SAND2022-3977C

Sandia National Laboratories

Exceptional service in the national interest

A Tiered Approach to Scientific Software Quality

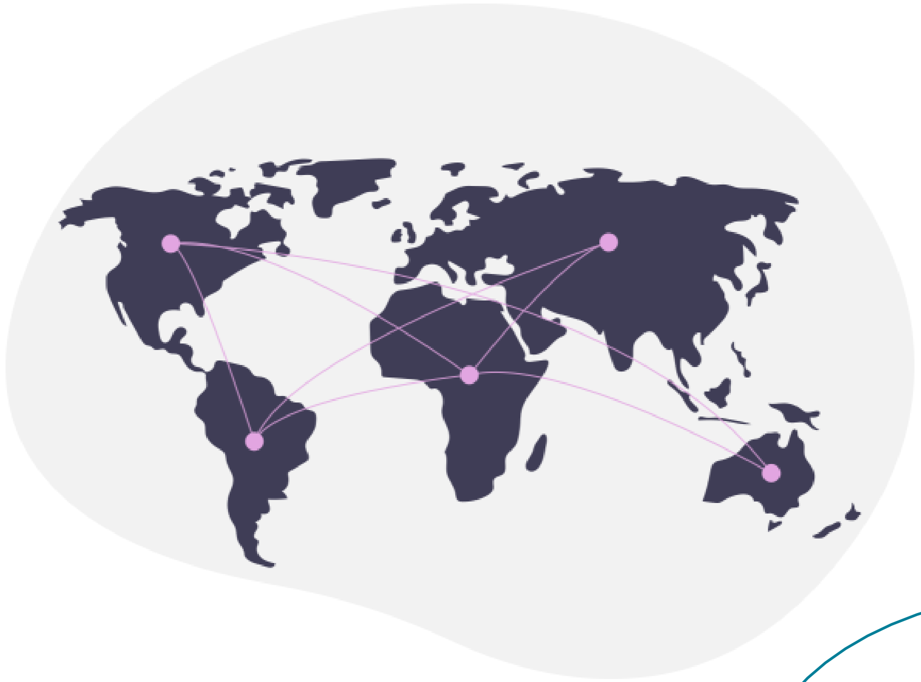
Presenter: Miranda Mundt – mmundt@sandia.gov
Coauthors: Wade Burgess, Dena Vigil

SEA's Improving Scientific Software Conference 2022
April 5th, 2022 – 10:20-10:50AM MDT

U.S. DEPARTMENT OF ENERGY

NNSA

Introduction



- Software quality is not a “one size fits all” problem
- How does scientific software differ from commercial software?
  - Primarily self-taught developers
  - Unique challenges (requirements, testing, funding, performance)
  - End goal: Progression of science

Images from undraw.co

Background & Motivation

- Software quality in the Center for Computing Research (CCR) at Sandia National Laboratories was not standardized
  - Each project lead determined their own definition and guidelines for quality
- Desire: Create a standardization framework for CCR
  - Problem: Existing software quality frameworks are aimed towards commercial software
  - Question: How do we *right-size* a framework?
  - Goal: Create a tiered software quality framework that scales and evolves naturally with a software project

Image from undraw.co

Methodology

Interviews

- All department managers within CCR
- One-hour, in-person interviews
- Open-ended with the goal of gathering answers to:
  - What is your definition of software quality?
  - What do you consider to be you process for quality?
  - What would be your trigger for a more rigor?
  - How do your stakeholders enforce quality?
  - How do you measure the success of a project?

Surveys

- Software project leads within CCR
- Anonymous web-based form
- Structured questions aimed at collecting information regarding:
  - Project metadata (e.g., maturity, size)
  - Value of research and software development activities
  - Current state of practice

Rapid Review

- Systemic, time-boxed literature review
  - Faster turnaround at the cost of certain steps (e.g., limited literature search)
- Motivated by a practical problem
- Two unique research questions to explore:
  - Do different teams work better under different sets of quality standards?
  - How do we *right-size* a software quality model?

Interviews

Fundamental Research

"It's clear that we are not a software development shop – we are a research development shop."

Triggers for more rigor

"Until we really had a sponsor that demanded a higher level of rigor (e.g., process documentation, traceability), there had to be a clear benefit for all of those [in order for us to do them]."

Non-prescriptive guidelines

"When I had my first ASC software quality assessment, one of the reviewers told me, 'We're not here to expose external processes for you. You clearly get work done - therefore, you have processes.'"

Surveys

Metadata

Maturity	%
Proof of concept	0
Somewhat exploratory	29
Somewhat productionized	50
Very productionized	14
Other	7

Team size	%
1-3	50
4-6	14
7-10	14
11+	22

Members with SWE Training	%
0-24%	50
25-49%	14
50-74%	14
75-100%	22

	Non-software end products			Software development activities		
Perceived Importance	Publications	Presentations	Research answers	SDLC *	Testing **	U&S Considerations ***
Not important	7%	0%	7%	4.8%	9.3%	9.2%
Somewhat important	29%	14%	21%	24.1%	37%	29.2%
Important	43%	71%	29%	44.6%	27.8%	35.4%
Very important	21%	14%	43%	26.5%	25.9%	26.2%

* Software development life cycle (SDLC) includes: software architecture; software design; software development; software release/deployment; software stability; software extensibility

** Testing includes: regular verification testing; regular validation testing; regular functionality testing; regular unit testing

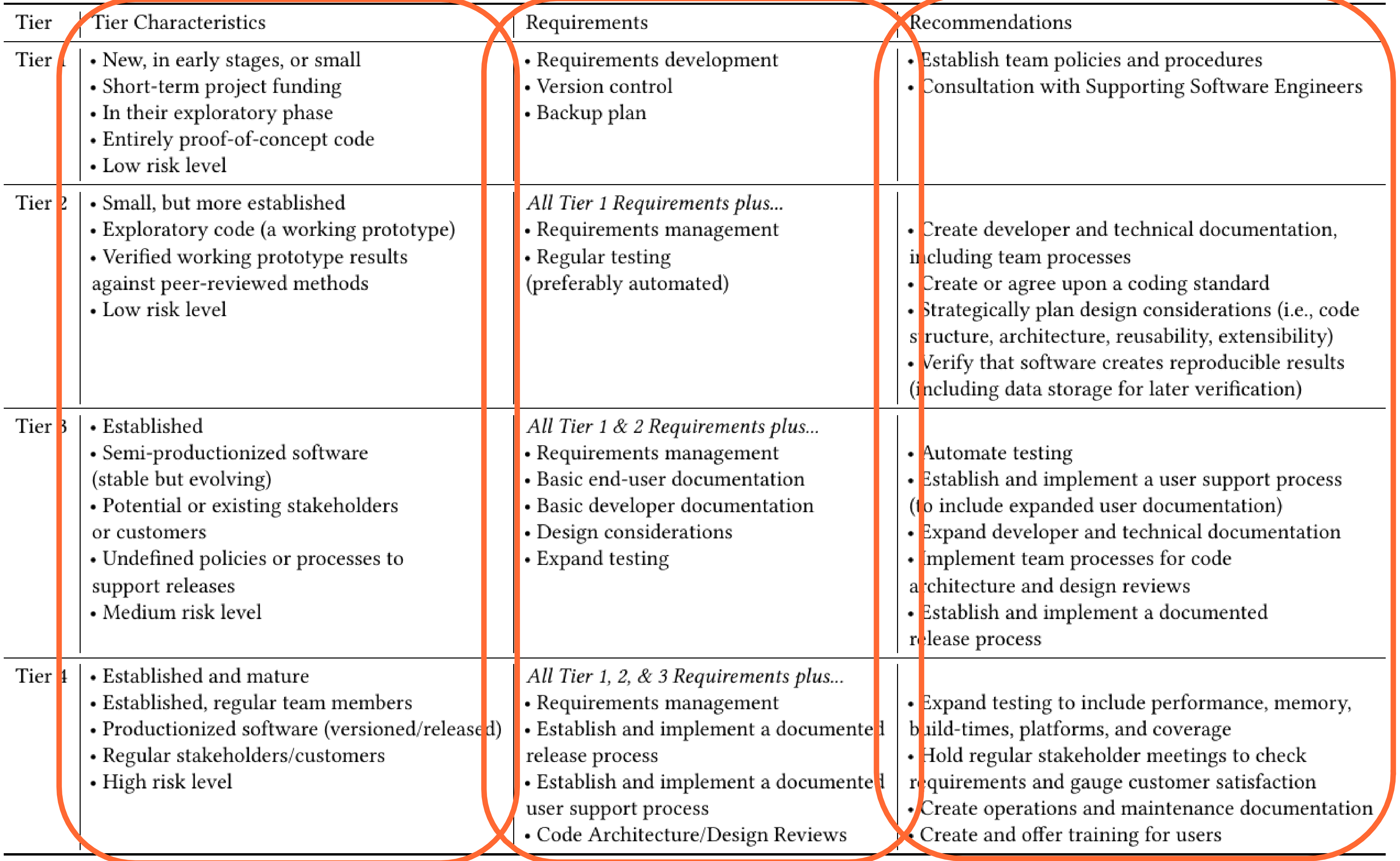
*** User & Stakeholder (U&S) Considerations includes: stakeholder specifications and requirements; stakeholder satisfaction; user experience (installation); user experience (usage); maintenance and support

Full survey questions can be found in accompanying paper.

Rapid Review

Relevant Factor	Description	Key Takeaway
Individual	Developers as individuals bring unique skills and perspectives to a project. This includes not only a familiarity with the problem domain or training in software development, but also motivations and past experiences.	A scientific software developer who has received formalized training in software development will ultimately create higher quality code in a more productive manner.
Team	A development team consists of several members working together on a software package with collaborative intent. This includes a sense of common identity, clear goals, and the rate of turnover.	Good teaming enables productivity and quality. That is, when team processes and workflows are clearly defined and followed, teams will be more productive, create higher quality code, and be able to support new development.
Organizational	An organization represents a shared value system and understanding of business goals and has sway over the direction of its projects. This includes support and commitment from upper management, budget allocations, and relationships to neighboring projects.	In order to achieve productivity and quality, a project must be in alignment with the organization's values and processes, but the organization or funding source must also value quality and allocate funding towards quality activities.
Technology	Tools for software quality practices supply efficient ways to manage those activities. This includes the use of software development tools such as version control, issue tracking, and code analysis.	The right tools used in the right way can improve quality. That is, technology can both enable and inhibit quality, depending on usage.
Process	Processes exist to deterministically designate the steps taken for a particular task. This includes the use of development methodologies, whether well-defined or ad hoc, and the extent to which the project is committed to using those methods.	Adherence to a managed and well-defined process for software development is likely to result in quality and productivity.
Customer	A customer may be a user or a stakeholder and can influence the direction of a project. This includes the frequency of changes in requirements, the extent of user involvement in the development, and users’ resistance to change.	N/A

Tiered Software Quality Framework



Tier	Tier Characteristics	Requirements	Recommendations
Tier 1	• New, in early stages, or small • Short-term project funding • In their exploratory phase • Entirely proof-of-concept code • Low risk level	• Requirements development • Version control • Backup plan	• Establish team policies and procedures • Consultation with Supporting Software Engineers
Tier 2	• Small, but more established • Exploratory code (a working prototype) • Verified working prototype results against peer-reviewed methods • Low risk level	*All Tier 1 Requirements plus...* • Requirements management • Regular testing (preferably automated)	• Create developer and technical documentation, including team processes • Create or agree upon a coding standard • Strategically plan design considerations (i.e., code structure, architecture, reusability, extensibility) • Verify that software creates reproducible results (including data storage for later verification)
Tier 3	• Established • Semi-productionized software (stable but evolving) • Potential or existing stakeholders or customers • Undefined policies or processes to support releases • Medium risk level	*All Tier 1 & 2 Requirements plus...* • Requirements management • Basic end-user documentation • Basic developer documentation • Design considerations • Expand testing	• Automate testing • Establish and implement a user support process (to include expanded user documentation) • Expand developer and technical documentation • Implement team processes for code architecture and design reviews • Establish and implement a documented release process
Tier 4	• Established and mature • Established, regular team members • Productionized software (versioned/released) • Regular stakeholders/customers • High risk level	*All Tier 1, 2, & 3 Requirements plus...* • Requirements management • Establish and implement a documented release process • Establish and implement a documented user support process • Code Architecture/Design Reviews	• Expand testing to include performance, memory, build-times, platforms, and coverage • Hold regular stakeholder meetings to check requirements and gauge customer satisfaction • Create operations and maintenance documentation • Create and offer training for users

Threats to Validity

Interviews

- Only one level of managers interviewed
- Open-ended interviews made quantitative extraction of data difficult

Surveys

- Reasonable response rate (14 out of 200, 7%)
- Lack of response from "least mature" category
  - Possible skew towards more production-ready values
  - No information on current practices – potential that Tier 1 is not appropriately right-sized

Rapid Review

- No full systemic review
- Omitted steps
  - Extensive literature search
  - Lower quality appraisal
- Fairly confident that this is a low risk

Framework

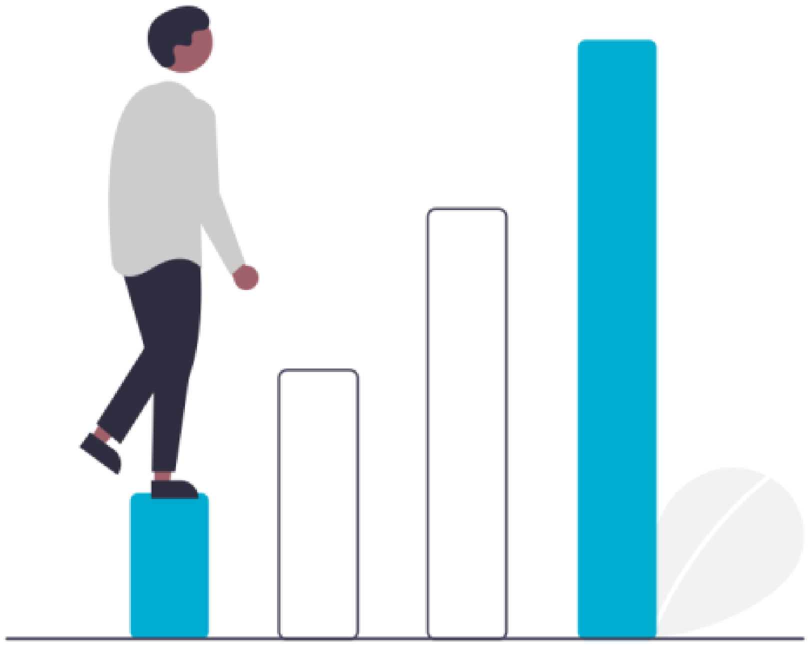
- Lack of strong enforcement
- Released during peak of COVID restrictions
  - No changes made in release process to account for this
- No reliable data for adoption and efficacy rate

Future Work

- Efficacy and reproducibility of framework creation process
- Case studies outside of national laboratories (e.g., academia)
- Mapping to common recognized measures of maturity or quality (e.g., technology readiness levels (TRLs) or ISO/IEC 25010)
- Integration of research quality into scientific software quality framework

Image from undraw.co

Conclusion



- Motivation: Provide a set of software quality assurance guidelines for scientific software developers
  - How do we *right-size* software quality for scientific software projects?
- Methodology: Interviews, surveys, and rapid literature review
  - Gauge value, current practices, and recommended practices
- Goal: Create a tiered framework that scales and evolves naturally with a software project

Image from undraw.co

Q&A