No Need for Excuses: Applying Software Engineering Principles to Facilitate Scientific Software Documentation

Spencer Smith, Malavika Srinivasan and Sumanth Shankar[*]

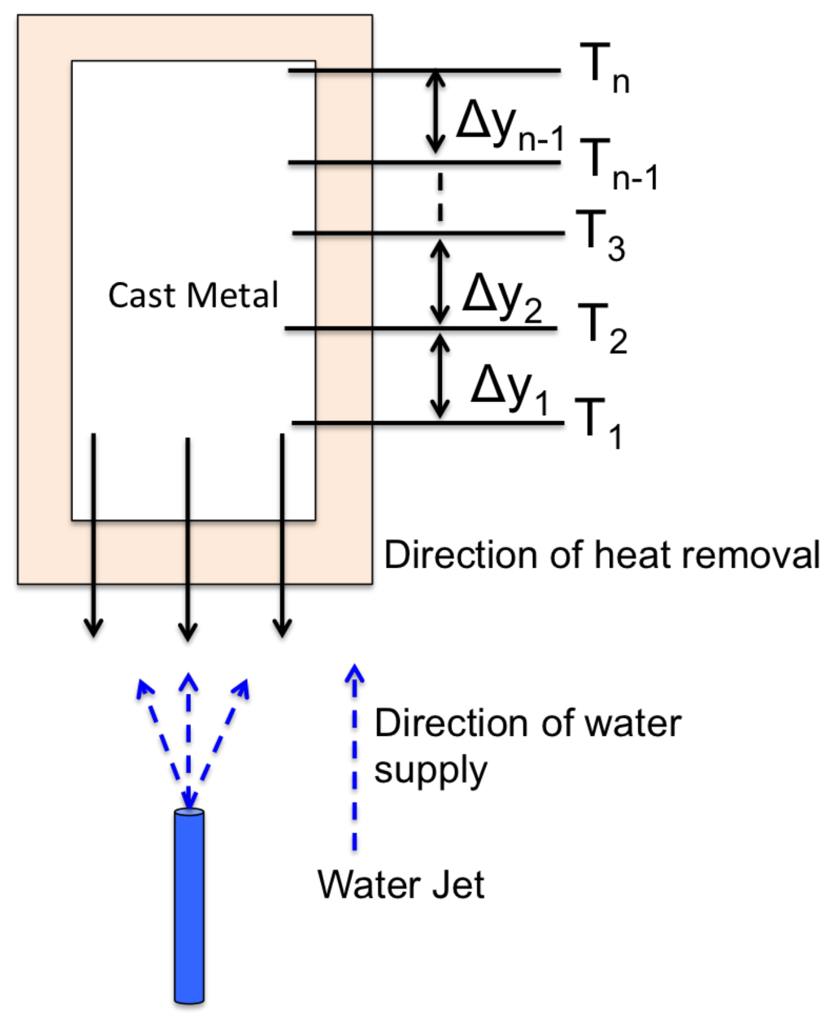
Computing and Software Department, McMaster University

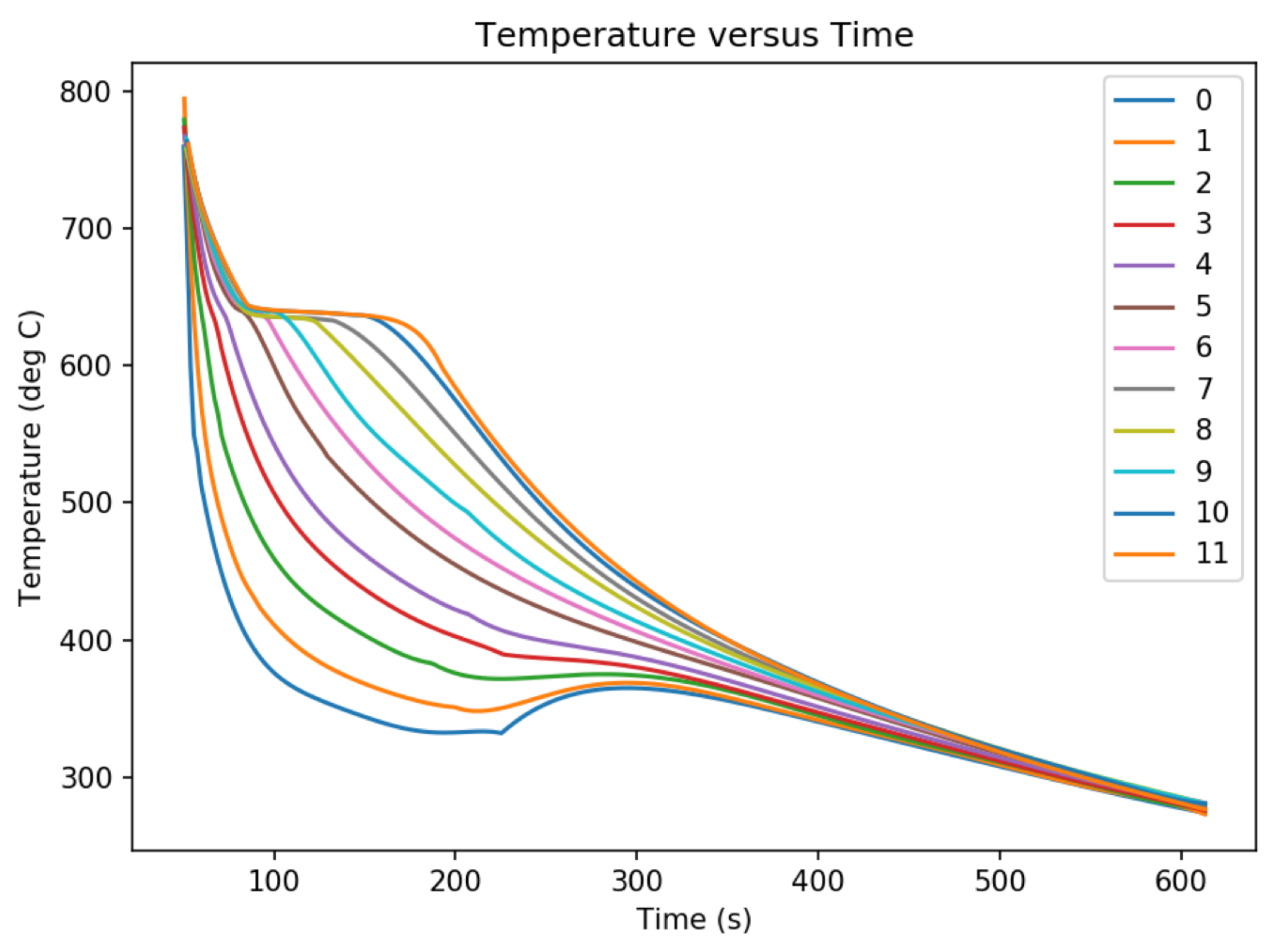
[*]Mechanical Engineering Department, McMaster University

smiths@mcmaster.ca

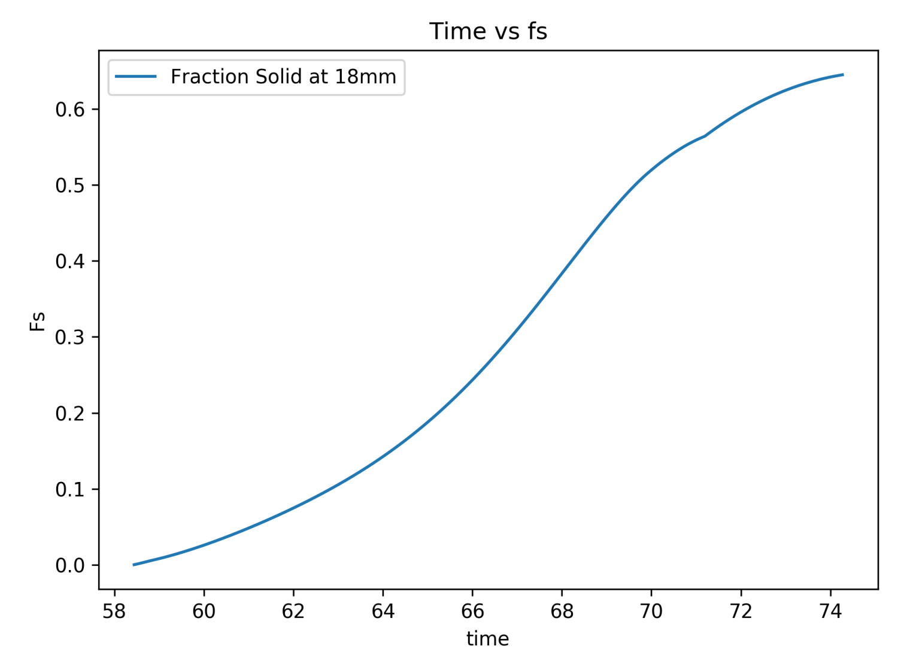
SOFTWARE documentation improves qualities such as maintainability, reusability, verifiability and usability. So why do scientific software developers underemphasize documentation? Reasons include: i) requirements are not known up-front (they emerge over time), ii) change is frequent, iii) rigid processes hamper creativity, iv) the software is too complex; and, v) there is no test oracle. These are reasons that documentation is challenging, *not excuses for avoiding it entirely*. Complexity and frequent change are not unique to scientific software. So how do other domains deal with these challenges – by applying Software Engineering (SE) principles, techniques and tools. Specific SE ideas helpful to scientific software include: faking a rational design process, documentation templates, abstraction, anticipation of change, generality, replicability, separation of concerns, information hiding, and tool support. Many developers are familiar with these ideas, but examples of adapting them to scientific software are rare. An example is shown here to illustrate SE ideas applied to software to analyze the solidification of a molten metal alloy.

1. Software For Solidification (SFS)



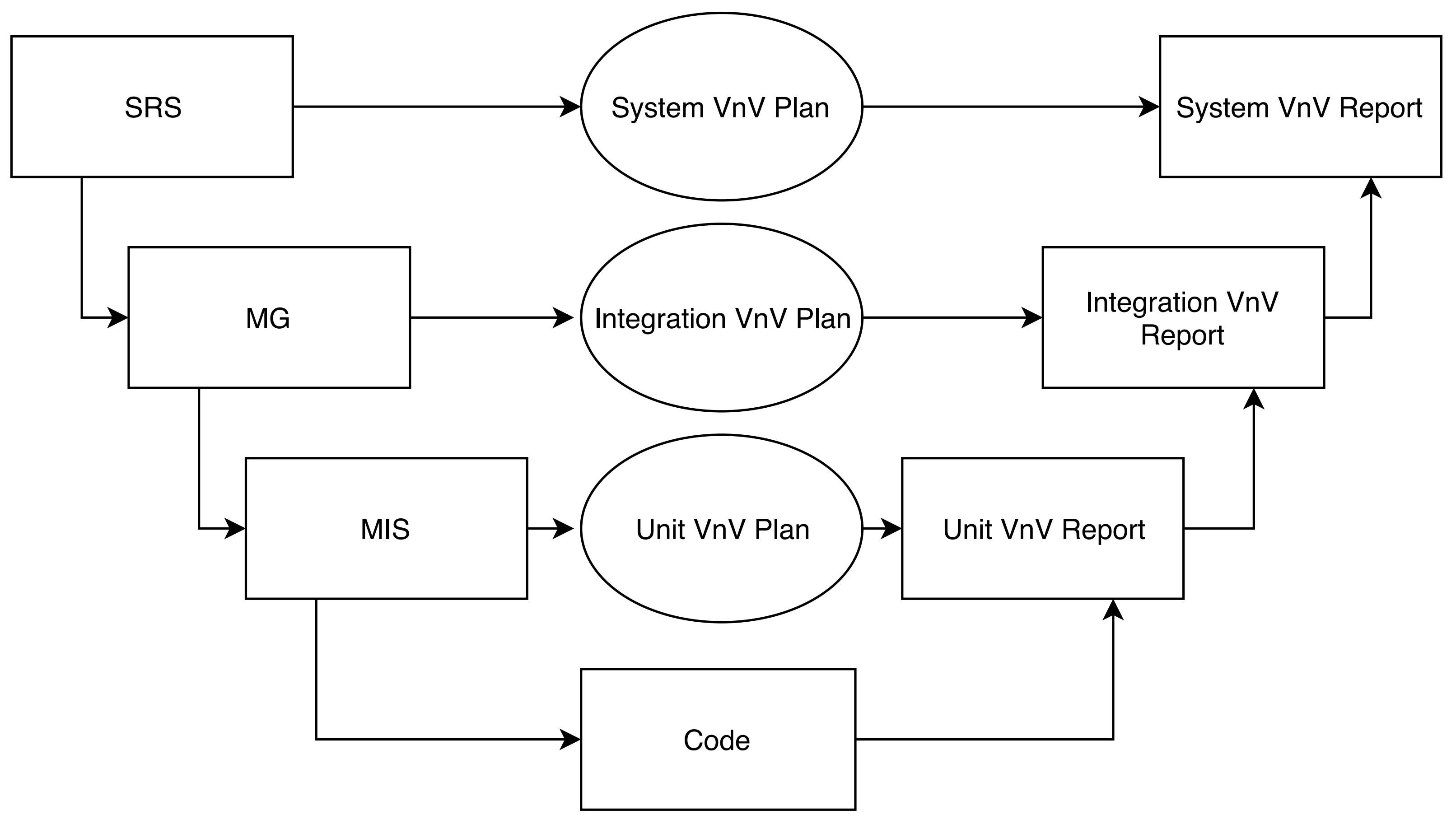


The output of SFS is the fraction solid (f_S), either as a function of time, temperature or cooling rate. f_S is obtained by solving an Ordinary Differential Equation (ODE) based on the experimental temperature data (shown above), material properties and processing conditions [6].



2. Faking a Rational Design Process

A rational design process, with up-front requirements flowing into successive design stages, is not usually feasible, but the best way to write documentation is still to "fake" a rational process [2]. SFS was developed using a V-model, where a testing phase is associated with each step in the typical waterfall process. The documents include a Software Requirements Specification (SRS), a software architecture in a Module Guide (MG), and a detailed design in a Module Interface Specification (MIS) [3].



3. Documentation Templates

A template tailored to scientific software [5] was adopted for the SRS. The template was designed to use SE principles, like abstraction, separation of concerns, anticipation of change and generalization.

1. Reference Material: a) Table of Units b) Table of Symbols c) Abbreviations and Acronyms
2. Introduction: a) Purpose of Document b) Scope of Requirements c) Intended Audience d) Organization of Document
3. Background
4. General System Description: a) System Context b) User Characteristics c) System Constraints
5. Specific System Description:
   a) Problem Description: i) Terminology and Definitions ii) Physical System Description iii) Goal Statements
   b) Solution Characteristics Specification: i) Assumptions ii) Theoretical Models iii) General Definitions iv) Data Definitions v) Instance Models vi) Data Constraints vii) Properties of a Correct Solution
6. Requirements:
   a) Functional Requirements: i) Configuration Mode ii) Calibration Mode iii) Calculation Mode
   b) Non-Functional Requirements: i) Look and Feel Requirements ii) Usability and Humanity Requirements iii) Installability Requirements iv) Performance Requirements v) Operating and Environmental Requirements vi) Maintainability and Support Requirements vii) Security Requirements viii) Cultural Requirements ix) Compliance Requirements
7. Likely Changes
8. Unlikely Changes
9. Supporting Information

Templates designed using SE principles were also used for the Module Guide, Module Interface Specification, and the Verification and Validation plans and reports. The Python code mostly followed the PEP8 standard. Doxygen was used to generate documentation for the Application Program Interface (API).

4. Abstraction

The SRS for SFS starts with an abstract overall goal, which is later refined via general definitions to an instanced model, which is itself refined into a design and subsequently code.

For a given experiment with a metal alloy, using the thermocouple locations, temperature readings, material properties and initial conditions, SFS:

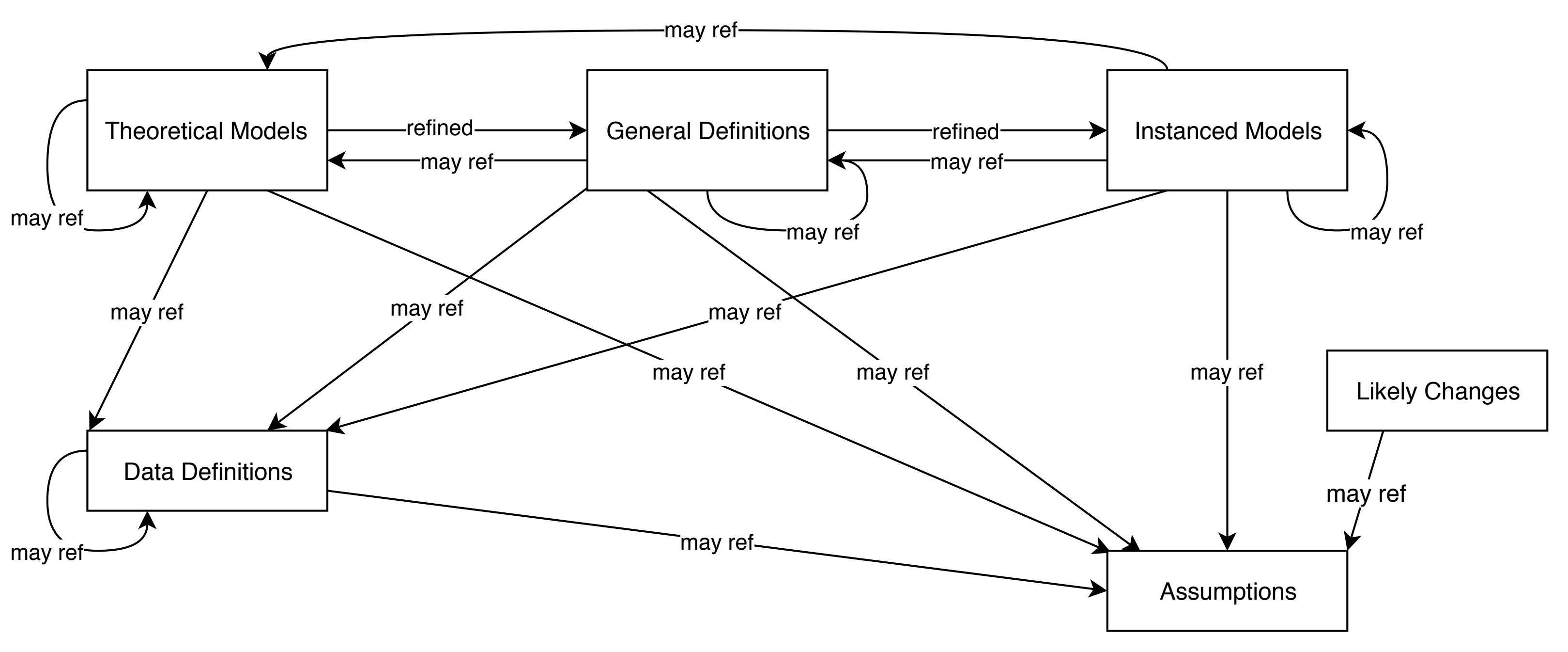
GS1: Computes the solid fraction (f_s) as a function of temperature (T) and cooling rate ($f_s(T, \frac{dT}{dt})$).

The requirements for the SRS for SFS were stable throughout this development project because changeable design decisions were postponed via abstract requirements. For instance, the SRS does not specify whether the fit will use regression, interpolation, polynomials or some other basis functions.

Number	GD4
Equation	$T(y,t) = \text{fit}(Tdata, y_{TC}, dt)$ where $\text{fit} : \mathbb{R}^{m\times n} \to \mathbb{R}^n \to \mathbb{R} \to (\mathbb{R} \to \mathbb{R} \to \mathbb{R})$
Descript.	Experimental temperature history data ($Tdata$) at the thermocouples is used to determine the function $T(y,t)$, where the inputs are y (m), the position as measured from the bottom of the cylinder, and t (second), the time. The output is the temperature (°C). $\text{fit}()$ is a function that takes the thermocouple data $Tdata$, the locations of the thermocouples y_{TC} and the time step dt, and returns the appropriate function $T(y,t)$. m is the number of instants of time where the thermocouple data is measured and n is the number of thermocouples.

5. Anticipation of Change

The SRS is designed with change in mind. The figure below shows the relationship between different parts of the requirements model. Cross-references are bi-directional. The traceability information is also summarized in a traceability matrix. A summary of likely changes is included in the SRS. Most likely changes are related to assumptions that may be modified in the future. Anticipation of change allows for creation of a family of related physical models.



6. Generality

The instance model for solving f_S is written in the general ODE form, to take advantage of existing ODE solvers. Other research on solidification creates an unnecessary challenge by neglecting generality and instead developing its own ad hoc algorithms.

Number	IM4
Input	$T(y,t)$ (see DD4), from which $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$ can be derived, as required Material properties $C_v^L(T)$, $C_v^S(T)$, $\rho_L(T)$, $\rho_S(T)$, α_b (from IM2), α_e (from IM3), and L y^*, (t_L, T_L) from DD6 and (t_S, T_S) from DD7
Output	Solve $f_s(t)$ at location y^* such that the following ODE is satisfied with $f_s(t_L) = 0$: $\dot{f}_s(f_s, t) = \frac{C_v(f_s)}{L\rho(f_s)}\left[\frac{\partial T(t)}{\partial t} - \alpha(f_s)\frac{\partial^2 T(t)}{\partial y^2}\right]$ where ... (derivation invokes simplifying assumptions)

7. Replicability

Although reproducibility is the cornerstone of the scientific method, until recently it has not been treated seriously in software [1]. To replicate the work of another, starting from the theory, requires full documentation of all relevant assumptions. Future change is anticipated via traceability to theories (T), unlikely changes (UC), likely changes (LC), general definitions (GD), data definitions (DD) and instance models (IM).

A1: The only form of energy that is relevant for this problem is thermal energy. All other forms of energy, such as mechanical energy, are assumed to be negligible [T1, UC1].

A2: The heat removal is assumed to be unidirectional and the heat conduction in axial direction is assumed to be 0 [GD1, UC2].

A3: Heat transfer through the cylinder takes place by conduction only, not advection. [DD5].

A4: We assume that $C_v(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD1, GD2].

A5: We assume that $\alpha(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD2, GD2].

A6: We assume that $\rho(T)$ can be expressed as a linear combination of the values at the beginning and at the end of solidification [IM4, DD3, GD2].

A7: Thermal conductivity through the liquid and solid metal is isotropic.

A8: Newton's law of convective cooling applies between the water and the cast alloy [GD3, DD8].

A9: The heat transfer coefficient at the bottom of the cylinder is assumed to be independent of temperature [GD3, LC1].

A10: The thermal resistance due to the thermocouples is assumed to be negligible [GD4].

A11: The cast metal is perfectly insulated by the sand mold so that there is no heat loss from the sand mold [GD1].

A12: The density of the solidifying material is assumed to be constant for the derivation of $\dot{f}_s$ [IM4].

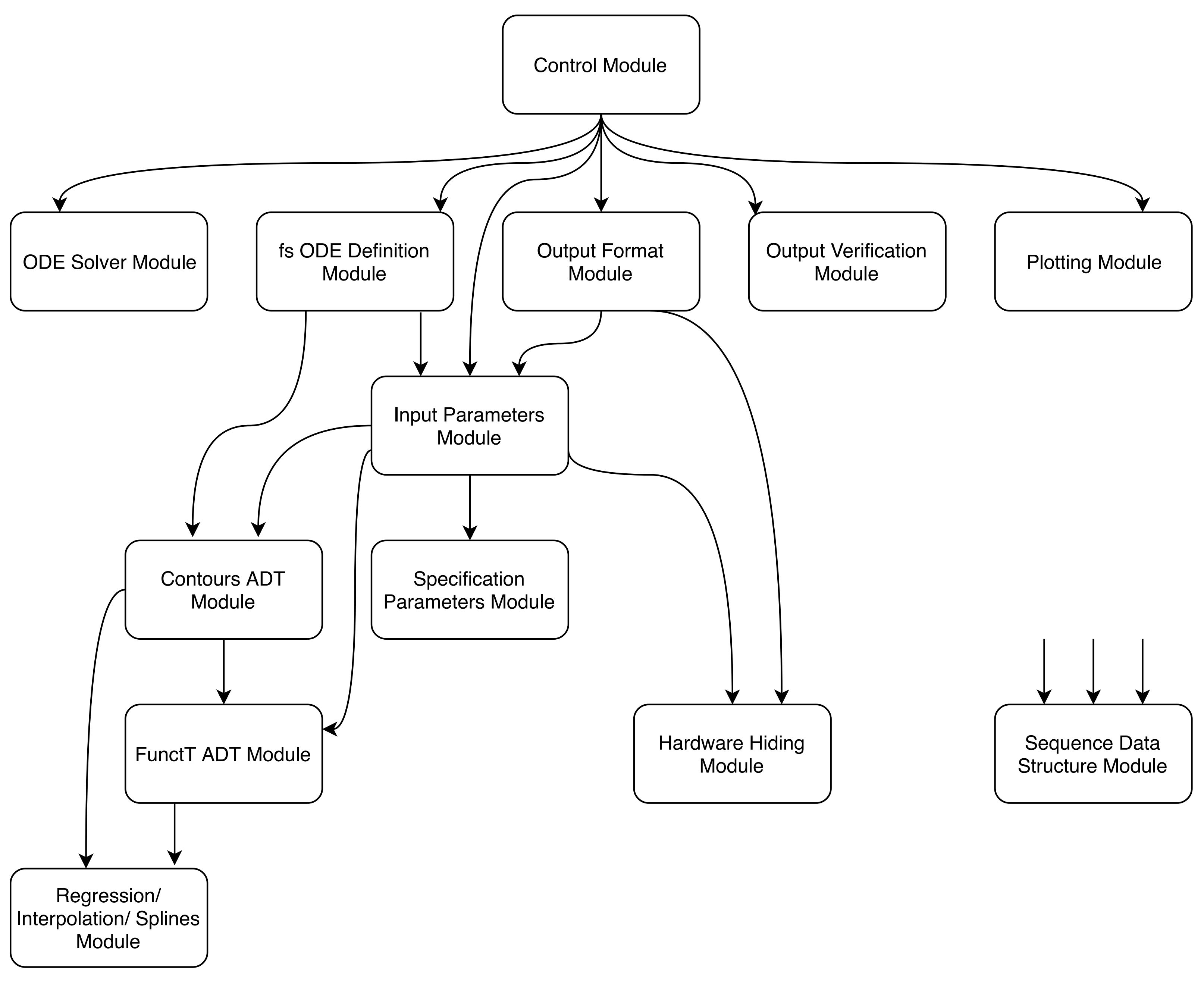
A13: The data collected from the thermocouples starts at time 0 and there is a constant time step between each data point [GD4, LC2].

8. Separation of Concerns

Properties of a correct solution can be brainstormed before a single line of code is written. The following property states that for any fixed time the temperature will be non-decreasing through the height of the cylinder.

Number	PC1
Equation	For any t, $t > 0$, $(\forall y_1, y_2 : \mathbb{R} \mid 0 \le y_1 \le H \wedge 0 \le y_2 \le H \wedge y_1 < y_2 : T_t(y_2) \ge T_t(y_1))$ where $T_t = \lambda y : T(y,t)$
Descript.	$T_t(y)$ is non-decreasing with increasing y (the temperature rises (or remains the same) with increasing height in the cylinder).

The design is divided into modules, as shown in the following uses relationship.



9. Information Hiding

The interface provides a constructor (new FunctT) that takes two vectors of data and returns a FunctT object. Accessors include minD and maxD for finding the extreme limits of the independent data variable. The accessor eval returns the value of the dependent variable (x), given a value for the independent variable (y). The syntax of FuncT shows exceptions if the data for the independent variable is not in ascending order, of if the number of data points in the two sequences do not match, or if a function evaluation is sought outside of the domain of the given data.

Routine	**In**	**Out**	**Exceptions**
new FunctT	X_{in}:$\mathbb{R}^n$, Y_{in}: $\mathbb{R}^n$	FunctT	IndepVarNotAscend, SeqSizeMismatch
minD		$\mathbb{R}$	
maxD		$\mathbb{R}$	
eval	$x : \mathbb{R}$	$\mathbb{R}$	OutOfDomain

The corresponding semantics for FunctT are given below.

State Variables
f: $\mathbb{R} \rightarrow \mathbb{R}$
minX: $\mathbb{R}$
maxX: $\mathbb{R}$

new FunctT($X_{\text{in}}, Y_{\text{in}}$):

- transition: minX, maxX, $f := X_0, X_{|X|-1}, (\lambda v : \text{interp}(X, Y, v))$
- output: $out :=$ self
- exception: $exc := (\neg\text{isAscending}(X_{\text{in}}) \Rightarrow \text{IndepVarNotAscend} \mid |X_{\text{in}}| \neq |Y_{\text{in}}| \Rightarrow \text{SeqSizeMismatch})$

minD():

- output: $out :=$ minX
- exception: None

...
eval(x):

- output: $out := f(x)$
- exception: $exc := (\neg(\text{minX} \leq x \leq \text{maxX}) \Rightarrow \text{OutOfDomain})$

Local Functions
interp: $\mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$
interp(X, Y, v)

$$\equiv \text{interpQuad}(X_{i-1}, Y_{i-1}, X_i, Y_i, X_{i+1}, Y_{i+1}, v) \text{ where } i = \text{index}(X, v)$$

interpQuad: $\mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
interpQuad(x_0, y_0, x_1, y_1, x_2, y_2, x)

$$\equiv y_1 + \frac{y_2 - y_0}{x_2 - x_0}(x - x_1) + \frac{y_2 - 2y_1 + y_0}{2(x_2 - x_1)^2}(x - x_1)^2$$

index: $\mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{N}$ *# constructor test ensures sequence is ascending*
index(X, x) $\equiv i$ such that $X_i \leq x < X_{i+1}$

Below we show the syntax and semantics for the MIS for ContourADT. The full data set is built up by adding each successive thermocouple's data. The accessors, eval, dydx and d2ydx2 are used to calculate the values of $T(y, t)$, $\frac{\partial T}{\partial t}$ and $\frac{\partial^2 T}{\partial y^2}$, respectively. The access program slice returns a new FuncT that would hold the temperature values through the height of the cylinder for a given value of the dependent variable t. The interfaces for FunctT and ContoursT are stable with respect to changes in the fitting algorithm or basis functions.

Syntax: *Exported Access Programs*

Routine	In	Out	Exceptions
ContoursT			
add	s: FunctT, $z : \mathbb{R}$		IndepVarNotAscend
getC	$i : \mathbb{N}$		InvalidIndex
eval	$x : \mathbb{R}, z : \mathbb{R}$		OutOfDomain
dydx	$x : \mathbb{R}, z : \mathbb{R}$		OutOfDomain
d2ydx2	$x : \mathbb{R}, z : \mathbb{R}$		OutOfDomain
slice	$x : \mathbb{R}$	FunctT	

Semantics: *State Variables*

S: sequence of FunctT
Z: sequence of $\mathbb{R}$

new ContoursT(i):

- transition: $S, Z := <>, <>$
- exception: none

add(s, z):

- transition: $S, Z := S || < s >, Z || < z >$
- exception: $exc := (|Z| > 0 \wedge z < Z_{|Z|-1} \Rightarrow \text{IndepVarNotAscend})$

getC(i):

- output: $out := S[i]$
- exception: $exc := (\neg(0 \leq i < |S|) \Rightarrow \text{InvalidIndex})$

eval(x, z):

- output: $out :=$ self.slice(x).eval(z)
- exception: none

dydx(x, z):

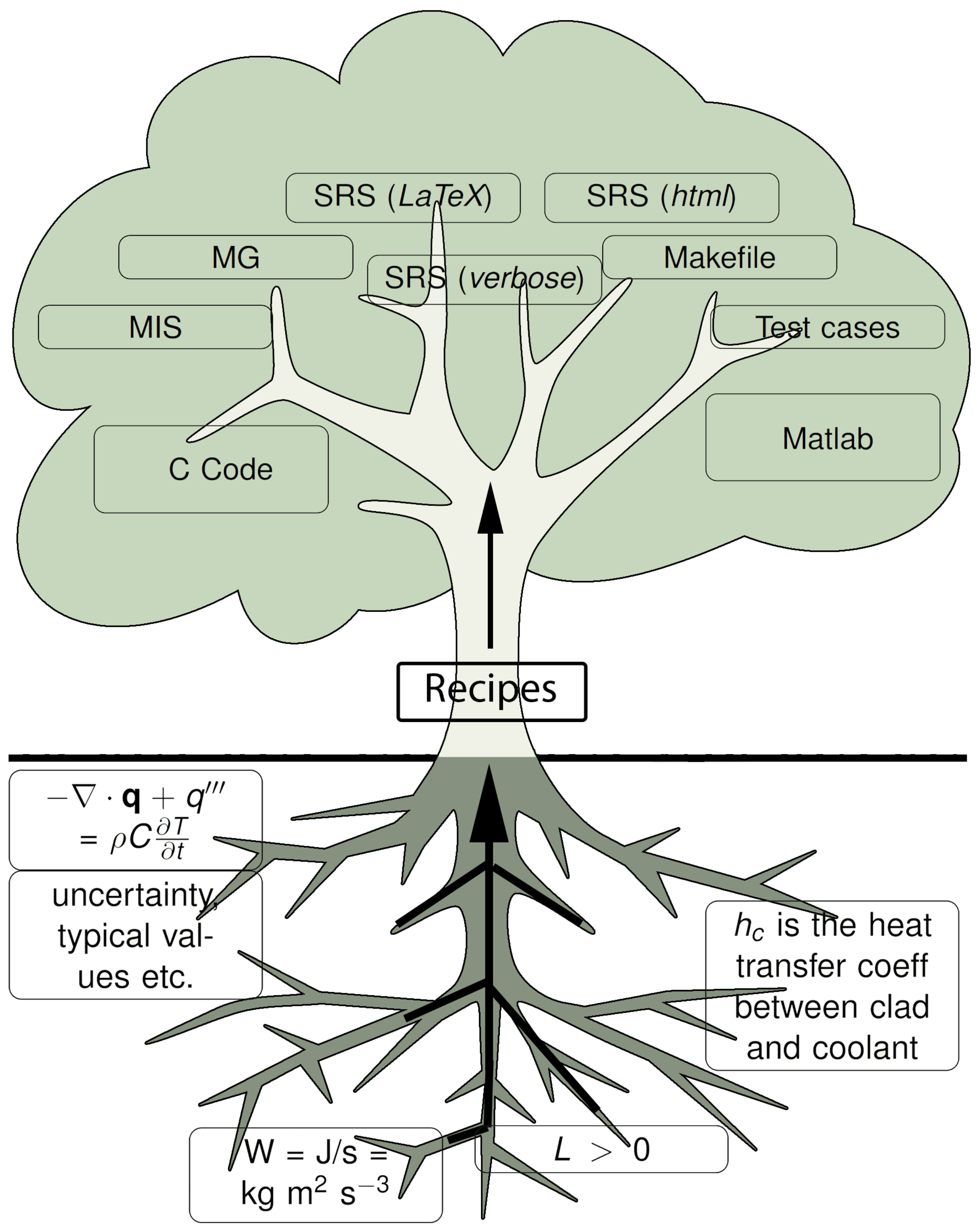
- output: $out := \frac{\text{eval}(x+\Delta x, z) - \text{eval}(x,z)}{\Delta x}$ *# Δx is a defined constant*
- exception: $exc := (\neg Z_0 \leq z \leq Z_{|Z|-1} \Rightarrow \text{OutOfDomain})$

...
slice(x):

- output: $out :=$ FunctT($Z, \langle S_0.\text{eval}(x), ..., S_{|S|-1}.\text{eval}(x) \rangle$)
- exception: None

10. Tool Support

Ideally, the documentation can be generated as different views of the scientific, computing and documentation knowledge [7].


SRS (*LaTeX*)
SRS (*html*)
MG
SRS (*verbose*)
Makefile
MIS
Test cases
Matlab
C Code
Recipes
$-\nabla \cdot \mathbf{q} + q''' = \rho C \frac{\partial T}{\partial t}$
uncertainty, typical values etc.
h_c is the heat transfer coeff between clad and coolant
W = J/s = kg m^2 s^{-3}
$L > 0$

11. Conclusions

No more excuses! Documentation of requirements, design, code and tests cases is not easy, but it is feasible. The long-term reliability, replicability, reusability and maintainability of scientific software depends on documentation. The following ideas from SE facilitate high quality documentation that is flexible to change:

- Faking a rational design process
- Abstraction
- Generality
- Separation of concerns
- Documentation templates
- Anticipation of change
- Replicability
- Information hiding

In the future, increasing tool support will enable documentation that is complete, consistent and traceable by construction [4].

Acknowledgements

The financial support of the Natural Sciences and Engineering Research Council (NSERC), Automotive Partnership Grant, APC 435504-12 is gratefully acknowledged.

References

[1] F. Benureau and N. Rougier. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *ArXiv e-prints*, August 2017.

[2] David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.

[3] W. Spencer Smith. A rational document driven design process for scientific computing software. In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, Chapman & Hall/CRC Computational Science, chapter Examples of the Application of Traditional Software Engineering Practices to Science, pages 33–63. Taylor & Francis, Boca Raton, FL, 2016.

[4] W. Spencer Smith. Beyond software carpentry. In *2018 International Workshop on Software Engineering for Science (held in conjunction with ICSE'18)*, pages 1–8, 2018.

[5] W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

[6] Malavika Srinivasan. Investigating common perceptions of software engineering methods applied to scientific computing software. Master's thesis, McMaster University, Hamilton, Ontario, Canada, 2018.

[7] Daniel Szymczak, W. Spencer Smith, and Jacques Carette. Position paper: A knowledge-based approach to scientific software development. In *Proceedings of SE4Science'16 in conjunction with the International Conference on Software Engineering (ICSE)*, Austin, Texas, United States, May 2016. In conjunction with ICSE 2016. 4 pp.