Leveraging Expertise to Support Scientific Software Process Improvement Decisions

Erika S. Mesh, Gabbie Burns, and J. Scott Hawker | Rochester Institute of Technology

Leveraging the underlying expertise and motivational factors that drive scientific software development practices, a hybrid scientific software process improvement framework (SciSPIF) is being constructed to allow scientific software developers to make SE process improvement decisions tailored to their own goals. Here, the strategy for designing and building SciSPIF is reported, along with preliminary insights.

Scientific software developers may not all be trained software engineering (SE) experts, but they're experts in their domains and the needs of their work. As the complexity of this scientific software increases, scientists are also often expected to be experts in the construction and maintenance of the software that supports their research. To support this paradigm, a number of different specialized approaches have emerged. As evidenced by the focus of this special issue of *Computing in Science & Engineering* (CiSE), computational scientists do value the quality of their software and are successfully using SE practices to support their research. Thus, our primary goal isn't to determine *if* scientists use/value SE practices, but to understand *why* and *how* they do so. In other words, what motivations and scenarios precede the successful adoption of SE practices and process improvements, and what patterns exist in the decision-making process? By using this insight to supplement existing SE knowledge and references, we are constructing a new scientific software process improvement framework (SciSPIF),[1] that allows scientific software developers to "self-drive" their SE process improvement activities according to their own goals.

SE for Scientific Software: A Process-Level Perspective

In any software development *project*, there are three key components: the *people*, the *product*, and the *process* used to tie the project together.[2] This process consists of a set of key practices, the specific procedures to implement them, and an overarching structure defining how these practices relate to each other. The goal of the process is to guide people in how to consistently achieve a high-quality software product within the available resource constraints (for example, the schedule, computing infrastructure, budget, personnel, and so on).

Regardless of the end process used, all software processes are defined via iterative assessment, planning, and enactment activities.[3] At each iteration, the project's current status is assessed with respect to its overall quality goals to create a plan to adapt the process to meet these goals. The planning activity involves a series of individual decisions regarding the overall goals and which practices to include for addressing these goals most effectively. Software development teams often use software process improvement frameworks (SPIFs) to aid in these decisions. These SPIFs can be considered via two basic categories: *prescriptive* and *supportive*.[3] To consider the applicability of existing SPIFs in scientific domains, the assumptions made by each must be considered.

- *Prescriptive SPI frameworks* (see Figure 1a) leverage the experience and commonalities of successful software development teams to suggest SPI plans. Using these plans, teams tailor a specific set of procedures to meet the needs of their project. This provides an assessment and planning structure that allows for comparison across projects and domains, but sacrifices the flexibility of supportive approaches. Prescriptive frameworks (Figure 1a) address this via assumptions regarding

 1521-9615/14/$31.00 © 2014 IEEE Copublished by the IEEE CS and the AIP

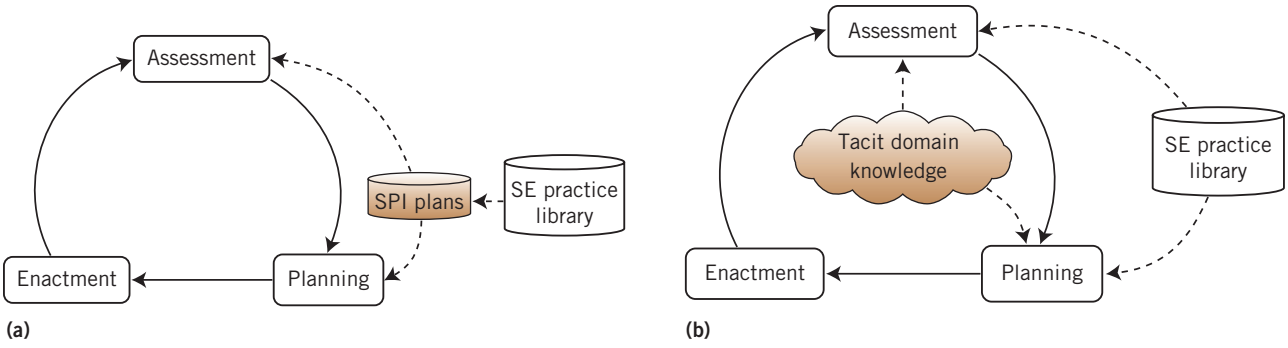

Figure 1. Types of software process improvement frameworks (SPIFs). (a) Prescriptive SPI frameworks leverage the experience and commonalities of successful software development teams to suggest SPI plans. (b) Supportive SPI frameworks supply a basic understanding of the common drivers of SPI decisions and the available information, but stop short of providing any strict SPI limitations or priorities.

common improvement priorities. However, the wide range of domains and situations in which scientific software is utilized[4] and mixed priorities[5] suggests that a common approach for all scientific software projects is impractical.

- *Supportive SPI frameworks* (see Figure 1b) supply a basic understanding of the common drivers of SPI decisions and the available information, but stop short of providing any strict SPI limitations or priorities. The project team determines its own SPI plans based on their own specific needs. While this approach provides the flexibility to support a wide range of scientific software projects, using a supportive SPIF depends heavily on the team's ability to identify situations that "resonate" with the goals associated with each practice. The mixed levels of expertise in scientific software projects[6] makes this difficult.

Deciding on a prescriptive or supportive SPIF for a particular SPI effort is akin to choosing a GPS device with automated route calculations versus a searchable map with road details, but no assumptions about the "optimal" route. If you know exactly where you're going and the benefit of a clear, exact route outweighs the need for customization, a GPS device is an ideal solution. In contrast, if you need flexibility in destination or have unique priorities for the route itself (for example, waypoints, optimizing for scenery over time, and so on), a high-quality map that provides guidance without decisions may be more appropriate.

The challenge in selecting the appropriate type of SPIF for scientific software is that neither provides both the simultaneous knowledge and flexibility required for use in scientific settings. While a process tailored to the needs of each specific project is required,[5] an exhaustive (that is, prescriptive) set of guidelines for creating such a process is impractical. Similar scenarios regarding complex problem solving support this conclusion. In a study of the efficacy of copier repair technicians, Julian Orr[7] found that prescriptive manuals were ineffective. The variety of possible scenarios and complicating factors was simply too broad to allow for an exhaustive manual that was still accessible by the target audience. Instead, technicians were provided a means to share lessons learned among them to find representative scenarios for their current problem. As a result, the process of locating a potential solution was improved and repair rates grew dramatically.

In the study of judgment and decision making, this process of decision selection based on a similar scenario instead of an exhaustive analysis of all possibilities is known as recognition-primed decision making.[8] This strategy lets experts rely on intuitive recognition of an appropriate course of action to make decisions where "sufficient and fast" is more important than "optimal." Further, as James Shanteau[9] discusses, such intuitive and heuristic approaches don't necessarily impact the overall quality of the decision. Reliance on expert knowledge rather than full analysis can be quite effective for tasks that are well-understood, repetitive, and decomposable, not unlike many software development tasks (once the project concept and needs are established). Similarly, the "inductive" nature of a supportive framework is often much better suited for small projects with domain-specific needs.[3] However, we can't assume scientific software developers will recognize their own concerns in the existing SE literature as required by a supportive framework.

SciSPIF

Rather than requiring scientific software developers to adjust their priorities to fit the goals of a

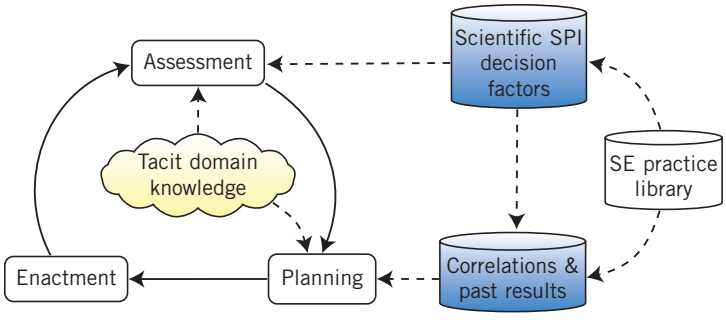

Figure 2. The Scientific SPI Framework (SciSPIF)[1] provides (a) planning guidance in the context of scientific projects (blue) for practitioners to correlate to their own tacit knowledge (yellow), and (b) references to supporting SE resources, but stops short of a fully prescriptive approach.

prescriptive framework, or to dedicate time to extensive training to be able to leverage existing supportive frameworks, we propose to bridge the gap via a hybrid SPIF to provide the support scientific software development teams require without limiting their ability to "self-drive" their SPI decisions. SciSPIF (see Figure 2)[1] will be a supportive approach built upon common SPI decision criteria and correlations to SE practices derived from real-world scientific software projects. In this way, scientific software teams can make SPI decisions in the context of their own tacit knowledge while being supported by an SE practice library that links them to common SE techniques and resources.

Although useful as a model of how existing SE knowledge can be combined with practitioner insights to guide scientific SPI, SciSPIF offers little practical advantage to the scientific community at large without a large amount of data. Our ongoing work is focused on populating this model with data and correlations from representative scientific software projects via three primary questions:

- Which mainstream SPI planning criteria are applicable to scientific software development projects?
- Which other factors influence priorities and decisions during scientific SPI planning activities?
- Which scientific SPI planning criteria are applied in decisions about specific SE practices?

Answering these questions could be approached either experimentally or empirically, and each approach has its merits. An experimental approach would test hypotheses regarding each possible decision factor and SE practice correlation. This allows for a higher degree of control and potentially stronger, more quantitative results, but is limited by our knowledge at the time the experiment is designed. In contrast, an empirical study offers the chance to uncover unexpected, but potentially high-reward concepts. In addition, designing and conducting experiments to cover even only the most likely scenarios would take substantial time and resources. An empirical approach allows us to leverage the fact that these scenarios, and many more, already exist in the scientific community.

At their core, grounded theory (GT) methodologies are designed to embrace change in research scenarios with mixed degrees of determinism.[10] In other words, when studying actors in the real world, anything can happen. "Anything" often is limited by a set of constraints and conditions, but these conditions may be so varied or complicated that a deterministic evaluation is infeasible. Like a software process model, the GT canons and procedures provide an overall structure, a set of recommended practices, and clear guidelines to assess the research. Similarly, just as a software process model alone isn't enough to formally define a project's process, a GT research project must also define how it implements the required canons and procedures to allow for objective evaluation and reproducibility.

Implementation Strategy

A central practice of grounded theory methodologies is the encoding of qualitative data and narratives into theoretical constructs and the relationships between them. In this way, qualitative information can be systematically analyzed to generate new theories firmly grounded in real-world data. The continuous collection and analysis of data is considered critical to allow the study to embrace change and generate novel theories. At any given point in time, a certain amount of literature has been analyzed and a certain number of interviews conducted and processed. Over time, the theoretical constructs generated by each analysis activity and the relationships between them feed back into the data collection and analysis methodology. Just as an iterative software development process allows for continuous improvement of the product and process, a GT approach allows for continuous improvement of the research constructs and the methodology itself.

The encoding process begins with an open analysis. Each data source (for example, research literature, an interview transcript, and so on) is reviewed line-by-line, and concepts relevant to the research questions are encoded as nodes or relationships between nodes. As this process progresses, a concept may repeat itself.

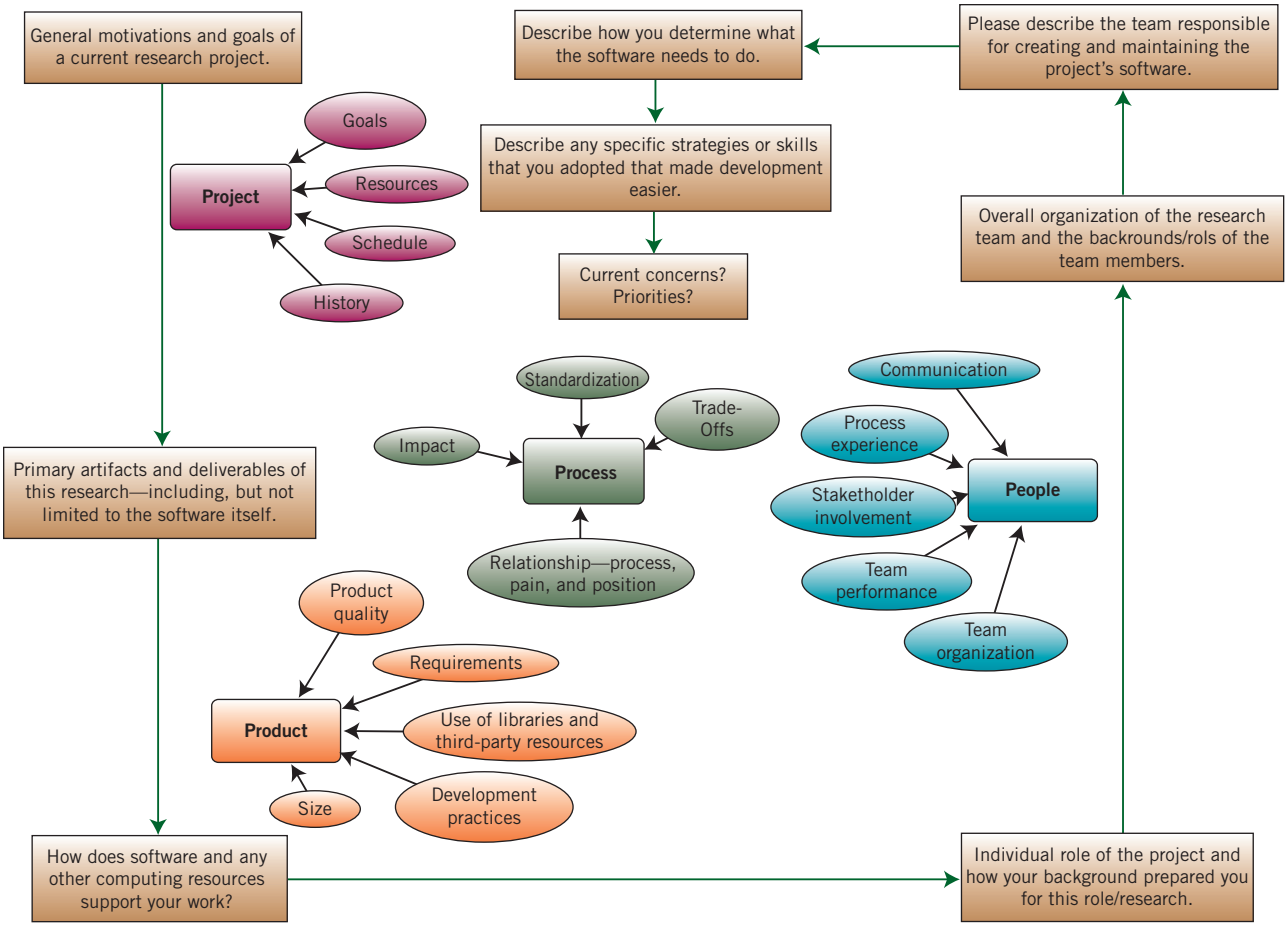

Figure 3. Visual outline of the interview structure being used. Brown boxes show the initial questions/requests to the subjects. The rectangular nodes represent aggregate categories of key concepts to be discussed, while the oval nodes offer examples of common subtopics (as discovered via earlier GT analysis of SE and scientific software literature). The remaining colors are for visual distinction between the four subgraphs: Project (purple), Product (orange), Process (green), and People (blue).

In this case, axial analysis allows for the recognition of overlapping and related concepts. Some axial analysis is done in tandem with the open encoding process, and more is done following the full analysis of the data source to look for broader categories. As a final step of axial analysis, nodes and relationships identified by multiple researchers are reviewed by the team as a whole, and a final set of constructs is agreed upon.

Through the axial encoding process, any concept that's potentially relevant is included in the theoretical model. Following the axial analysis of a data source (or series of data sources), selective encoding is performed to review the constructs (such as SPI planning criteria identified as relevant in various data sources) for overarching themes, redundancy, and relevance. To prevent investigator bias and preconceived expectations from impacting the results, the open and axial encoding processes are, by necessity, liberal and err on the side of including anything that might be relevant to SPI planning. Thus, the selective analysis of the resulting constructs ties the results back to the original research questions.

Data Sources and Preliminary Analysis

While scientific software is unique in some key aspects, it significantly overlaps with well-studied areas of software engineering. With this in mind, the encoding procedures have been applied in an iterative manner to a set of foundational software engineering and published scientific software experience reports. This began with an analysis of five scientific software case studies,[6] and two published reports regarding scientific software process improvement experiences[11,12] as both a learning exercise in GT analysis and as a way to generate a base node structure (see the center graphs in Figure 3).

Figure 4. Word clouds of the top 500 words in data sources by category (over three characters long and excluding common stop words, such as "while" or "because"). (a) Published scientific software case studies.[6,11,12] (b) Interview subjects' responses.

Although formal analysis of the available literature will be ongoing, the base node structure identified was sufficient to design a semistructured interview worksheet (shown in the outer line of discussion points in Figure 3) to begin conducting interviews. As people and project-related criteria are highly referenced in both SE and scientific software literature, the interviews began by clarifying the background of the project and subjects. The remainder was dedicated to delving into the details of the subject's current development product, how it was created, and the rationale behind the use of specific practices.

For each topic, interviews were conducted as an open conversation initiated by a broad request or statement. In this way, the interview was kept on track and organized without the risk of biasing the subjects towards specific answers or limiting the conversation in a way that would prevent discovery of new concepts.

An initial analysis of the interview transcripts indicates that some existing SPI decision criteria are likely to be applicable in scientific domains although the priorities may be different in different scenarios. This conclusion has also been implied by previous analysis of scientific software domains done by Diane Kelly[5] and is demonstrated via the visualizations shown in Figure 4. The word clouds shown demonstrate the top 500 words in the three case studies analyzed (Figure 4a) versus the transcripts of five interviews conducted with principal investigators (PIs) at the Rochester Institute of Technology (Figure 4b) representing eight different domains. (We should note that the transcripts analyzed included only the verbal responses of the subjects, not the questions from the interviewers.) As you can see in Figures 4a and 4b, the two images denote a clear distinction. The case studies focus on large projects and were written by authors with SE expertise. Conversely, the interview transcripts reflect the thoughts of PIs with little to no formal SE training, working on small academic research projects that regularly include graduate students. Although there are many commonalities (such as development, coding, and software), the case studies use more "traditional" SE terminology (such as model, team, tools, and engineers), while the PIs focus on practical and more academic terms (work, knows/knowledge, and students). We should also briefly note that the word "work" was most often used in the context of quality (for example, "it works").

Like any study involving human subjects, our work is limited by the amount of information that people are willing to share and their own memory and understanding of their actions. Uncovering

themes and grounded theories in tacit knowledge and unconscious decisions is a challenging and time-consuming process. Additionally, until we get more data, SciSPIF can give insight into the scientific SPI decision-making process, but isn't ready for the scientific community to use as a standalone resource. As the data collection and analysis activities proceed, distinctions such as this and other patterns/insights will lead to the development of a set of theories to describe the application of SE practices in scientific software projects. Once a reasonably useful set of decision criteria and correlations is collected, an online, interactive version of SciSPIF will be implemented and disseminated. This has two primary benefits:

- Scientists get access to ongoing research and the lessons learned of their peers, which could provide some immediate benefit to their own work.
- Lessons learned regarding the portions of SciSPIF that are most useful or requests for the inclusion of additional information will provide invaluable feedback about possible holes in our working theory of the relevant scientific SPI priorities and practices.

Long-term, this data can be used to generate and test new hypotheses regarding SE process improvements for scientists. This incremental delivery of value and methodology refinement is also in line with our long-term research goals: to provide value to both the scientific and software engineering research communities. Teaming with other SE researchers and scientific teams will allow our work to provide immediate value to specific scientific research teams while also advancing the field of study and producing lessons learned that could assist other software engineering process researchers.

In the meantime, anyone interested in learning more or participating in these efforts (as a research subject or investigator) is welcomed and encouraged to contact Erika S. Mesh directly at esm1884@rit.edu. With their own personal experiences and lessons learned from their peers, the support of training classes (such as www.software-carpentry.com), and published experience reports and articles (including this special issue of *CiSE*), scientific software teams already have access to the information needed to make informed SPI decisions for their own projects. With SciSPIF, we hope to understand how teams make these decisions, and we also hope to offer guidance and heuristics to help the scientific community at large. ■

References

1. E. Mesh and J. S. Hawker, "Scientific Software Process Improvement Decisions: A Proposed Research Strategy," *Proc. 5th Int'l Workshop on Software Engineering for Computational Science and Engineering*, 2013, pp. 32–39.
2. I. Jacobson, G. Booch, and J. Rumbaugh, "The Four Ps: People, Project, Product, and Process in Software Development," *The Unified Software Development Process*, ch. 2, Addison-Wesley Longman, 1999.
3. F. Pettersson et al., "A Practitioner's Guide to Light Weight Software Process Assessment and Improvement Planning," *J. Systems and Software*, vol. 81, no. 6, 2008, pp. 972–995.
4. J. Hannay et al., "How Do Scientists Develop and Use Scientific Software?" *Proc. 2009 ICSE Workshop on Software Eng. for Computational Science and Eng.*, 2009; doi:10.1109/SECSE.2009.5069155.
5. D. Kelly, "An Analysis of Process Characteristics for Developing Scientific Software," *J. Organizational and End User Computing*, vol. 23, no. 4, 2011, pp. 64–79.
6. J. Carver et al., "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies," *Proc. Int'l Conf. Software Eng.,* 2007, pp. 550–559.
7. J.E. Orr, *Talking About Machines: An Ethnography of a Modern Job*, ILR Press, 1996.
8. G.A. Klein, "Recognition-Primed Decisions," *Advances in Man-Machine Systems Research*, JAI Press, vol. 5, 1989, pp. 47–92.
9. J. Shanteau, "Competence in Experts: The Role of Task Characteristics," *Organizational Behavior and Human Decision Processes*, vol. 53, no. 2, 1992, pp. 252–266.
10. J. Corbin and A. Strauss, "Grounded Theory Research: Procedures, Canons, and Evaluative Criteria," *Qualitative Sociology*, vol. 13, no. 1, 1990, pp. 3–21.
11. D. Kane, "Introducing Agile Development into Bioinformatics: An Experience Report," *Proc. Agile Development Conf.,* 2003, pp. 132–139.
12. S. Easterbrook and T. Johns, "Engineering the Software for Understanding Climate Change," *Computing in Science & Eng.*, vol. 11, no. 6, 2009, pp. 65–74.

Erika S. Mesh is a PhD student and US National Science Foundation Graduate Fellow at the Rochester Institute of Technology (RIT). Her primary research interests include software engineering process improvement for computational science and engineering (CSE) domains, program comprehension, software engineering knowledge management, and requirements engineering. Mesh

has an MS in software engineering from RIT and is a student member of IEEE and the ACM. Contact her at esm1884@rit.edu.

Gabbie Burns is currently working as a software tester at SmugMug in Mountain View, California. Her research interests include the tradeoffs of software engineering process models, particularly agile development, and tailoring the application of such models to maximize the effectiveness of a development team. Burns has an MS in software engineering from the Rochester Institute of Technology. Contact her at gbb5658@rit.edu.

J. Scott Hawker is an associate professor in the Department of Software Engineering at the Rochester Institute of Technology. His research interests include computational cyberinfrastructure for environmental modeling and decision support, collaborative learning and knowledge management systems, model-driven software development, and software process engineering. Hawker has a PhD in electrical engineering from Lehigh University and is a member of IEEE, the ACM, and the American Society for Engineering Education (ASEE). Contact him at hawker@mail.rit.edu.