

P1	P2	P3	P4	P5	P6	P7	P8									
Categories	RE-Elicitation	RE-Process	Process	RE-Awareness	RE-Documentation	RE-Management	RE-Challenges	Quality Requirements	Stakeholders	Tradeoff	Practice	Purpose/ Goal	Testing	Workflow	Context	Characteristics of RS
Codes	No user feedback	Ad-hoc RE due to low number of contributors (-> no need/possibility to discuss design decisions)	Opportunistic, exploratory, unplanned	RE term unclear	Unstructured documentation in Google Docs	Project-Driven, no shared set of requirements	Reproducibility problematic for software written for large HPC/Supercomputer	Extensibility	Multiple stakeholders	Understandability vs. Performance	Prototyping	Implement and test scientific methods	Testing by example (no relation to requirements)	New patch every two weeks; no plan	No permanent positions	Organic growth
	Project plans must be broken down to requirements	"Non-scientific" requirements addressed first to ensure functional correctness	Weekly meetings	RE term not known	Changes documented in PRs	No requirements backlog	Ad-hoc RE due to low number of contributors (-> no need/possibility to discuss design decisions)	Interoperability	Many scientists as users, worldwide; more people working on it; high turnover	Delivering features vs. refactoring software	Prototyping	RS as base asset	Quantified simulation errors as test oracles	No initial plan on which release should include which features	No direct reward for SW	RS is vital to research group
	Requirements defined based on research interests	Requirements discussed with colleagues	Software mainly developed alone (during dissertation)	Realized requirements management and planning is more central than he thought	Documentation with Kanban board; Issues with labels	Issues mainly used as notepad	Requirements change constantly (due to external factors); good base required	Performance	User need: Rapid response for a scientific question	Features vs. intern maintenance	Automatic documentation generation	No defined goals, features, or project work	Regression Testing	(Shortterm) Roadmap with deadlines	Majority of time spend developing software (80-100 %)	Often no special releases
	Requirements were implemented to answer current RQs	Requirements collected (in form of issues), then implemented when they became relevant	Software documentation fell short due to lack of time (esp. during dissertation)	Wants to invest more in RE	Scope of documentation depends on number of participants/developers	RE-prioritization initially via Gitlab milestones, then locally on a list on computer	Quality-requirements suffer due to need for quick research results	Installability	User Demonstrations			Development for own purposes	Automated tests & coverage calculation	Feature-branch workflow applied	> 10 years experience	Research-driven
	Need for something more modular, reengineering; born from dissertation; dynamically evolved	Ad-hoc requirements discussion	Reproducibility must be considered early in development process	No RE awareness	Progress documented in changelog	Priority changes weekly - depending on the features that are needed to make scientific progress	Not all requirements known in the beginning -> leads to disadvantageous decisions	Reproducibility	Devs are not users			Produce scientific results quickly	No testing due to time	No release	50 % time spend developing software	Complexity
	Requirements based on research needs		SE practices driven by personal interest	As-is works out good, do not see need for improvement	No RE documentation	No backlog	Reproducibility requirement must be considered from the start -> if not leads to problems later	Installability				Software development is only a means to gain scientific results	Test based on experience	No defined milestones		Importance of results, not code quality
	Observation		Important to handle one requirement at a time, otherwise negative effect on quality		Missing documentation		Requirement for reproducibility by publishers can be problematic due to massive amounts of data	Platform independence				Software is a means not a product	Testing based on experience	Rolling releases		Central asset
	Discussion with experienced people		No meetings		Progress in Change Log		SE-techniques are not taught	Scalability				Involve researchers in SW dev	No idea on how to derive test cases from data	Rolling releases		Focus on results
	Interviews		Not much communication, everyone just pushes what they need		Documentation behavior emerged over time		Changing requirements might be a "danger"	Conformance to software-stack				Software just to see theoretic ideas in action	Not enough time to do more testing			Not much time
	Prototyping		Upcoming changes, might need more communication		Missing documentation hinders onboarding		Time; deadlines	Performance				Initial goal: next publication	Confusion how to test			Need to convince supervisors etc.
	Driven by a research question		Additions from outside users; certain changes can happen without review, others not		Issues and PR as doc		Lack of knowledge	Maintainability					No extensive testing, only major functions, no higher testing than unit testing			No formal requirements
	No systematic requirement elicitation		Manual backlog		Use issues as changelog		Breaking down a scientific question into requirements	Reproducibility								Redundancy
	Focus groups		Agile not possible because working mostly alone		Ad-hoc requirements discussion, todos in code		No clear benefit because of small teams	Good dependency handling								Dynamic growth
	Mainly requirements on the fly, but more planning will be important for new project		No regular dev meetings		Word document for personal todos (not team!)		Fixed time iterations	Reproducibility								Research-driven
	Plans to make some kind of questionnaire (maybe not as structured)		Monthly sprint meetings with issue board				Publication more important than SW	Performance								Focus on results
	Ad-hoc requirements		No scheduled meetings, but planned to				More feature requests in less time	Reusability								Working software > quality
			No onboarding, but documentation				Technical debt	Reproducibility								No awareness for quality
			Discussions about software often "emotional"					Performance								No releases
			Attempt to establish code reviews					Portability								Research-driven
			Warn others when pushing					Reproducibility is important but no one does anything for it								Focus on results
			Rarely talk about the code as such					Portability								
			No managed integration					Performance								
								Platform independence								
								Reproducibility								