorType="ReturnAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="54" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="55" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="56" spValue="55" spType="Tracking" />
```

```
    <SetPoint name="MinOAFSP" id="57" spValue="1500" spType="LowerBound" />
    <SetPoint name="RAPSP" id="621" spValue="0.1" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOOAIncrease" id="58" setPointId="57"
        feedBackId="50" outputId="4" pidType="Reverse" />
    <ClosedLoopControl name="MAOCooling" id="60" setPointId="56"
        feedBackId="49" outputId="16" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="61" setPointId="56"
        feedBackId="49" outputId="22" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="62" setPointId="56"
        feedBackId="49" outputId="28" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="63" setPointId="54"
        feedBackId="51" outputId="34" pidType="Reverse" />
    <ClosedLoopControl name="EAOSpaceDepressurization" id="64"
        setPointId="55" feedBackId="52" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="53" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.18: ASHRAE AHU 18

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU18: VAV2D2-15232"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD-EAD" id="1"
        compType="MixingBox_ModLinkOAMAEA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="41" expOperator="LT" opBId="43" />
            <Command name="MAO" id="11" varName="AHU18oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="MinOAD" id="14" compType="MixingBox_ModMinOA">
        <Function name="MinOADOAIncrease" id="15" funcType="OAFlowIncrease">
            <EnergyEfficiency name="MinOADEfficiency" id="16"
                efficiency="99" />
            <Command name="MinOAD" id="17" varName="AHU18minoad">
                <SaturationPoint name="MinOADMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MinOADMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage PHTCoil" id="20" compType="PreheatCoil">
        <Function name="PCHeating" id="21" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="PHO" id="23" varName="AHU18pho">
                <SaturationPoint name="PHOMax" id="24" satType="Max"
                    satValue="4" />
                <SaturationPoint name="PHOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
    </Function>
</AHUComponent>
<AHUComponent name="4-Stage CHWCoil" id="26" compType="CoolingCoil">
    <Function name="CCCooling" id="27" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="28" efficiency="1.0"
            />
        <Command name="CCO" id="29" varName="AHU18cco">
            <SaturationPoint name="CCOMax" id="30" satType="Max"
                satValue="4" />
            <SaturationPoint name="CCOMin" id="31" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="32" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="33"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="SFO" id="35" varName="AHU18sfo">
            <SaturationPoint name="SFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="38" compType="ReturnFan_VFD">
    <Function name="RFRAFlowIncrease" id="381"
        funcType="RAFlowIncrease">
        <EnergyEfficiency name="RFEfficiency" id="382" efficiency="1.0"
            />
        <Command name="RFO" id="383" varName="AHU18rfo">
            <SaturationPoint name="RFOMax" id="384" satType="Max"
                satValue="100" />
            <SaturationPoint name="RFOMin" id="385" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="41" varName="AHU18oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="42" varName="AHU18mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="43" varName="AHU18rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="44" varName="AHU18sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="MinOAF" id="45" varName="AHU18minOAF"
    sensorType="OutdoorAirFlow" />
<Sensor name="SSP" id="46" varName="AHU18ssp"
    sensorType="SupplyAirPressure" />
<Sensor name="SAF" id="47" varName="AHU18saf"
    sensorType="SupplyAirFlow" />
<Sensor name="RAF" id="48" varName="AHU18raf"
    sensorType="ReturnAirFlow" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="49" spValue="0.75" spType="Tracking" />
<SetPoint name="SATSP" id="50" spValue="55" spType="Tracking" />
<SetPoint name="MinOAFlowSP" id="51" spValue="1500"
    spType="LowerBound" />
<SetPoint name="RAFSP" id="511" spValue="1200" spType="Tracking" />
<!-- Controllers -->
```

```
    <ClosedLoopControl name="MAOCooling" id="53" setPointId="50"
        feedBackId="44" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="MinOADOAIncrease" id="54" setPointId="51"
        feedBackId="45" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="PHOHeating" id="55" setPointId="50"
        feedBackId="44" outputId="23" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="56" setPointId="50"
        feedBackId="44" outputId="29" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="57" setPointId="49"
        feedBackId="46" outputId="35" pidType="Reverse" />
    <ClosedLoopControl name="RFORAIncrease" id="571" setPointId="511"
        feedBackId="48" outputId="383" pidType="Reverse" />
</Project>
```

Listing A.19: ASHRAE AHU 19

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU19: VAV3A1-25153"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="43" expOperator="LT" opBId="45" />
            <Command name="MAO" id="11" varName="AHU19oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU19pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU19cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="100" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
        <Function name="SFPressurization" id="27"
            funcType="SAPressurization">
            <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
            />
        <Command name="SFO" id="29" varName="AHU19sfo">
            <SaturationPoint name="SFOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="31" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDRF" id="32" compType="ReturnFan_VFD">
    <Function name="RFO" id="371" funcType="MAPressurization">
        <EnergyEfficiency name="RFEfficiency" id="372" efficiency="1" />
        <Command name="RFO" id="373" varName="AHU19rfo">
            <SaturationPoint name="RFOMax" id="374" satType="Max"
                satValue="100" />
            <SaturationPoint name="RFOMin" id="375" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="EAD" id="33" compType="MixingBox_ModEA">
    <Function name="SpaceDepressurization" id="34"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="35" efficiency="99"
            />
        <Command name="EAO" id="36" varName="AHU19ead">
            <SaturationPoint name="EAOMax" id="37" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="38" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="2-Stage MinOAD" id="39"
    compType="MixingBox_2StageOA"></AHUComponent>
<AHUComponent name="CAVInjection" id="40"
    compType="Injection_CAV"></AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="43" varName="AHU19oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="44" varName="AHU19mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="45" varName="AHU19rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="46" varName="AHU19sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="PHT" id="47" varName="AHU19pht"
    sensorType="PreheatAirTemperature" />
<Sensor name="SSP" id="48" varName="AHU19ssp"
    sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="49" varName="AHU19spaceap"
    sensorType="SpacePressure" />
<Sensor name="RAP" id="50" varName="AHU19rap"
    sensorType="ReturnAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="51" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="52" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="53" spValue="55" spType="Tracking" />
<SetPoint name="RAPSP" id="621" spValue="0.1" spType="Tracking" />
<!-- Controllers -->
<ClosedLoopControl name="MAOCooling" id="55" setPointId="53"
    feedBackId="46" outputId="11" pidType="Direct" />
<ClosedLoopControl name="PHOHeating" id="56" setPointId="53"
```

```
        feedBackId="46" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="57" setPointId="53"
        feedBackId="46" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="58" setPointId="51"
        feedBackId="48" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="EAOSpaceDepressurization" id="59"
        setPointId="52" feedBackId="49" outputId="36" pidType="Direct" />
    <ClosedLoopControl name="RFOMAPressurization" id="691"
        setPointId="621" feedBackId="50" outputId="373" pidType="Reverse" />
</Project>
```

Listing A.20: ASHRAE AHU 20

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU20: VAV3C1-25154"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="54" expOperator="LT" opBId="53" />
            <Command name="MAO" id="11" varName="AHU20oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU20pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU20cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="4" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
        <Function name="SFPressurization" id="27"
            funcType="SAPressurization">
            <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
                />
            <Command name="SFO" id="29" varName="AHU20sfo">
                <SaturationPoint name="SFOMax" id="30" satType="Max"
                    satValue="100" />
                <SaturationPoint name="SFOMin" id="31" satType="Min"
                    satValue="30" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDRF" id="32"
        compType="ReturnFan_VFD"></AHUComponent>
    <AHUComponent name="EAD" id="33"
        compType="MixingBox_ModEA"></AHUComponent>
    <AHUComponent name="2-Stage MinOAD" id="39"
        compType="MixingBox_2StageOA"></AHUComponent>
    <AHUComponent name="CAVInjection" id="40"
        compType="Injection_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="43" varName="AHU20oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="44" varName="AHU20mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="45" varName="AHU20rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="46" varName="AHU20sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SSP" id="48" varName="AHU20ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="49" varName="AHU20spaceap"
        sensorType="SpacePressure" />
    <Sensor name="RAP" id="50" varName="AHU20rap"
        sensorType="ReturnAirPressure" />
    <Sensor name="RAH" id="51" varName="AHU20rah"
        sensorType="ReturnAirHumidity" />
    <Sensor name="OAH" id="52" varName="AHU20oah"
        sensorType="OutdoorAirHumidity" />
    <Sensor name="RAE" id="53" varName="AHU20rae"
        sensorType="ReturnAirEnthalpy" />
    <Sensor name="OAE" id="54" varName="AHU20oae"
        sensorType="OutdoorAirEnthalpy" />
    <Sensor name="PHT" id="55" varName="AHU20pht"
        sensorType="PreheatAirTemperature" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="56" spValue="0.75" spType="Tracking" />
    <SetPoint name="SATSP" id="58" spValue="55" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="60" setPointId="58"
        feedBackId="46" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="61" setPointId="58"
        feedBackId="46" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="62" setPointId="58"
        feedBackId="46" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="63" setPointId="56"
        feedBackId="48" outputId="29" pidType="Reverse" />
</Project>
```

Listing A.21: ASHRAE AHU 21

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU21: VAV3C2-15252"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
        <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
        <FunctioningCondition name="MBCoolingCondition" id="10"
           opAId="43" expOperator="LT" opBId="45" />
        <Command name="MAO" id="11" varName="AHU21oad">
           <SaturationPoint name="MAOMax" id="12" satType="Max"
              satValue="100" />
           <SaturationPoint name="MAOMin" id="13" satType="Min"
              satValue="0" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
     <Function name="PCHeating" id="15" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
           />
        <Command name="PHO" id="17" varName="AHU21pho">
           <SaturationPoint name="PHOMax" id="18" satType="Max"
              satValue="100" />
           <SaturationPoint name="PHOMin" id="19" satType="Min"
              satValue="0" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
     <Function name="CCCooling" id="21" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
           />
        <Command name="CCO" id="23" varName="AHU21cco">
           <SaturationPoint name="CCOMax" id="24" satType="Max"
              satValue="4" />
           <SaturationPoint name="CCOMin" id="25" satType="Min"
              satValue="0" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
     <Function name="SFPressurization" id="27"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
           />
        <Command name="SFO" id="29" varName="AHU21sfo">
           <SaturationPoint name="SFOMax" id="30" satType="Max"
              satValue="100" />
           <SaturationPoint name="SFOMin" id="31" satType="Min"
              satValue="30" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="VFDRF" id="32" compType="ReturnFan_VFD">
     <Function name="RFRAFlowIncrease" id="381"
        funcType="RAFlowIncrease">
        <EnergyEfficiency name="RFEfficiency" id="382" efficiency="1.0"
           />
        <Command name="RFO" id="383" varName="AHU21rfo">
           <SaturationPoint name="RFOMax" id="384" satType="Max"
              satValue="100" />
           <SaturationPoint name="RFOMin" id="385" satType="Min"
              satValue="30" />
        </Command>
     </Function>
  </AHUComponent>
  <AHUComponent name="EAD" id="33"
     compType="MixingBox_ModEA"></AHUComponent>
  <AHUComponent name="2-Stage MinOAD" id="39"
```

```
        compType="MixingBox_2StageOA"></AHUComponent>
    <AHUComponent name="CAVInjection" id="40"
        compType="Injection_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="43" varName="AHU21oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="44" varName="AHU21mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="45" varName="AHU21rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="46" varName="AHU21sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="PHT" id="47" varName="AHU21pht"
        sensorType="PreheatAirTemperature" />
    <Sensor name="SSP" id="48" varName="AHU21ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="49" varName="AHU21spaceap"
        sensorType="SpacePressure" />
    <Sensor name="RAF" id="50" varName="AHU21raf"
        sensorType="ReturnAirFlow" />
    <Sensor name="SAF" id="60" varName="AHU21saf"
        sensorType="SupplyAirFlow" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="51" spValue="0.75" spType="Tracking" />
    <SetPoint name="SATSP" id="53" spValue="55" spType="Tracking" />
    <SetPoint name="RAFSP" id="511" spValue="1200" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="55" setPointId="53"
        feedBackId="46" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="56" setPointId="53"
        feedBackId="46" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="57" setPointId="53"
        feedBackId="46" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="58" setPointId="51"
        feedBackId="48" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="RFORAIncrease" id="571" setPointId="511"
        feedBackId="50" outputId="383" pidType="Reverse" />
</Project>
```

Listing A.22: ASHRAE AHU 22

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU22: VAV4A1-25322"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="52" expOperator="LT" opBId="54" />
            <Command name="MAO" id="11" varName="AHU22oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
            />
        <Command name="PHO" id="17" varName="AHU22pho">
            <SaturationPoint name="PHOMax" id="18" satType="Max"
                satValue="100" />
            <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
    <Function name="CCCooling" id="21" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
            />
        <Command name="CCO" id="23" varName="AHU22cco">
            <SaturationPoint name="CCOMax" id="24" satType="Max"
                satValue="100" />
            <SaturationPoint name="CCOMin" id="25" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="27"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
            />
        <Command name="SFO" id="29" varName="AHU22sfo">
            <SaturationPoint name="SFOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="31" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
    <Function name="EFDepressurization" id="33"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="EFO" id="35" varName="AHU22efo">
            <SaturationPoint name="EFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="EFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
    <Function name="EADDepressurization" id="39"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
        <Command name="EAO" id="41" varName="AHU22ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="MinOAD" id="44" compType="MixingBox_ModMinOA">
    <Function name="MinOADOAIncrease" id="45" funcType="OAFlowIncrease">
        <EnergyEfficiency name="MinOADEfficiency" id="46"
```

```
                efficiency="99" />
            <Command name="MinMAO" id="47" varName="AHU22minoad">
                <SaturationPoint name="MinOADMax" id="48" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MinOADMin" id="49" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="52" varName="AHU22oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="53" varName="AHU22mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="54" varName="AHU22rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="55" varName="AHU22sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="SpaceCO2" id="56" varName="AHU22spaceCO2"
        sensorType="SpaceCO2" />
    <Sensor name="MinOAF" id="57" varName="AHU22minoaf"
        sensorType="OutdoorAirFlow" />
    <Sensor name="SSP" id="58" varName="AHU22ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="59" varName="AHU22spaceap"
        sensorType="SpacePressure" />
    <Sensor name="MAP" id="60" varName="AHU22map"
        sensorType="MixedAirPressure" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="61" spValue="0.75" spType="Tracking" />
    <SetPoint name="SpaceAPSP" id="62" spValue="0.02" spType="Tracking" />
    <SetPoint name="SATSP" id="63" spValue="55" spType="Tracking" />
    <SetPoint name="MinOAFSP" id="64" spValue="1500" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="66" setPointId="63"
        feedBackId="55" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="67" setPointId="63"
        feedBackId="55" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="68" setPointId="63"
        feedBackId="55" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="69" setPointId="61"
        feedBackId="58" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="EFODepressurization" id="70" setPointId="62"
        feedBackId="59" outputId="35" pidType="Direct" />
    <ClosedLoopControl name="EAODepressurization" id="71" setPointId="62"
        feedBackId="59" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="MinMAOOAIncrease" id="72" setPointId="64"
        feedBackId="57" outputId="47" pidType="Reverse" />
</Project>
```

Listing A.23: ASHRAE AHU 23

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU23: VAV4B1-25322"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
            opAId="52" expOperator="LT" opBId="54" />
         <Command name="MAO" id="11" varName="AHU23oad">
            <SaturationPoint name="MAOMax" id="12" satType="Max"
               satValue="100" />
            <SaturationPoint name="MAOMin" id="13" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="4-Stage PHTCoil" id="14" compType="PreheatCoil">
      <Function name="PCHeating" id="15" funcType="SAHeating">
         <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
            />
         <Command name="PHO" id="17" varName="AHU23pho">
            <SaturationPoint name="PHOMax" id="18" satType="Max"
               satValue="4" />
            <SaturationPoint name="PHOMin" id="19" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
      <Function name="CCCooling" id="21" funcType="SACooling">
         <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
            />
         <Command name="CCO" id="23" varName="AHU23cco">
            <SaturationPoint name="CCOMax" id="24" satType="Max"
               satValue="100" />
            <SaturationPoint name="CCOMin" id="25" satType="Min"
               satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
      <Function name="SFPressurization" id="27"
         funcType="SAPressurization">
         <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
            />
         <Command name="SFO" id="29" varName="AHU23sfo">
            <SaturationPoint name="SFOMax" id="30" satType="Max"
               satValue="100" />
            <SaturationPoint name="SFOMin" id="31" satType="Min"
               satValue="30" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
      <Function name="EFDepressurization" id="33"
         funcType="SpaceDepressurization">
         <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
            />
         <Command name="EFO" id="35" varName="AHU23efo">
            <SaturationPoint name="EFOMax" id="36" satType="Max"
               satValue="100" />
            <SaturationPoint name="EFOMin" id="37" satType="Min"
               satValue="30" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
      <Function name="EADDepressurization" id="39"
         funcType="SpaceDepressurization">
         <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
            <Command name="EAO" id="41" varName="AHU23ead">
                <SaturationPoint name="EAOMax" id="42" satType="Max"
                    satValue="100" />
                <SaturationPoint name="EAOMin" id="43" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="MinOAD" id="44" compType="MixingBox_ModMinOA">
        <Function name="MinOADOAIncrease" id="45" funcType="OAFlowIncrease">
            <EnergyEfficiency name="MinOADEfficiency" id="46"
                efficiency="99" />
            <Command name="MinMAO" id="47" varName="AHU23minoad">
                <SaturationPoint name="MinOADMax" id="48" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MinOADMin" id="49" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="52" varName="AHU23oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="53" varName="AHU23mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="54" varName="AHU23rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="55" varName="AHU23sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="MinOAF" id="57" varName="AHU23minoaf"
        sensorType="OutdoorAirFlow" />
    <Sensor name="SSP" id="58" varName="AHU23ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="59" varName="AHU23spaceap"
        sensorType="SpacePressure" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="61" spValue="0.75" spType="Tracking" />
    <SetPoint name="SpaceAPSP" id="62" spValue="0.02" spType="Tracking" />
    <SetPoint name="SATSP" id="63" spValue="55" spType="Tracking" />
    <SetPoint name="MinOAFSP" id="64" spValue="1500" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="66" setPointId="63"
        feedBackId="55" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="67" setPointId="63"
        feedBackId="55" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="68" setPointId="63"
        feedBackId="55" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="69" setPointId="61"
        feedBackId="58" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="EFODepressurization" id="70" setPointId="62"
        feedBackId="59" outputId="35" pidType="Direct" />
    <ClosedLoopControl name="EAODepressurization" id="71" setPointId="62"
        feedBackId="59" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="MinMAOOAIncrease" id="72" setPointId="64"
        feedBackId="57" outputId="47" pidType="Reverse" />
</Project>
```

Listing A.24: ASHRAE AHU 24

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU24: VAV4C1-25333"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
<!-- Components -->
<AHUComponent name="Linked OAD-MAD" id="1"
    compType="MixingBox_ModLinkOAMA">
    <Function name="MBCooling" id="8" funcType="SACooling">
        <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
        <FunctioningCondition name="MBCoolingCondition" id="10"
            opAId="52" expOperator="LT" opBId="54" />
        <Command name="MAO" id="11" varName="AHU24oad">
            <SaturationPoint name="MAOMax" id="12" satType="Max"
                satValue="100" />
            <SaturationPoint name="MAOMin" id="13" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
    <Function name="PCHeating" id="15" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
            />
        <Command name="PHO" id="17" varName="AHU24pho">
            <SaturationPoint name="PHOMax" id="18" satType="Max"
                satValue="100" />
            <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
    <Function name="CCCooling" id="21" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
            />
        <Command name="CCO" id="23" varName="AHU24cco">
            <SaturationPoint name="CCOMax" id="24" satType="Max"
                satValue="4" />
            <SaturationPoint name="CCOMin" id="25" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="27"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
            />
        <Command name="SFO" id="29" varName="AHU24sfo">
            <SaturationPoint name="SFOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="31" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
    <Function name="EFDepressurization" id="33"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="EFO" id="35" varName="AHU24efo">
            <SaturationPoint name="EFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="EFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
        <Function name="EADDepressurization" id="39"
            funcType="SpaceDepressurization">
            <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
                />
            <Command name="EAO" id="41" varName="AHU24ead">
                <SaturationPoint name="EAOMax" id="42" satType="Max"
                    satValue="100" />
                <SaturationPoint name="EAOMin" id="43" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="MinOAD" id="44" compType="MixingBox_ModMinOA">
        <Function name="MinOADOAIncrease" id="45" funcType="OAFlowIncrease">
            <EnergyEfficiency name="MinOADEfficiency" id="46"
                efficiency="99" />
            <Command name="MinMAO" id="47" varName="AHU24minoad">
                <SaturationPoint name="MinOADMax" id="48" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MinOADMin" id="49" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="52" varName="AHU24oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="53" varName="AHU24mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="54" varName="AHU24rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="55" varName="AHU24sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="MinOAF" id="57" varName="AHU24minoaf"
        sensorType="OutdoorAirFlow" />
    <Sensor name="SSP" id="58" varName="AHU24ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="59" varName="AHU24spaceap"
        sensorType="SpacePressure" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="61" spValue="0.75" spType="Tracking" />
    <SetPoint name="SpaceAPSP" id="62" spValue="0.02" spType="Tracking" />
    <SetPoint name="SATSP" id="63" spValue="55" spType="Tracking" />
    <SetPoint name="MinOAFSP" id="64" spValue="1500" spType="LowerBound" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="66" setPointId="63"
        feedBackId="55" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="67" setPointId="63"
        feedBackId="55" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="68" setPointId="63"
        feedBackId="55" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="69" setPointId="61"
        feedBackId="58" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="EFODepressurization" id="70" setPointId="62"
        feedBackId="59" outputId="35" pidType="Direct" />
    <ClosedLoopControl name="EAODepressurization" id="71" setPointId="62"
        feedBackId="59" outputId="41" pidType="Direct" />
    <ClosedLoopControl name="MinMAOOAIncrease" id="72" setPointId="64"
        feedBackId="57" outputId="47" pidType="Reverse" />
</Project>
```

Listing A.25: ASHRAE AHU 25

```
    <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU25: VAV4D1-11342"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="52" expOperator="LT" opBId="54" />
            <Command name="MAO" id="11" varName="AHU25oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU25pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="4" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU25cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="4" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
        <Function name="SFPressurization" id="27"
            funcType="SAPressurization">
            <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
                />
            <Command name="SFO" id="29" varName="AHU25sfo">
                <SaturationPoint name="SFOMax" id="30" satType="Max"
                    satValue="100" />
                <SaturationPoint name="SFOMin" id="31" satType="Min"
                    satValue="30" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
        <Function name="EFDepressurization" id="33"
            funcType="SpaceDepressurization">
            <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
                />
            <Command name="EFO" id="35" varName="AHU25efo">
                <SaturationPoint name="EFOMax" id="36" satType="Max"
```

```xml
                satValue="100" />
            <SaturationPoint name="EFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
    <Function name="EADDepressurization" id="39"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
        <Command name="EAO" id="41" varName="AHU25ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="MinOAD" id="44"
    compType="MixingBox_ModMinOA"></AHUComponent>
<!-- Sensors -->
<Sensor name="OAT" id="52" varName="AHU25oat"
    sensorType="OutdoorAirTemperature" />
<Sensor name="MAT" id="53" varName="AHU25mat"
    sensorType="MixedAirTemperature" />
<Sensor name="RAT" id="54" varName="AHU25rat"
    sensorType="ReturnAirTemperature" />
<Sensor name="SAT" id="55" varName="AHU25sat"
    sensorType="SupplyAirTemperature" />
<Sensor name="SpaceCO2" id="56" varName="AHU25spaceCO2"
    sensorType="SpaceCO2" />
<Sensor name="MAP" id="57" varName="AHU25map"
    sensorType="MixedAirPressure" />
<Sensor name="SSP" id="58" varName="AHU25ssp"
    sensorType="SupplyAirPressure" />
<Sensor name="SpaceAP" id="59" varName="AHU25spaceap"
    sensorType="SpacePressure" />
<Sensor name="MAP" id="60" varName="AHU25map"
    sensorType="MixedAirPressure" />
<!-- Service setpoint -->
<SetPoint name="SSPSP" id="61" spValue="0.75" spType="Tracking" />
<SetPoint name="SpaceAPSP" id="62" spValue="0.02" spType="Tracking" />
<SetPoint name="SATSP" id="63" spValue="55" spType="Tracking" />
<!-- Controllers -->
<ClosedLoopControl name="MAOCooling" id="66" setPointId="63"
    feedBackId="55" outputId="11" pidType="Direct" />
<ClosedLoopControl name="PHOHeating" id="67" setPointId="63"
    feedBackId="55" outputId="17" pidType="Reverse" />
<ClosedLoopControl name="CCOCooling" id="68" setPointId="63"
    feedBackId="55" outputId="23" pidType="Direct" />
<ClosedLoopControl name="SFOPressurization" id="69" setPointId="61"
    feedBackId="58" outputId="29" pidType="Reverse" />
<ClosedLoopControl name="EFODepressurization" id="70" setPointId="62"
    feedBackId="59" outputId="35" pidType="Direct" />
<ClosedLoopControl name="EAODepressurization" id="71" setPointId="62"
    feedBackId="59" outputId="41" pidType="Direct" />
</Project>
```

Listing A.26: ASHRAE AHU 26

```xml
    <?xml version="1.0" encoding="UTF-8"?>
```

```
<Project name="AHU26: VAV5A1-25352"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
    <!-- Components -->
    <AHUComponent name="Linked OAD-MAD" id="1"
        compType="MixingBox_ModLinkOAMA">
        <Function name="MBCooling" id="8" funcType="SACooling">
            <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
            <FunctioningCondition name="MBCoolingCondition" id="10"
                opAId="48" expOperator="LT" opBId="50" />
            <Command name="MAO" id="11" varName="AHU26oad">
                <SaturationPoint name="MAOMax" id="12" satType="Max"
                    satValue="100" />
                <SaturationPoint name="MAOMin" id="13" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
        <Function name="PCHeating" id="15" funcType="SAHeating">
            <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
                />
            <Command name="PHO" id="17" varName="AHU26pho">
                <SaturationPoint name="PHOMax" id="18" satType="Max"
                    satValue="100" />
                <SaturationPoint name="PHOMin" id="19" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="CHWCoil" id="20" compType="CoolingCoil">
        <Function name="CCCooling" id="21" funcType="SACooling">
            <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
                />
            <Command name="CCO" id="23" varName="AHU26cco">
                <SaturationPoint name="CCOMax" id="24" satType="Max"
                    satValue="100" />
                <SaturationPoint name="CCOMin" id="25" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
        <Function name="SFPressurization" id="27"
            funcType="SAPressurization">
            <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
                />
            <Command name="SFO" id="29" varName="AHU26sfo">
                <SaturationPoint name="SFOMax" id="30" satType="Max"
                    satValue="100" />
                <SaturationPoint name="SFOMin" id="31" satType="Min"
                    satValue="30" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
        <Function name="EFDepressurization" id="33"
            funcType="SpaceDepressurization">
            <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
                />
            <Command name="EFO" id="35" varName="AHU26efo">
                <SaturationPoint name="EFOMax" id="36" satType="Max"
                    satValue="100" />
                <SaturationPoint name="EFOMin" id="37" satType="Min"
                satValue="30" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
      <Function name="EADDepressurization" id="39"
         funcType="SpaceDepressurization">
         <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
            />
         <Command name="EAO" id="41" varName="AHU26ead">
            <SaturationPoint name="EAOMax" id="42" satType="Max"
                satValue="100" />
            <SaturationPoint name="EAOMin" id="43" satType="Min"
                satValue="0" />
         </Command>
      </Function>
   </AHUComponent>
   <AHUComponent name="MinOAD" id="44"
      compType="MixingBox_2StageOA"></AHUComponent>
   <AHUComponent name="CAVInjection" id="45"
      compType="Injection_CAV"></AHUComponent>
   <!-- Sensors -->
   <Sensor name="OAT" id="48" varName="AHU26oat"
      sensorType="OutdoorAirTemperature" />
   <Sensor name="MAT" id="49" varName="AHU26mat"
      sensorType="MixedAirTemperature" />
   <Sensor name="RAT" id="50" varName="AHU26rat"
      sensorType="ReturnAirTemperature" />
   <Sensor name="SAT" id="51" varName="AHU26sat"
      sensorType="SupplyAirTemperature" />
   <Sensor name="PHT" id="52" varName="AHU26pht"
      sensorType="PreheatAirTemperature" />
   <Sensor name="SSP" id="53" varName="AHU26ssp"
      sensorType="SupplyAirPressure" />
   <Sensor name="SpaceAP" id="54" varName="AHU26spaceap"
      sensorType="SpacePressure" />
   <!-- Service setpoint -->
   <SetPoint name="SSPSP" id="55" spValue="0.75" spType="Tracking" />
   <SetPoint name="SpaceAPSP" id="56" spValue="0.02" spType="Tracking" />
   <SetPoint name="SATSP" id="57" spValue="55" spType="Tracking" />
   <!-- Controllers -->
   <ClosedLoopControl name="MAOCooling" id="59" setPointId="57"
      feedBackId="51" outputId="11" pidType="Direct" />
   <ClosedLoopControl name="PHOHeating" id="60" setPointId="57"
      feedBackId="51" outputId="17" pidType="Reverse" />
   <ClosedLoopControl name="CCOCooling" id="61" setPointId="57"
      feedBackId="51" outputId="23" pidType="Direct" />
   <ClosedLoopControl name="SFOPressurization" id="62" setPointId="55"
      feedBackId="53" outputId="29" pidType="Reverse" />
   <ClosedLoopControl name="EFODepressurization" id="63" setPointId="56"
      feedBackId="54" outputId="35" pidType="Direct" />
   <ClosedLoopControl name="EAODepressurization" id="64" setPointId="56"
      feedBackId="54" outputId="41" pidType="Direct" />
</Project>
```

Listing A.27: ASHRAE AHU 27

```
   <?xml version="1.0" encoding="UTF-8"?>
<Project name="AHU27: VAV5C1-25352"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="AHUSchema_v2_3.xsd">
   <!-- Components -->
```

```
<AHUComponent name="Linked OAD-MAD" id="1"
    compType="MixingBox_ModLinkOAMA">
    <Function name="MBCooling" id="8" funcType="SACooling">
        <EnergyEfficiency name="MBEfficiency" id="9" efficiency="99" />
        <FunctioningCondition name="MBCoolingCondition" id="10"
            opAId="48" expOperator="LT" opBId="50" />
        <Command name="MAO" id="11" varName="AHU27oad">
            <SaturationPoint name="MAOMax" id="12" satType="Max"
                satValue="100" />
            <SaturationPoint name="MAOMin" id="13" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="PHTCoil" id="14" compType="PreheatCoil">
    <Function name="PCHeating" id="15" funcType="SAHeating">
        <EnergyEfficiency name="PCEfficiency" id="16" efficiency="1.0"
            />
        <Command name="PHO" id="17" varName="AHU27pho">
            <SaturationPoint name="PHOMax" id="18" satType="Max"
                satValue="100" />
            <SaturationPoint name="PHOMin" id="19" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="4-Stage CHWCoil" id="20" compType="CoolingCoil">
    <Function name="CCCooling" id="21" funcType="SACooling">
        <EnergyEfficiency name="CCEfficiency" id="22" efficiency="1.0"
            />
        <Command name="CCO" id="23" varName="AHU27cco">
            <SaturationPoint name="CCOMax" id="24" satType="Max"
                satValue="4" />
            <SaturationPoint name="CCOMin" id="25" satType="Min"
                satValue="0" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDSF" id="26" compType="SupplyFan_VFD">
    <Function name="SFPressurization" id="27"
        funcType="SAPressurization">
        <EnergyEfficiency name="SFEfficiency" id="28" efficiency="1.0"
            />
        <Command name="SFO" id="29" varName="AHU27sfo">
            <SaturationPoint name="SFOMax" id="30" satType="Max"
                satValue="100" />
            <SaturationPoint name="SFOMin" id="31" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
<AHUComponent name="VFDEF" id="32" compType="ReliefFan_VFD">
    <Function name="EFDepressurization" id="33"
        funcType="SpaceDepressurization">
        <EnergyEfficiency name="EFEfficiency" id="34" efficiency="1.0"
            />
        <Command name="EFO" id="35" varName="AHU27efo">
            <SaturationPoint name="EFOMax" id="36" satType="Max"
                satValue="100" />
            <SaturationPoint name="EFOMin" id="37" satType="Min"
                satValue="30" />
        </Command>
    </Function>
</AHUComponent>
```

```
    <AHUComponent name="EAD" id="38" compType="MixingBox_ModEA">
        <Function name="EADDepressurization" id="39"
            funcType="SpaceDepressurization">
            <EnergyEfficiency name="EADEfficiency" id="40" efficiency="99"
                />
            <Command name="EAO" id="41" varName="AHU27ead">
                <SaturationPoint name="EAOMax" id="42" satType="Max"
                    satValue="100" />
                <SaturationPoint name="EAOMin" id="43" satType="Min"
                    satValue="0" />
            </Command>
        </Function>
    </AHUComponent>
    <AHUComponent name="MinOAD" id="44"
        compType="MixingBox_2StageOA"></AHUComponent>
    <AHUComponent name="CAVInjection" id="45"
        compType="Injection_CAV"></AHUComponent>
    <!-- Sensors -->
    <Sensor name="OAT" id="48" varName="AHU27oat"
        sensorType="OutdoorAirTemperature" />
    <Sensor name="MAT" id="49" varName="AHU27mat"
        sensorType="MixedAirTemperature" />
    <Sensor name="RAT" id="50" varName="AHU27rat"
        sensorType="ReturnAirTemperature" />
    <Sensor name="SAT" id="51" varName="AHU27sat"
        sensorType="SupplyAirTemperature" />
    <Sensor name="PHT" id="52" varName="AHU27pht"
        sensorType="PreheatAirTemperature" />
    <Sensor name="SSP" id="53" varName="AHU27ssp"
        sensorType="SupplyAirPressure" />
    <Sensor name="SpaceAP" id="54" varName="AHU27spaceap"
        sensorType="SpacePressure" />
    <!-- Service setpoint -->
    <SetPoint name="SSPSP" id="55" spValue="0.75" spType="Tracking" />
    <SetPoint name="SpaceAPSP" id="56" spValue="0.02" spType="Tracking" />
    <SetPoint name="SATSP" id="57" spValue="55" spType="Tracking" />
    <!-- Controllers -->
    <ClosedLoopControl name="MAOCooling" id="59" setPointId="57"
        feedBackId="51" outputId="11" pidType="Direct" />
    <ClosedLoopControl name="PHOHeating" id="60" setPointId="57"
        feedBackId="51" outputId="17" pidType="Reverse" />
    <ClosedLoopControl name="CCOCooling" id="61" setPointId="57"
        feedBackId="51" outputId="23" pidType="Direct" />
    <ClosedLoopControl name="SFOPressurization" id="62" setPointId="55"
        feedBackId="53" outputId="29" pidType="Reverse" />
    <ClosedLoopControl name="EFODepressurization" id="63" setPointId="56"
        feedBackId="54" outputId="35" pidType="Direct" />
    <ClosedLoopControl name="EAODepressurization" id="64" setPointId="56"
        feedBackId="54" outputId="41" pidType="Direct" />
</Project>
```

Appendix B

Control logic fault causes categorized as mutations

Fault ID	No.	Actual / Alternate fix	LOC	Change Type	MutOp type	Mutate from	Mutate to
PAM Fault2	1	Actual	72	Modify	LVR	POS	POS
	2	Alternate	134	Modify	EVR	<IDENTIFIER>	POS
	3	Actual	158	Modify	EVR	<IDENTIFIER>as Method Argument	<METHOD_INVOCATION>
	4	Alternate	186	Modify	EVR	<IDENTIFIER>as Method Argument	<METHOD_INVOCATION>
	5	Actual	177	Modify	EVR	<IDENTIFIER>as Method Argument	<IDENTIFIER>
PAM Fault7	6	Actual	161	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	7	Alternate	198	Modify	EVR	<IDENTIFIER>as Method Argument	POS(1)
	8	Alternate	223	Modify	EVR	<IDENTIFIER>	<IDENTIFIER>
	9	Alternate	141	Modify	EVR	<IDENTIFIER>	NEG
	10	Alternate	67	Modify	LVR	POS	NEG
PAM Fault9	11	Actual	174	Modify	EVR	<IDENTIFIER>as Method Argument	<IDENTIFIER>
	12	Alternate	219	Modify	EVR	<IDENTIFIER>	<METHOD_INVOCATION>
PAM Fault10	13	Actual	81	Modify	LVR	0	POS
	14	Alternate	182	Modify	EVR	Arithmetic Expression	<IDENTIFIER>
	15	Alternate	185	Modify	EVR	<METHOD_INVOCATION>	<IDENTIFIER>
	16	Alternate	220	Modify	EVR	<IDENTIFIER>	<IDENTIFIER>
PAM Fault15	17	Actual	161	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	18	Alternate	198	Modify	EVR	<IDENTIFIER>as Method Argument	POS(1)
	19	Alternate	223	Modify	EVR	<IDENTIFIER>	<IDENTIFIER>
	20	Alternate	141	Modify	EVR	<METHOD_INVOCATION>	NEG
	21	Alternate	67	Modify	LVR	POS	NEG
	22	Actual	177	Modify	EVR	<IDENTIFIER>as Method Argument	<IDENTIFIER>
Delta Fault2	23	Actual	247	Modify	COR/ROR	Add logic expression(s)	
	24	Actual	256	Modify	COR/ROR	Add logic expression(s)	
	25	Actual	266	Delete	STD	Condition Block	<NO-OP>
	26	Actual	268	Delete	STD	Condition Block	<NO-OP>
	27	Actual	269	Delete	STD	Assignment Statement	<NO-OP>
	28	Actual	270	Delete	STD	Condition Block	<NO-OP>
Delta Fault3	29	Actual	188	Modify	COR/ROR	Delete one logical expression	
	30	Alternate	166	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	31	Actual	189	Modify	COR/ROR	Delete one logical expression	
	32	Alternate	167	Modify	EVR	<METHOD_INVOCATION>	0
	33	Actual	190	Delete	STD	Condition Block	<NO-OP>
	34	Alternate	172	Modify	LVR	0	POS
	35	Actual	192	Delete	STD	Condition Block	<NO-OP>
	36	Actual	193	Delete	STD	Assignment Statement	<NO-OP>
	37	Actual	194	Delete	STD	Condition Block	<NO-OP>
	38	Alternate	213	Modify	EVR	<IDENTIFIER>	<IDENTIFIER>
					LVR	POS	0
Delta Fault4	39-41	Actual	107	Modify	LVR	0	POS
					EVR	<IDENTIFIER>as Method Argument	<IDENTIFIER>
	42	Alternate	110	Modify	COR/ROR	Delete multiple logical expressions	
	43	Alternate	115	Modify	COR/ROR	Delete one logical expression	
	44	Actual	111	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	45	Actual	116	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	46	Actual	142	Modify	COR/ROR	Delete one logical expression	
	47	Actual	143	Modify	LVR	POS	POS
	48	Actual	148	Modify	COR/ROR	Add logic expression(s)	
	49	Actual	149	Modify	LVR	POS	POS
					LVR	POS	0
DeltaFault11	50-52	Actual	107	Modify	LVR	0	POS
					EVR	<IDENTIFIER>as Method Argument	<IDENTIFIER>
	53	Alternate	110	Modify	COR/ROR	Delete multiple logical expressions	
	54	Alternate	115	Modify	COR/ROR	Delete one logical expression	
	55	Actual	111	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	56	Actual	116	Modify	EVR	<METHOD_INVOCATION>	POS(1)
	57	Actual	142	Modify	COR/ROR	Delete one logical expression	
	58	Actual	143	Modify	LVR	POS	POS
	59	Actual	148	Modify	COR/ROR	Add logic expression(s)	
	60	Actual	149	Modify	LVR	POS	POS

Table B.1: Actual fault fixing code changes categorized by mutation operation

Bibliography

[1] ASHRAE, *Sequences of Operation for Common HVAC Systems.* American Society of Heating Refrigerating and Air-Conditioning Engineers, 2005.

[2] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[3] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proceedings of the 39th International Conference on Software Engineering.* IEEE Press, 2017, pp. 609–620.

[4] M. Ardehali and T. Smith, "Literature review to identify existing case studies of controls-related energy-inefficiencies in buildings," *Department of Mechanical and Industrial Engineering, The University of Iowa, Technical Report: ME-TFS-01-007*, 2002.

[5] F. E. Barwig, J. M. House, C. J. Klaassen, M. M. Ardehali, and T. F. Smith, "The national building controls information program," in *Proc. ACEEE summer study on energy efficiency in buildings.* Citeseer, 2002.

[6] K. W. Roth, D. Westphalen, M. Y. Feng, P. Llana, and L. Quartararo, "Energy Impact of Commercial Building Controls and Performance Diagnostics: Market Characterization, Energy Impact of Building Faults and Energy Savings Potential," *U.S. Department of Energy*, 2005.

[7] K. Gillespie, "A general commissioning acceptance procedure for ddc systems in commissioning test protocol library," Report, 2002.

[8] S. Katipamula and M. R. Brambley, "Review article: methods for fault detection, diagnostics, and prognostics for building systemsâĂŤa review, part i," *HVAC&R Research*, vol. 11, no. 1, pp. 3–25, 2005.

[9] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering.* ACM, 2005, pp. 273–282.

[10] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[11] K. Bruton, P. Raftery, B. Kennedy, M. M. Keane, and D. O'sullivan, "Review of automated fault detection and diagnostic tools in air handling units," *Energy efficiency*, vol. 7, no. 2, pp. 335–351, 2014.

[12] J. House and H. Vaezi-Nejad, "An expert rule set for fault detection in air-handling units," *ASHRAE Transactions*, vol. 107, no. ASHRAE Transactions, 2001.

[13] J. Schein, S. T. Bushby, N. S. Castro, and J. M. House, "A rule-based fault detection method for air handling units," *Energy and buildings*, vol. 38, no. 12, pp. 1485–1492, 2006.

[14] A. Platzer, "Logic and compositional verification of hybrid systems," in *International Conference on Computer Aided Verification.* Springer, 2011, pp. 28–43.

[15] M. E. Fagan, *Design and Code Inspections to Reduce Errors in Program Development.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 301–334. [Online]. Available: https://doi.org/10.1007/978-3-642-48354-7_13

[16] E. Doolan, "Experience with fagan's inspection method," *Software: Practice and Experience*, vol. 22, no. 2, pp. 173–182, 1992.

[17] H. Remus, "Integrated software validation in the view of inspections/reviews," in *Proc. of a symposium on Software validation: inspection-testing-verification-alternatives.* Elsevier North-Holland, Inc., 1984, pp. 57–64.

[18] A. F. Ackerman, L. S. Buchwald, and F. H. Lewski, "Software inspections: an effective verification process," *IEEE software*, vol. 6, no. 3, pp. 31–36, 1989.

[19] M. E. Fagan, "Advances in software inspections," in *Pioneers and Their Contributions to Software Engineering.* Springer, 2001, pp. 335–360.

[20] O. Laitenberger, "A survey of software inspection technologies," in *Handbook of Software Engineering and Knowledge Engineering: Volume II: Emerging Technologies.* World Scientific, 2002, pp. 517–555.

[21] E. M. Clarke Jr, O. Grumberg, and D. Peled, *Model checking.* MIT press, 1999.

[22] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer, "Formal verification of obstacle avoidance and navigation of ground robots," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1312–1340, 2017.

[23] M. L. Minsky, *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

[24] A. Pnueli, "The temporal logic of programs," in *Foundations of Computer Science, 1977., 18th Annual Symposium on.* IEEE, 1977, pp. 46–57.

[25] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid systems.* Springer, 1993, pp. 209–229.

[26] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, A. Bertolino *et al.*, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

[27] A. Mili and F. Tchier, *Software testing: Concepts and operations.* John Wiley & Sons, 2015.

[28] P. Bourque, R. E. Fairley *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press, 2014.

[29] "Ieee standard for software unit testing," *ANSI/IEEE Std 1008-1987*, 1986.

[30] W. G. Vincenti *et al.*, *What engineers know and how they know it.* Baltimore: Johns Hopkins University Press, 1990, vol. 141.

[31] ASHRAE, *ASHRAE Handbook - HVAC Applications (I-P Edition)*, 2015.

[32] ——, *ASHRAE Handbook: Heating, Ventilating, and Air-conditioning Systems and Equipment.* American Society of Heating Refrigerating and Air-Conditioning Engineers, 2012.

[33] ——, *ASHRAE Guideline 11 Field Testing of HVAC Controls Components*, 2009.

[34] ——, *BSR/ASHRAE Guideline 36P High Performance Sequences of Operation for HVAC Systems (Advisory Public Review Draft)*, 2015, no. April.

[35] A. C. Group. (2016-06-26) Acg commissioning guideline. [Online]. Available: http://www.commissioning.org/commissioningguideline/

[36] P. E. C. Inc. (2016-06-26) Functional testing guide: from the fundamentals to the field. [Online]. Available: http://www.ftguide.org/ftg/index.htm

[37] A. Martin and C. Banyard, "Application guide ag7/98: library of system control strategies," *The Building Services Research and Information Association (BSRIA), ISBN 0-86022*, vol. 497, 1998.

[38] R. McDowall and R. Montgomery, *Fundamentals of HVAC Control Systems.* ASHRAE, 2011, vol. SI ed. [Online]. Available: http://search.ebscohost.com.proxy.library.cmu.edu/login.aspx?direct=true&db=nlebk&AN=1542424&site=ehost-live

[39] A. Dexter and J. Pakanen, *Demonstrating automated fault detection and diagnosis methods in real buildings.* Technical Research Centre of Finland (VTT), 2001.

[40] H. Wang, Y. Chen, C. W. Chan, and J. Qin, "An online fault diagnosis tool of vav terminals for building management and control systems," *Automation in Construction*, vol. 22, pp. 203–211, 2012.

[41] D. Choinière and M. Corsi, "A bems-assisted commissioning tool to improve the energy performance of hvac systems," 2003.

[42] D. Choinière, "Dabo: a beam assisted on-going commissioning tool," in *National Conference on Building Commissioning*, 2008.

[43] N. S. Castro and H. Vaezi-Nejad, "Cite-ahu, an automated commissioning tool for air-handling units," in *National Conference on Building Commissioning: May*, vol. 4, 2005, p. 6.

[44] P. Godefroid, N. Klarlund, and K. Sen, "Dart: directed automated random testing," in *ACM Sigplan Notices*, vol. 40, no. 6. ACM, 2005, pp. 213–223.

[45] R. Majumdar, I. Saha, and Z. Wang, "Systematic testing for control applications," in *2010 8th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*. IEEE, 2010, pp. 1–10.

[46] M. Padilla, "A review of fault detection, diagnostic and isolation methods for vav-air handing units," *Intelligent buildings sub-program*, 2014.

[47] S. Katipamula and M. R. Brambley, "Review article: methods for fault detection, diagnostics, and prognostics for building systemsâĂŤa review, part ii," *HVAC&R Research*, vol. 11, no. 2, pp. 169–187, 2005.

[48] A. Dexter and J. Pakanen, "Annex 34: Computeraided evaluation of hvac system performance, final report," *International Energy Agency, Oxford*, 2001.

[49] M. Stefik, *Introduction to knowledge systems.* Morgan Kaufmann, 1995.

[50] Iowa Energy Center, "Building Energy Use and Control Problems: Defining the Connection," *National Building Controls Information Program*, 2002.

[51] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking.* MIT press, 1999.

[52] B. C. Rawlings, J. M. Wassick, and B. E. Ydstie, "Error detection for chemical plant automation logic using supervisory control theory," in *Control Applications (CCA), 2015 IEEE Conference on.* IEEE, 2015, pp. 376–381.

[53] C. M. Park, S. Park, and G.-N. Wang, "Control logic verification for an automotive body assembly line using simulation," *International Journal of Production Research*, vol. 47, no. 24, pp. 6835–6853, 2009.

[54] L. Wang, K. C. Tan, X. Jiang, and Y. Chen, "A flexible automatic test system for rotating-turbine machinery," *IEEE transactions on automation science and engineering*, vol. 2, no. 1, pp. 1–18, 2005.

[55] A. Van Lamsweerde, "Goal-oriented requirements engineering: A guided tour," in *Proceedings fifth ieee international symposium on requirements engineering.* IEEE, 2001, pp. 249–262.

[56] A. Lapouchnian, "Goal-oriented requirements engineering: An overview of the current research," *University of Toronto*, p. 32, 2005.

[57] A. Dardenne, A. Van Lamsweerde, and S. Fickas, "Goal-directed requirements acquisition," *Science of computer programming*, vol. 20, no. 1-2, pp. 3–50, 1993.

[58] A. I. Anton, "Goal identification and refinement in the specification of software-based information systems," 1997.

[59] I.-T. S. Sector, "Draft specification of the goaloriented requirement language (z. 151)," 2001.

[60] G. Regev and A. Wegmann, "Where do goals come from: the underlying principles of goal-oriented requirements engineering," in *13th IEEE International Conference on Requirements Engineering (RE'05)*. IEEE, 2005, pp. 353–362.

[61] R. Darimont and A. Van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," in *ACM SIGSOFT Software Engineering Notes*, vol. 21, no. 6. ACM, 1996, pp. 179–190.

[62] S. Tiwari, S. S. Rathore, and A. Gupta, "Selecting requirement elicitation techniques for software projects," in *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*. IEEE, 2012, pp. 1–10.

[63] T. Bureš, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, "The invariant refinement method," in *Software Engineering for Collective Autonomic Systems*. Springer, 2015, pp. 405–428.

[64] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A survey of context modelling and reasoning techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[65] D. Dermeval, J. Vilela, I. I. Bittencourt, J. Castro, S. Isotani, P. Brito, and A. Silva, "Applications of ontologies in requirements engineering: a systematic review of the literature," *Requirements Engineering*, vol. 21, no. 4, pp. 405–437, 2016.

[66] D. V. Dzung and A. Ohnishi, "Ontology-based reasoning in requirements elicitation," in *2009 seventh iEEE international conference on software engineering and formal methods.* IEEE, 2009, pp. 263–272.

[67] H. Kaiya and M. Saeki, "Using domain ontology as domain knowledge for requirements elicitation," in *14th IEEE International Requirements Engineering Conference (RE'06).* IEEE, 2006, pp. 189–198.

[68] M. Shibaoka, H. Kaiya, and M. Saeki, "Goore: Goal-oriented and ontology driven requirements elicitation method," in *International Conference on Conceptual Modeling.* Springer, 2007, pp. 225–234.

[69] P. Delgoshaei and M. A. Austin, "Framework for knowledge-based fault detection and diagnostics in multi-domain systems: Application to heating ventilation and air conditioning systems," *International Journal On Advances in Systems and Measurements*, 2017.

[70] H. Yang, S. Cho, C.-S. Tae, and M. Zaheeruddin, "Sequential rule based algorithms for temperature sensor fault detection in air handling units," *Energy Conversion and Management*, vol. 49, no. 8, pp. 2291–2306, 2008.

[71] J. Shiozaki and F. Miyasaka, "A fault diagnosis tool for hvac systems using qualitative reasoning algorithms," in *Proceedings of the Building Simulation'99, 6th IBPSA conference, Kyoto, Japan*, 1999.

[72] N. Fernandez, M. R. Brambley, and S. Katipamula, "Self-correcting hvac controls: Algorithms for sensors and dampers in air-handling units," Pacific Northwest National Lab.(PNNL), Richland, WA (United States), Tech. Rep., 2009.

[73] M. Brambley, N. Fernandez, W. Wang, K. Cort, H. Cho, H. Ngo, and J. Goddard, "Self-correcting controls for vav system faults: Filter/fan/coil and vav box sections," *Richland, WA: Pacific Northwest National Laboratory*, 2011.

[74] K. Bruton, P. Raftery, P. O'Donovan, N. Aughney, M. M. Keane, and D. O'Sullivan, "Development and alpha testing of a cloud based automated fault detection and diagnosis tool for air handling units," *Automation in Construction*, vol. 39, pp. 70–83, 2014.

[75] H. Wang, Y. Chen, C. W. Chan, and J. Qin, "A robust fault detection and diagnosis strategy for pressure-independent vav terminals of real office buildings," *Energy and Buildings*, vol. 43, no. 7, pp. 1774–1783, 2011.

[76] W.-Y. Lee, J. M. House, and N.-H. Kyong, "Subsystem level fault diagnosis of a building's air-handling unit using general regression neural networks," *Applied Energy*, vol. 77, no. 2, pp. 153–170, 2004.

[77] J. Hyvarnen *et al.*, "Iea annex 25, building optimization and fault diagnosis source book," *Paris: International Energy Agency*, 1995.

[78] S. Li, "A model-based fault detection and diagnostic methodology for secondary hvac systems," 2009.

[79] S. R. West, Y. Guo, X. R. Wang, and J. Wall, "Automated fault detection and diagnosis of hvac subsystems using statistical machine learning," in *12th International Conference of the International Building Performance Simulation Association*, 2011.

[80] R. Sacks, C. Eastman, G. Lee, and P. Teicholz, *BIM handbook: A guide to building information modeling for owners, designers, engineers, contractors, and facility managers.* Wiley, 2018.

[81] E. Dado, R. Beheshti, and M. van de Ruitenbeek, "Product modelling in the building and construction industry: a history and perspectives," in *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies.* IGI Global, 2010, pp. 104–137.

[82] V. Bazjanac and D. Crawley, "Industry foundation classes and interoperable commercial software in support of design of energy-efficient buildings," in *Proceedings of Building Simulation'99*, vol. 2, 1999, pp. 661–667.

[83] B. East, "Hvac information exchange (hvacie)," *National Institute of Building Science*, 2013.

[84] S. Roth, "A building information modeling solution for our green world, gbxml schema (5.12)," 2014.

[85] C. M. Eastman, *Building Product Models: Computer Environments Supporting Design and Construction*, 1st ed. Boca Raton, FL, USA: CRC Press, Inc., 1999.

[86] V. Bazjanac, J. Forester, P. Haves, D. Sucic, and P. Xu, "Hvac component data modeling using industry foundation classes," *Lawrence Berkeley National Laboratory*, 2002.

[87] X. Yang and S. Ergan, "Towards a formal approach for determining functions of hvac components represented in ifc," in *Computing in Civil and Building Engineering (2014)*, 2014, pp. 633–640.

[88] X. Liu, B. Akinci, J. Garrett, and M. Bergés, "Requirements and development of a computerized approach for analyzing functional relationships among hvac components using building information models," *Proceedings of the CIB W78-W102*, 2011.

[89] X. Liu, "An integrated information support framework for performance analysis and improvement of secondary hvac systems," Ph.D. dissertation, Carnegie Mellon University, 2012.

[90] M. T. Turkaslan-Bulbul and O. Akin, "Computational support for building evaluation: Embedded commissioning model," *Automation in construction*, vol. 15, no. 4, pp. 438–447, 2006.

[91] Y. Chen, S. J. Treado, and J. I. Messner, "Building hvac control knowledge data schema–towards a unified representation of control system knowledge," *Automation in Construction*, vol. 72, pp. 174–186, 2016.

[92] G. F. Schneider, P. Pauwels, and S. Steiger, "Ontology-based modeling of control logic in building automation systems," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3350–3360, 2017.

[93] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal *et al.*, "Brick: Towards a unified metadata schema for buildings," in *Proceedings of the ACM International Conference on Embedded Systems for Energy-Efficient Built Environments (BuildSys). ACM*, 2016.

[94] Haystack, "Project haystack," http://project-haystack.org/, 2014.

[95] B. Dong and K. P. Lam, "A real-time model predictive control for building heating and cooling systems based on the occupancy behavior pattern detection and local weather forecasting," *Building Simulation*, vol. 7, no. 1, pp. 89–106, 2014. [Online]. Available: http://link.springer.com/10.1007/s12273-013-0142-7

[96] S. Wang and Z. Ma, "Supervisory and Optimal Control of Building HVAC Systems: A Review," *HVAC&R Research*, vol. 14, no. 1, pp. 3–32, 2008.

[97] B. Dong, "Integrated Building Heating, Cooling and Ventilation Control," *Thesis*, pp. 1–174, 2010. [Online]. Available: http://repository.cmu.edu/dissertations/4/

[98] A. I. Dounis and C. Caraiscos, "Advanced control systems engineering for energy and comfort management in a building environment - a review," *Renewable and Sustainable Energy Reviews*, vol. 13, no. 6, pp. 1246–1261, 2009.

[99] ASHRAE, *ASHRAE Handbook - Fundamentals (I-P Edition)*, 2013.

[100] D. A. Wiegmann and S. A. Shappell, *A human error approach to aviation accident analysis: The human factors analysis and classification system.* Routledge, 2017.

[101] K. Fearon, F. Strasser, S. D. Anker, I. Bosaeus, E. Bruera, R. L. Fainsinger, A. Jatoi, C. Loprinzi, N. MacDonald, G. Mantovani *et al.*, "Definition and classification of cancer cachexia: an international consensus," *The lancet oncology*, vol. 12, no. 5, pp. 489–495, 2011.

[102] S. Kamley, S. Jaloree, and R. Thakur, "Performance comparison between forward and backward chaining rule based expert system approaches over global

stock exchanges," *International Journal of Computer Science and Information Security*, vol. 14, no. 3, p. 74, 2016.

[103] A. Al-Ajlan, "The comparison between forward and backward chaining," *International Journal of Machine Learning and Computing*, vol. 5, no. 2, p. 106, 2015.

[104] R. H. Bruning, G. J. Schraw, and R. R. Ronning, *Cognitive psychology and instruction.* ERIC, 1999.

[105] J. L. Dietz, *Enterprise ontology: theory and methodology.* Springer Science & Business Media, 2006.

[106] D. J. Armstrong, "The quarks of object-oriented development," *Communications of the ACM*, vol. 49, no. 2, pp. 123–128, 2006.

[107] S. Finger and J. R. Dixon, "A review of research in mechanical engineering design. Part II: Representations, analysis, and design for the life cycle," *Research in Engineering Design*, vol. 1, no. 2, pp. 121–137, 1989.

[108] A. Dardenne, S. Fickas, and A. van Lamsweerde, "Goal-directed concept acquisition in requirements elicitation," in *Proceedings of the 6th international workshop on Software specification and design.* IEEE Computer Society Press, 1991, pp. 14–21.

[109] I. Sommerville and P. Sawyer, *Requirements engineering: a good practice guide.* John Wiley & Sons, Inc., 1997.

[110] A. Van Lamsweerde, R. Darimont, and E. Letier, "Managing conflicts in goal-driven requirements engineering," *IEEE transactions on Software engineering*, vol. 24, no. 11, pp. 908–926, 1998.

[111] N. R. Mead, "Requirements prioritization introduction (https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=299135)," *Software Eng. Inst. web pub., Carnegie Mellon Univ*, 2013. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=299135

[112] A. Salado and R. Nilchiani, "The concept of order of conflict in requirements engineering," *IEEE Systems Journal*, vol. 10, no. 1, pp. 25–35, 2016.

[113] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE software*, vol. 14, no. 5, pp. 67–74, 1997.

[114] E. Domínguez, J. Lloret, B. Pérez, Á. Rodríguez, Á. L. Rubio, and M. A. Zapata, "A survey of uml models to xml schemas transformations," in *International Conference on Web Information Systems Engineering*. Springer, 2007, pp. 184–195.

[115] T. Krumbein and T. Kudrass, "Rule-based generation of xml schemas from uml class diagrams." in *Berliner XML Tage*, vol. 2003, 2003, pp. 213–227.

[116] Liquid Technologies, "Liquid studio," 2017. [Online]. Available: https://www.liquid-technologies.com/xml-studio

[117] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini, and A. Bemporad, "Model predictive control (mpc) for enhancing building and hvac system energy efficiency: Problem formulation, applications and opportunities," *Energies*, vol. 11, no. 3, p. 631, 2018.

[118] Y. Ma, F. Borrelli, B. Hencey, B. Coffey, S. Bengea, and P. Haves, "Model predictive control for the operation of building cooling systems," *IEEE Transactions on control systems technology*, vol. 20, no. 3, pp. 796–803, 2012.

[119] J. Rehrl and M. Horn, “Temperature control for hvac systems based on exact linearization and model predictive control,” in *2011 IEEE International Conference on Control Applications (CCA).* IEEE, 2011, pp. 1119–1124.

[120] P.-D. Moroşan, R. Bourdais, D. Dumur, and J. Buisson, “Building temperature regulation using a distributed model predictive control,” *Energy and Buildings*, vol. 42, no. 9, pp. 1445–1452, 2010.

[121] M. Gouda, S. Danaher, and C. Underwood, “Thermal comfort based fuzzy logic controller,” *Building services engineering research and technology*, vol. 22, no. 4, pp. 237–253, 2001.

[122] R. Z. Homod, K. S. M. Sahari, H. A. Almurib, and F. H. Nagi, “Gradient autotuned takagi–sugeno fuzzy forward control of a hvac system using predicted mean vote index,” *Energy and Buildings*, vol. 49, pp. 254–267, 2012.

[123] A. Shepherd and W. Batty, “Fuzzy control strategies to provide cost and energy efficient high quality indoor environments in buildings with high occupant densities,” *Building Services Engineering Research and Technology*, vol. 24, no. 1, pp. 35–45, 2003.

[124] S. Soyguder, M. Karakose, and H. Alli, “Design and simulation of self-tuning pid-type fuzzy adaptive control for an expert hvac system,” *Expert systems with applications*, vol. 36, no. 3, pp. 4566–4573, 2009.

[125] J. Liang and R. Du, “Thermal comfort control based on neural network for hvac application,” in *Proceedings of 2005 IEEE Conference on Control Applications, 2005. CCA 2005.* IEEE, 2005, pp. 819–824.

[126] A. E. Ben-Nakhi and M. A. Mahmoud, "Energy conservation in buildings through efficient a/c control using neural networks," *Applied Energy*, vol. 73, no. 1, pp. 5–23, 2002.

[127] G. P. Henze and R. E. Hindman, "Control of air-cooled chiller condenser fans using clustering neural networks," *TRANSACTIONS-AMERICAN SOCIETY OF HEATING REFRIGERATING AND AIR CONDITIONING ENGINEERS*, vol. 108, no. 2, pp. 232–244, 2002.

[128] S. Katipamula and M. R. Brambley, "Methods for fault detection, diagnostics, and prognostics for building systemsâĂŤa review, part ii," *Hvac&R Research*, vol. 11, no. 2, pp. 169–187, 2005.

[129] S. Moon, Y. Kim, M. Kim, and S. Yoo, "Ask the mutants: Mutating faulty programs for fault localization," in *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on.* IEEE, 2014, pp. 153–162.

[130] M. Papadakis and Y. Le Traon, "Metallaxis-fl: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605–628, 2015.

[131] M. M. Ardehali and T. F. Smith, "Literature review to identify existing case studies of controls-related energy-inefficiency in buildings," *Prepared for the National Building Controls Information Program*, 2001.

[132] H. Wang, Y. Chen, C. W. Chan, J. Qin, and J. Wang, "Online model-based fault detection and diagnosis strategy for vav air handling units," *Energy and Buildings*, vol. 55, pp. 252–263, 2012.

[133] N. Fernandez, H. Cho, M. R. Brambley, J. Goddard, S. Katipamula, and L. Dinh, "Self-correcting hvac controls," *Project Final Report PNNL-19074, Pacific Northwest National Laboratory, Richland, WA*, 2009.

[134] M. R. Brambley, N. Fernandez, W. Wang, K. A. Cort, H. Cho, H. Ngo, and J. K. Goddard, "Final project report: Self-correcting controls for vav system faults filter/fan/coil and vav box sections," Pacific Northwest National Lab.(PNNL), Richland, WA (United States), Tech. Rep., 2011.

[135] Z. Li, C. J. Paredis, G. Augenbroe, and G. Huang, "A rule augmented statistical method for air-conditioning system fault detection and diagnostics," *Energy and Buildings*, vol. 54, pp. 154–159, 2012.

[136] S. Wang and Z. Jiang, "Valve fault detection and diagnosis based on cmac neural networks," *Energy and Buildings*, vol. 36, no. 6, pp. 599–610, 2004.

[137] S. Wang and F. Xiao, "Ahu sensor fault diagnosis using principal component analysis method," *Energy and Buildings*, vol. 36, no. 2, pp. 147–160, 2004.

[138] ——, "Detection and diagnosis of ahu sensor faults using principal component analysis method," *Energy Conversion and Management*, vol. 45, no. 17, pp. 2667–2686, 2004.

[139] J. Qin and S. Wang, "A fault detection and diagnosis strategy of vav air-conditioning systems for improved energy and control performances," *Energy and Buildings*, vol. 37, no. 10, pp. 1035–1048, 2005.

[140] S. Wang and J. Qin, "Sensor fault detection and validation of vav terminals in air conditioning systems," *Energy Conversion and Management*, vol. 46, no. 15-16, pp. 2482–2500, 2005.

[141] Z. Hou, Z. Lian, Y. Yao, and X. Yuan, “Data mining based sensor fault diagnosis and validation for building air conditioning system,” *Energy Conversion and Management*, vol. 47, no. 15-16, pp. 2479–2490, 2006.

[142] X. Jin and Z. Du, “Fault tolerant control of outdoor air and ahu supply air temperature in vav air conditioning systems using pca method,” *Applied Thermal Engineering*, vol. 26, no. 11-12, pp. 1226–1237, 2006.

[143] F. Xiao, S. Wang, and J. Zhang, “A diagnostic tool for online sensor health monitoring in air-conditioning systems,” *Automation in Construction*, vol. 15, no. 4, pp. 489–503, 2006.

[144] Z. Du, X. Jin, and L. Wu, “Fault detection and diagnosis based on improved pca with jaa method in vav systems,” *Building and Environment*, vol. 42, no. 9, pp. 3221–3232, 2007.

[145] ——, “Pca-fda-based fault diagnosis for sensors in vav systems,” *Hvac&R Research*, vol. 13, no. 2, pp. 349–367, 2007.

[146] Z. Du and X. Jin, “Detection and diagnosis for multiple faults in vav systems,” *Energy and Buildings*, vol. 39, no. 8, pp. 923–934, 2007.

[147] ——, “Detection and diagnosis for sensor fault in hvac systems,” *Energy Conversion and Management*, vol. 48, no. 3, pp. 693–702, 2007.

[148] ——, “Multiple faults diagnosis for sensors in air handling unit using fisher discriminant analysis,” *Energy Conversion and Management*, vol. 49, no. 12, pp. 3654–3665, 2008.

[149] Z. Du, X. Jin, and Y. Yang, “Fault diagnosis for temperature, flow rate and pressure sensors in vav systems using wavelet neural network,” *Applied energy*, vol. 86, no. 9, pp. 1624–1631, 2009.

[150] Z. Du, X. Jin, and X. Yang, “A robot fault diagnostic tool for flow rate sensors in air dampers and vav terminals,” *Energy and Buildings*, vol. 41, no. 3, pp. 279–286, 2009.

[151] F. Xiao, S. Wang, X. Xu, and G. Ge, “An isolation enhanced pca method with expert-based multivariate decoupling for sensor fdd in air-conditioning systems,” *Applied Thermal Engineering*, vol. 29, no. 4, pp. 712–722, 2009.

[152] B. Fan, Z. Du, X. Jin, X. Yang, and Y. Guo, “A hybrid fdd strategy for local system of ahu based on artificial neural network and wavelet analysis,” *Building and Environment*, vol. 45, no. 12, pp. 2698–2708, 2010.

[153] Y. Chen and L. Lan, “Fault detection, diagnosis and data recovery for a real building heating/cooling billing system,” *Energy Conversion and Management*, vol. 51, no. 5, pp. 1015–1024, 2010.

[154] X.-B. Yang, X.-Q. Jin, Z.-M. Du, Y.-H. Zhu, and Y.-B. Guo, “A hybrid model-based fault detection strategy for air handling unit sensors,” *Energy and buildings*, vol. 57, pp. 132–143, 2013.

[155] X.-B. Yang, X.-Q. Jin, Z.-M. Du, and Y.-H. Zhu, “A novel model-based fault detection method for temperature sensor using fractal correlation dimension,” *Building and Environment*, vol. 46, no. 4, pp. 970–979, 2011.

[156] S. Li and J. Wen, "A model-based fault detection and diagnostic methodology based on pca method and wavelet transform," *Energy and Buildings*, vol. 68, pp. 63–71, 2014.

[157] ——, "Application of pattern matching method for detecting faults in air handling unit system," *Automation in Construction*, vol. 43, pp. 49–58, 2014.

[158] K. Duost and L. Rosenthal, "Test assertion guide," W3C, Tech. Rep., Apr. 2006, https://www.w3.org/2006/03/test-assertion-guide/.

[159] M. A. Alipour, "Automated fault localization techniques: a survey," *Oregon State University*, 2012.

[160] P. Parmar and M. Patel, "Software fault localization: A survey," *International Journal of Computer Applications*, vol. 154, pp. 6–13, 11 2016.

[161] C. Gouveia, J. Campos, and R. Abreu, "Using html5 visualizations in software fault localization," in *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. IEEE, 2013, pp. 1–10.

[162] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 international symposium on software testing and analysis*. ACM, 2011, pp. 199–209.

[163] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016, pp. 165–176.

[164] X. Xia, L. Bao, D. Lo, and S. Li, "" automated debugging considered harmful" considered harmful: A user study revisiting the usefulness of spectra-based

fault localization techniques with professionals using real bugs from large systems," 2016.

[165] H. A. d. Souza, "Assessment of spectrum-based fault localization for practical use," Ph.D. dissertation, Universidade de São Paulo, 2018.

[166] E. O. Soremekun, M. Böhme, and A. Zeller, "Programmers should still use slices when debugging," 2016.

[167] S. Ali, J. H. Andrews, T. Dhandapani, and W. Wang, "Evaluating the accuracy of fault localization techniques," in *Proceedings of the 2009 IEEE/ACM international conference on automated software engineering.* IEEE Computer Society, 2009, pp. 76–87.

[168] R. Just, "The Major mutation framework: Efficient and scalable mutation analysis for Java," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, San Jose, CA, USA, July 23–25 2014, pp. 433–436.

[169] V. K. Palepu. (2016) Tacoco: Integrated software analysis framework. [Online]. Available: https://github.com/spideruci/tacoco

[170] K. Beck and E. Gamma, "Test infected: Programmers love writing tests," *Java Report*, vol. 3, no. 7, pp. 37–50, 1998.

[171] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis.* ACM, 2014, pp. 437–440.

[172] Y. Chen, “Building control knowledge information modeling and control self-configuration,” Ph.D. dissertation, 2015, copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2016-06-07. [Online]. Available: https://search.proquest.com/docview/1734867668?accountid=9902

[173] R. C. Dorf and R. H. Bishop, *Modern control systems.* Pearson, 2011.

[174] A. Endres, “An analysis of errors and their causes in system programs,” *IEEE Transactions on Software Engineering*, no. 2, pp. 140–149, 1975.

[175] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, and M.-Y. Wong, “Orthogonal defect classification-a concept for in-process measurements,” *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[176] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE transactions on dependable and secure computing*, vol. 1, no. 1, pp. 11–33, 2004.

[177] T. Hall, D. Bowes, S. Counsell, L. Moonen, and A. Yamashita, “Software fault characteristics: A synthesis of the literature,” 2015.

[178] J. Ploski, M. Rohr, P. Schwenkenberg, and W. Hasselbring, “Research issues in software fault categorization,” *ACM SIGSOFT Software Engineering Notes*, vol. 32, no. 6, p. 6, 2007.

[179] K. Pan, S. Kim, and E. J. Whitehead, “Toward an understanding of bug fix patterns,” *Empirical Software Engineering*, vol. 14, no. 3, pp. 286–315, 2009.

[180] Y. Zhao, H. Leung, Y. Yang, Y. Zhou, and B. Xu, "Towards an understanding of change types in bug fixing code," *Information and software technology*, vol. 86, pp. 37–53, 2017.

[181] E. N. Narciso, M. E. Delamaro, and F. D. L. D. S. Nunes, "Test case selection: A systematic literature review," *International Journal of Software Engineering and Knowledge Engineering*, vol. 24, no. 04, pp. 653–676, 2014.

[182] H. Hemmati, A. Arcuri, and L. Briand, "Reducing the cost of model-based testing through test case diversity," in *IFIP International Conference on Testing Software and Systems*. Springer, 2010, pp. 63–78.

[183] P. Sapna and H. Mohanty, "Clustering test cases to achieve effective test selection," in *Proceedings of the 1st Amrita ACM-W Celebration on Women in Computing in India*. ACM, 2010, p. 15.

[184] L. Rokach and O. Maimon, *Clustering Methods*. Boston, MA: Springer US, 2005, pp. 321–352. [Online]. Available: https://doi.org/10.1007/0-387-25465-X{_}15

[185] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[186] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[187] M. D. Ernst, “Static and dynamic analysis: Synergy and duality,” in *WODA 2003: ICSE Workshop on Dynamic Analysis*. New Mexico State University Portland, OR, 2003, pp. 24–27.

[188] “American auto-matrix,” http://www.aamatrix.com/.