Scientific Software Process Improvement Decisions: A Proposed Research Strategy

Erika S. Mesh
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York 14623 USA
esm1884@rit.edu

J. Scott Hawker
Software Engineering
Rochester Institute of Technology
Rochester, New York 14623 USA
hawker@se.rit.edu

***Abstract*—Scientific research is hard enough; software shouldn't make it harder. While traditional software engineering development and management practices have been shown to be effective in scientific software projects, adoption of these practices has been limited. Rather than presume to create a prescriptive scientific software process improvement manual or leave scientists to determine their own plans with only minimal references as support, we posit that a hybrid approach is required to adequately support and guide scientific SPI decisions. This paper presents a grounded theory approach for determining the driving factors of scientific software process planning activities in order to generate supporting data for a proposed Scientific Software Process Improvement Framework (SciSPIF).**

I. Introduction

Modern scientific research projects rely heavily on complex, customized software. Despite the fact that many scientific software programs will never be commercialized, they can be considered among the most valuable programs developed. Unfortunately, software bugs that would be preventable (or at least detectable) in commercial software development organizations have been shown to have significant consequences in scientific domains [1]. While mainstream software engineering (SE) practices have been shown to be effective in scientific software projects [2], adoption of modern SE techniques has been limited, and a gap has emerged between the scientific and SE communities [3].

Past research offers some explanations for this gap [3]–[5]; however, research to enable scientific SE process improvement (SPI) has been limited to checklists [6], [7], general models [8], and project specific experience reports (e.g. [9]). While valuable, this work has not yet been shown to be applicable on a large scale to the broader scientific community.

Scientific software developers are often domain experts and have expert knowledge regarding the requirements and constraints for their particular software development project [4], [10]. Thus, although they may require support with respect to specific SE terminology and practices, they are the best suited to identify and prioritize the areas of their project(s) that need improvement. Simple guidelines mapping common scientific software concerns to general SE principles and concepts will allow scientists to leverage their domain knowledge and drive their own SPI activities with greater success than existing prescriptive approaches. This paper presents a grounded theory [11] approach for and preliminary results of a study of the traditional and domain-specific criteria that influence scientific SPI planning priorities and decisions.

II. Project Context

A. Scientific Software

Software has become a critical component in many forms of scientific research. However, this dependence on software brings with it a risk of impacting not only the immediate users, but also the field of study as a whole [1].

Software developed for and used during scientific research activities is described via a number of terms that overlap in meaning and significance. "Computational science" is most commonly used to refer to complex or large computing calculations that support scientific or engineering endeavors [12]. More generally, software developed to support scientific research, regardless of the required processing or data resources, is referred to simply as "scientific software" and has three primary characteristics [4], [10], [13]:

1) The software product's main purpose is to support the exploration of a scientific question, not answer it directly.
2) The software product is developed by, or via close interaction with, the domain experts.
3) Scientific software suffers from a lack of known results/expected outputs making verification and validation difficult (i.e. the "oracle" problem).

Studying how existing SE knowledge can be applied to the scientific community has resulted in several promising insights. A survey by Carver et al. [5] suggests that lack of practice adoption is tied to a lack of knowledge regarding specific techniques. Kelly [10] discusses how scientific software projects must adapt quickly to advances in the domain while also ensuring data integrity and experiment repeatability. These mixed goals make the determination of an appropriate SE process challenging. Proposed solutions include enumerations of recommended practices [7], customizations of specific techniques [14], and increased training [3].

A number of potential secondary characteristics affect the applicability of traditional SE practices and SPI techniques to scientific software development projects. As a number

978-1-4673-6261-0/13/$31.00 © 2013 IEEE

SE-CSE 2013, San Francisco, CA, USA

of surveys have demonstrated, there is a wide range of SE experience, software purposes (e.g. internal or external users), development environments, and team organizations in scientific software development projects [13], [15]. Further understanding of how these factors impact scientific SPI is required before a generalized SPI framework can be proposed.

B. SE Process Improvement

When discussing software development projects, both *what* is to be created and *how* it is created must be considered. In software engineering terminology, "what" is the software product and "how" is the software engineering process (or simply SE process or process) [16]. As the process defined for a project matures and is more formally defined and modeled, its ability to ensure consistent and high-quality software development increases. Regardless of the actual process employed, the guidance of process maturation is referred to as SE process improvement (SPI) and is achieved via an iterative cycle of assessment, planning, enactment, and re-assessment [17], [18] (Fig. 1).

This cycle can proceed in a standardized way, known as a prescriptive approach (Fig. 2), dictating specific plans for the prioritization and enactment of process improvement activities. In contrast, the SPI cycle can also be tailored to the needs of the project where the most valuable process improvements are determined based on the concerns of a given project. In the latter case, SPI planning activities are "inductive" [17] in nature. The common underlying SPI framework can be viewed as supportive (Fig. 3), rather than prescriptive.

The SEI Capability Maturity Model (CMM) [19] offers a prescriptive assessment framework via sets of staged Key Process Areas (KPAs) for gradual process improvement. Assessment results against the same set of KPAs can also be used in an inductive manner by customizing the prioritization and scheduling of process improvement activities. This is referred to as a continuous representation.

While the staged representation allows for greater consistency and comparisons across projects and organizations, small or dynamic organizations like scientific research teams may find many of the required KPAs to be irrelevant for their situations. In these cases, "inductive" approaches are often more applicable [17].

For non-traditional domains, continuous "roadmaps" allow for a customized approach based on the needs of the project [20]. Kroll and MacIsaac [21] take a similar inductive approach by suggesting levels of application of their process disciplines and practices. In order to determine which practices to focus on first for SPI, the authors suggest two alternate approaches: (1) Review the practices and the problems they address for issues that resonate with the project under consideration, (2) Review the project's history and identify past problems/current priorities, then look for practices that address these issues. At a higher level, Boehm and Turner (B&T) [22] suggest assessment of a project along five dimensions in order to suggest if agile or plan-driven processes are more applicable.

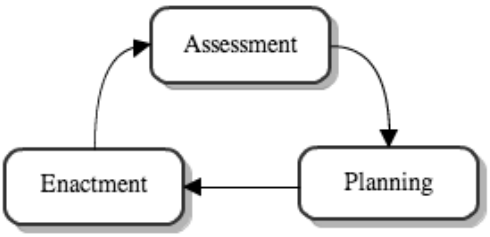

Fig. 1. General Software Process Improvement Model

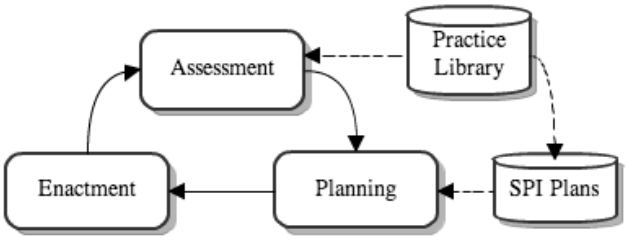

Fig. 2. Prescriptive SPI Model

With supportive SPI frameworks, the challenge lies in determining the applicability and priority of potential SPIs. Software process simulation and modeling (SPSM) has been used successfully to assess the possible impacts of specific SPIs [23], [24], but SPSM and manual approaches assume that the practitioners have the expertise and resources to identify and assess their software process. This may not always be the case in scientific software projects [5].

C. Decision Making

With respect to specific SE process improvement options, SPI assessment and planning tasks can be considered as a series of individual, but related decisions. As such, we can gain some insight into how best to guide scientific software developers in these decisions by considering related work in cognitive decision-making involving experts vs. novices.

Naturalistic Decision Making (NDM) [25] and, specifically, Recognition-Primed Decision Making (RPDM) [26] models describe how decisions are made in the real world and do not prescribe specific thought processes or analytical reasoning

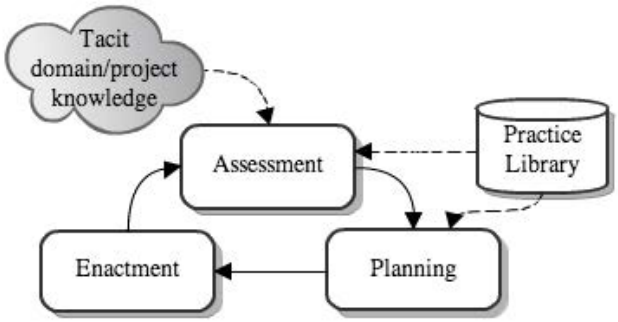

Fig. 3. Supportive SPI Model

steps. Instead they model the intuitive pattern matching and decision making done by experts. NDM models can be used to improve decisions by making subjects aware of their own internal processes. Still, intuitive decision making is limited by the available context and expertise of the subject [25]. A common example of NDM models involves critical decision making as in during firefighter ground commander tasks. In such scenarios, there is little time for exhaustive analytical reasoning [27].

In contrast, prescriptive, heuristic and bias (HB) models are based on the consistent application of a decision model within human limitations in order to predict the expected decision, not just how the decision is made. Errors in decisions are used to gain knowledge about the heuristics and biases at play [28]. HB models are useful in training novices about the analytical processes and criteria leveraged by experts. To leverage HB models effectively, there is an assumption that a "correct" decision (or at least a less wrong decision) can be defined.

As discussed by Kahneman and Klein [29], these two approaches are not mutually exclusive. Intuitive and analytical decision-making can be combined as needed. In fact, the authors posit that situations where expertise and intuition are successful for some decision tasks, but not for others is often the case. The applicability of expertise in decision-making must be considered at the task level, not at the overall domain.

D. Grounded Theory

Formal grounded theory practices [11] suggest (1) that data collection and data analysis should be tied together, (2) that the research process itself is important, and (3) that hypotheses should evolve over time until they are verified, not simply be proposed and then verified or invalidated. In order to support the systematic evolution of theory, data collection and analysis are executed simultaneously to prevent planned analysis from creating bias in deciding which data to collect or discard. Literature reviews, data collection procedures, and experiments are designed and conducted in an organized and traceable manner leveraging consistent analysis procedures. Grounded theory analysis is often referred to as "coding" to reflect the process of "encoding" data into a set of key concepts and relationships.

This approach is especially relevant for the study of scientific software engineering due to the need to form a logical and consistent connection between the core software engineering body of knowledge and the current practices of the scientific community. In order to appropriately leverage grounded theory, a number of risks must be considered. First and foremost, as noted by Adolph *et al.* [30], grounded theory is not a single, specific methodology, but instead, a set of possible approaches all with the same (but commonly misunderstood) foundational goals. In their lessons learned report, the authors discuss how their use and understanding of grounded theory changed over the course of three incremental studies. They began with a very simple interpretation of grounded theory: collect data and incrementally analyze it. It turns out this misses the point of grounded theory. To interview/survey/collect data/etc., incrementally analyze it, and then pose a new set of research questions still focuses on a traditional propose/verify approach.

The goal of grounded theory is to start with a broad research question so as to allow for true growth of the theory in any direction that may arise. Because this can be achieved by applying grounded theory practices to all or just part of a research project, the most important role of a grounded theory approach to software engineering empirical research is to clearly document the philosophy and procedures used in order to avoid assumptions or validity concerns.

TABLE I
SUMMARY OF SPI RELATED WORK

SPI Activity	Mainstream SE	Scientific SE
Assessment	Technical vs. Management Complexity [21] B&T Model [22] CMMI KPAs [19]	Product quality [31] General process characteristics [10]
Prescriptive Planning	CMMI Staged Model [19] CMMI Template Roadmaps [20] Established process models [22]	Checklists [6], [7] Case studies and practice summaries [2], [4] Generalized models [8]
Supportive Planning	CMMI Continuous Model [19] Agile disciplines SPI approach [21]	
Enactment	CMMI KPAs [19] Miscellaneous texts, tutorials, and classes to teach software development and management skills.	Checklists [6], [7] software carpentry boot camps [32]

III. RESEARCH GOALS

Given the need for increased usage of SE practices in scientific research projects, and given that such teams may have limited formal knowledge of such practices, scientific SPI research can proceed in two possible directions:

1) A prescriptive SPI framework that defines the specific practices (and their priorities) needed to achieve a standard level of quality in scientific software projects.
2) A supportive SPI that provides assessment criteria and decision factors relevant to scientific software development projects, but leaves the selection and prioritization of specific improvements to the project team.

In selecting an approach, we must first consider the long-term implications of each direction. While a prescriptive approach provides greater consistency across projects and would help to normalize SE practices in scientific computing, even a customizable prescriptive model (e.g. processes based on Staged CMMI assessments) makes some assumptions about the nature of the projects. In contrast, purely supportive SPI models, such as Kroll and MacIsaac's [21] reference set of agile disciplines and practices, leave much to the discretion of the project team. While this approach provides the flexibility to

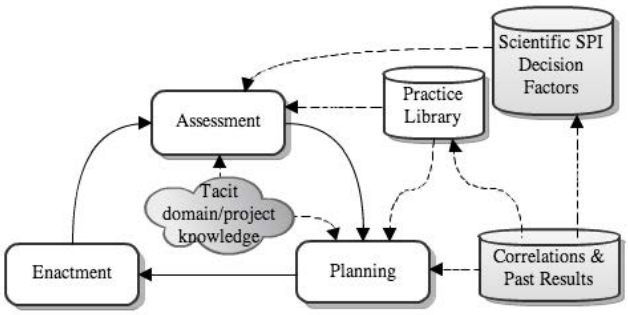

Fig. 4. Scientific SPI Framework (SciSPIF)

support a wide range of scientific software projects, it depends heavily on the teams' abilities to identify or recognize pain points and situations that "resonate" with the agile guidelines.

Although scientific software developers may be novices with respect to SE concepts [5], it may still be possible to leverage their tacit knowledge in SPI decision making. A solely prescriptive or supportive model may not be sufficient, but it is possible that a hybrid decision model will provide the support scientific software development teams require without limiting their ability to "self-drive" their SPI decisions based on their domain expertise.

A. Scientific SPI Framework (SciSPIF)

Rather than follow a prescriptive improvement model (Fig. 2) guided by specific heuristics (e.g. estimated probability of risk), a supportive (i.e. naturalistic or intuitive) model (Fig. 3) would allow for the inclusion of tacit knowledge. However, though they are experts in their own domains, scientists may not have the SE expertise to recognize cues in their own projects that correlate to recommended SE practices.

This "fractured expertise" supports the need for a hybrid decision framework to guide scientists. The combined Scientific SPI Framework (SciSPIF) will supplement scientific expertise with concise descriptions of available SE practices and the problems they traditionally solve (Fig. 4).

B. Research Questions

As discussed in section II and shown in Table I, there is a gap in the related work for supportive scientific software process improvement. While the SciSPIF model provides a general structure to guide scientific SPI, we must consider that existing sources of information about common software project problems and process improvement planning priorities may not apply to scientific software. In order to implement the SciSPIF model as a generalizable framework to support a range of scientific software projects, the following research questions must be addressed:

RQ1 Which mainstream process planning criteria are applicable to scientific software development projects?

RQ2 What other factors influence priorities and decisions during scientific software process planning activities?

RQ3 Which scientific software process planning criteria are applied in decisions about specific SE practices?

IV. Proposed Methodology

Answering the questions posed above will provide the necessary inputs to populate SciSPIF with the factors that impact scientific SPI decisions. In order to ensure the validity of any studies based upon this model, the results must be grounded in established SE principles using rigorous research methodologies that avoid bias and allow for the generation of new and novel theories.

For example, it is reasonable to hypothesize that the decision to use source control management practices will correlate to the size of the research project; however, this is only one of many possible hypotheses. Formally defining and testing each hypothesis would be inefficient and may overlook factors that apply in scientific domains but that, as software engineers, we cannot foresee. A more effective methodology is to discover these relationships via a broader, inductive study that leverages past work and newly collected data to answer the research questions. To this end, this research will leverage a grounded theory approach to ensure that each research construct and conclusion is clearly derived from its predecessors (Fig. 5).

A. Step 1 - Candidate Criteria and Correlations

Carver et al. [4] and Sletholt et al. [2] have identified a number of key characteristics and practices of scientific software projects by analyzing publicly available projects. Since case studies are representative of the scientific software domain in general, they offer valuable insight into possible drivers of scientific SPI decisions. In order to generate an initial set of scientific SPI planning criteria (RQ2-3), these source studies (and any summaries available) will be revisited and analyzed using an iterative coding approach (Fig. 5, dotted line). At the same time, analysis of mainstream SPI frameworks from the perspective of scientific software development will provide a foundational set of SPI planning criteria to be considered (RQ1).

1) Step 1a: Open Coding: The open coding process entails reviewing the available data and marking any information relevant to the research questions. At this stage, the investigator also creates notes (a.k.a. "codes") to denote key concepts (a.k.a. "nodes") and the relationships between them. Nodes may also be created based on the frequency of certain words or concepts in a data source via the generation of word clouds as shown in Fig. 6. The example shown in Fig. 7 includes codes created to represent possible defining characteristics of a scientific software project ("data size" and "rate of change of domain"). The data source for this example is an experience report regarding the addition of agile practices to a bioinformatics project [9].

The resulting nodes are shown in Fig. 8 (in addition to nodes created when coding the Boehm and Turner dimensions for process planning [22] and Kelly's analysis of scientific software [10]). It is important to note that node names do not necessarily have to be taken directly from the source text. For example, the "data size" code represents an interpretation of the implication that the large number of human proteins has for

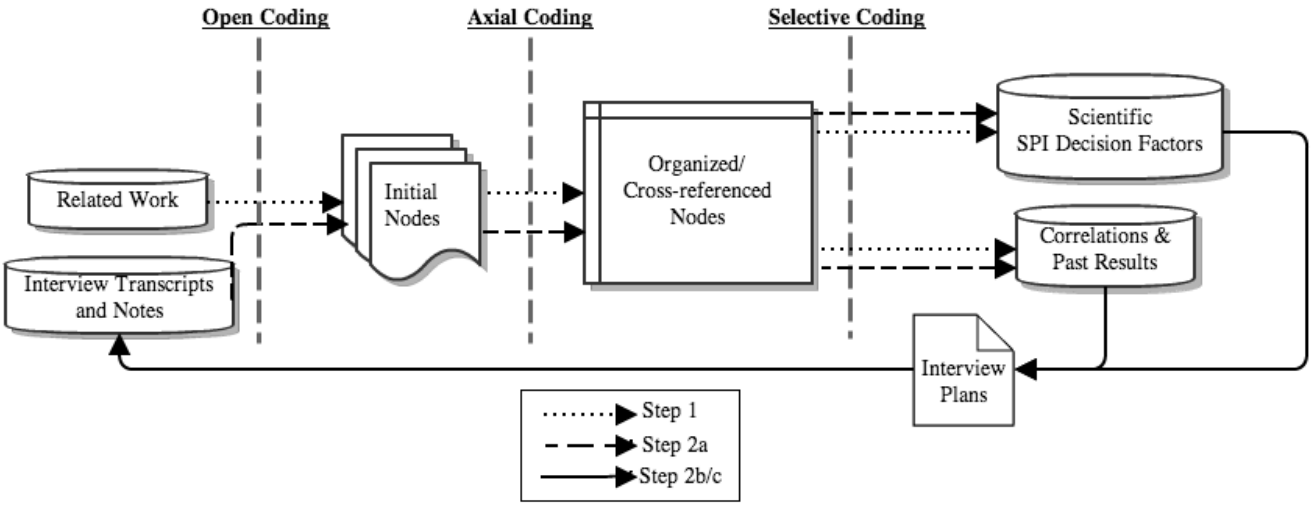

Fig. 5. Overview of Proposed Methodology

Fig. 6. Word cloud for a case study paper [9]

software requirements. This is an example of a constructivist view of grounded theory [33].

2) Step 1b: Axial Coding: The axial coding process leverages the set of open codes from step 1a and seeks overlap and patterns across data sources. For example, the "rate of change of domain" node from Fig. 8 would clearly overlap with the "dynamism" aspect of the Boehm and Turner (B&T) model of process model selection [22]. This result is supported by the analysis of scientific software against the B&T model conducted by Kelly [10] (Fig. 9).

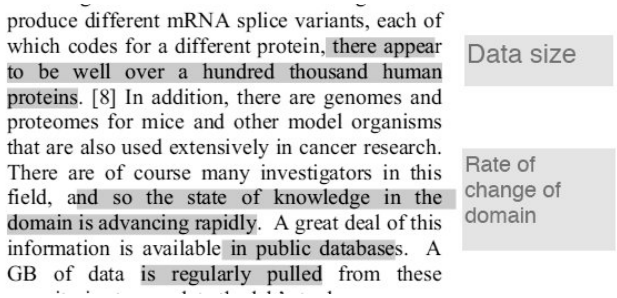

Fig. 7. Example of open coding of a case study paper [9]

3) Step 1c: Selective Coding: The open and axial coding processes are designed to gather all possible related information, not the most relevant. From the cross-referenced set of nodes generated in step 1b, we are left with a large set of candidate criteria and SPI correlations. Selective coding provides a systematic way to review the generated nodes for patterns and frequency and reduce them to a core set most likely to be relevant for the entire population.

As the investigators are all trained software engineers who are likely to emphasize known SE factors, there is some risk of confirmation bias if only manual coding procedures are used. In order to mitigate the risk of discounting or overvaluing some nodes, NVivo 10's [34] ability to search and summarize references to nodes will be leveraged. An example of this capability is shown in Fig. 10.

B. Step 2: Elaboration

While coding of related work provides a strong base for an initial implementation of SciSPIF, the results would be limited to the data collected by the original investigators. New data collected via consistent procedures across multiple representative projects is required to fully explore the candidate criteria and allow for the generation of new theories. The most

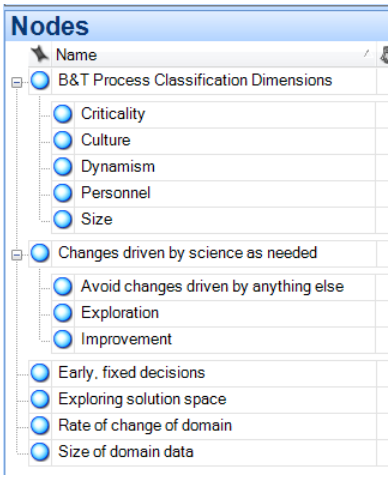

Fig. 8. Nodes created via initial open coding

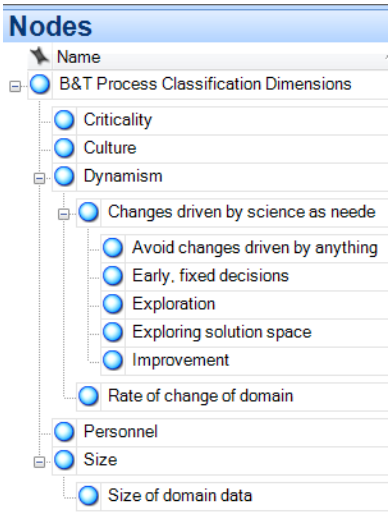

Fig. 9. Nodes reorganized by relationship to each other

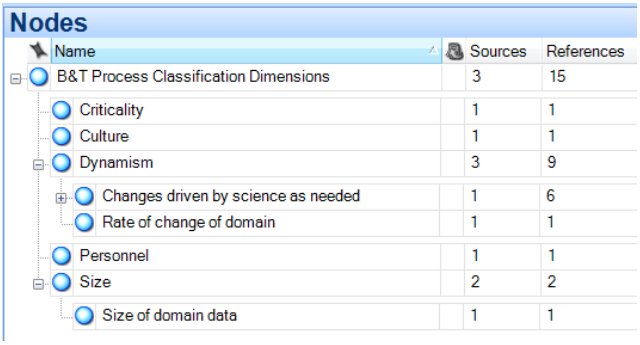

Fig. 10. Reference counts for use in selective coding

effective way to gather this data is via real-time interviews with active scientific software development teams.

1) Step 2a: Interviews: Interviews can be conducted in one of two primary scenarios: structured or unstructured. Structured interviews have the benefit of providing consistency across interview subjects, but have a high risk of missing potentially valuable information that can come from "asides". If too structured, an interview risks becoming a live survey rather than a qualitative data gathering exercise. Unstructured interviews allow for an open and conversational dialog between the subject and researcher, but they risk going off-topic and becoming grossly inefficient.

For the work proposed in this paper, the cost of missing unforeseen insights is considered greater than any potential inefficiency. Thus, a "semi-structured" approach will be undertaken. For each interview, a number of constraints will be in place:

1) The interviewer will conduct relevant background research about the nature of the project and arrive at the scheduled interview with a general set of topics he/she wishes to cover.
2) Interview topics will be guided by known project information and the candidate criteria theorized in Step 1 (Fig. 5, solid line).
3) If part of the interview is to include a review of project artifacts of any kind, this will be coordinated in advance with the subject.
4) The researcher will take pen and ink notes during the interview, but will otherwise not disrupt the flow of the conversation with detailed record keeping.
5) The interview will be recorded for later transcription.
6) Each transcription will be supplemented with the pen and ink notes taken by the researcher. This procedure will help to ensure consistency throughout the course of the research project. In addition, interview results can be easily revisited for analysis from other perspectives.

2) Step 2b: Process Interviews: The interview transcripts, notes, and any artifacts provided by the subjects will be processed using the same open, axial, and selective coding procedures described in step 1. Additionally, the new coding activities will build upon the results of step 1, not ignore or supplant them (Fig. 5, dashed line).

3) Step 2c: Interview Follow-ups: Following the completed processing of each interview, the results will be reviewed with the interview subject. The primary motivation for this step is to minimize the impact of investigator bias on the results. While some level of interpretation is desired, it is important to review the conclusions with the subjects to ensure that the subjects concur.

Additionally, follow-up meetings and interviews establish two-way communication between the investigators and subjects. In discussing the results and conclusions drawn, it is possible that the subjects will consider their own work in a new light and think of further information to share. This process will also be recorded as per the interview procedures

and coded to update the original data collected with anything new that arises from the follow-up discussions.

V. Current Status

As shown via the sample results in section IV, the open, axial, and selective coding procedures for related work have been piloted and refined via application to specific data sources. Enactment of step 1 on a broader scale is under way. Additionally, a pilot interview to test transcription and coding procedures was conducted in order to refine the methodology described in step 2. Formal human subject research approval is pending. Interviews will likely begin in late spring, 2013, and will continue throughout the remainder of the year.

VI. Future Work

A. Initial Implementation

In order to make the results of this work accessible to the scientific community, SciSPIF will be implemented as an online decision aide. The initial version will likely be a simple website that supports exploration of the identified decision criteria and references to existing supplemental materials such as the CMMI specification [19], Kroll and MacIsaac's list of agile practices [21], published work regarding scientific computing, and scientific computing training materials (e.g. software-carpentry.org [32]).

B. Generalization

As Carver et al. [5] have shown, there is limited knowledge of many core SE practices in the scientific community. While quality software is valued and many practices are in use, we cannot assume consistent use of SE terminology. Creating "plain English" supplemental materials for SPI concepts and linking these to more detailed materials is a high priority.

C. Ongoing Data Collection

In order to validate the model itself and allow it to evolve, a feedback loop is required. Rather than only disseminate information gained via investigator driven research, an interactive implementation of SciSPIF will allow scientists to provide feedback about the usefulness of the information provided to them. Iteration through the SPI process using SciSPIF creates data regarding why specific practices were chosen and if/how they contributed to the project quality. For example, relevant post-SPI planning activity questions could be:

1) Which SciSPIF decision criteria were most relevant for your project?
2) For each criterion, how useful did you find the correlations suggested by SciSPIF?
3) Do you have project concerns you feel were unaddressed by SciSPIF?

In order to pilot this feedback loop, the next step of this research is to enact small surveys and case studies asking scientists to leverage the model for a limited set of features. In addition to piloting the model and providing valuable long-term data, such studies will allow a deeper understanding of the motivating factors behind specific scientific SPI decisions (e.g. "What is considered when deciding if/how to use practice ABC?").

Long term, the feedback loop may be implemented directly into the public model to allow constant and ongoing data collection in addition to project process tracking capabilities for scientific research teams.

VII. Final Thoughts

There is a wealth of information regarding the state of SE practices in scientific domains, SE process improvement in traditional domains, and understanding decision making processes in general. The SPI framework proposed in this paper will provide an immediate aide for scientific software teams and a framework for the collection of data regarding scientific SPI. Long-term, this data can then be used to generate and test new hypotheses regarding SE process improvement for scientists.

In addition to discovering useful correlations and SPI lessons learned for the scientific community, the most important aspect of this research lies in establishing protocols and a base of strong empirical data for future work. 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 may assist other software engineering process researchers.

Anyone interested in learning more or participating in these efforts (as an interview subject or investigator) is welcomed and encouraged to contact Erika S. Mesh directly at esm1884@rit.edu.

References

[1] G. Miller, "A scientists nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, 2006.

[2] M. Sletholt and J. Hannay, "What do we know about scientific software development's agile practices?" *Computing in Science & Engineering*, vol. 14, no. 2, pp. 24–37, 2012.

[3] D. Kelly, "A software chasm: Software engineering and scientific computing," *Software, IEEE*, vol. 24, no. 6, pp. 120–119, 2007.

[4] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software development environments for scientific and engineering software: A series of case studies," in *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, 2007, pp. 550–559.

[5] J. Carver, D. Heaton, L. Hochstein, and R. Bartlett, "Self-perceptions about software engineering: A survey of scientists and engineers," *Computing in Science & Engineering*, vol. 15, no. 1, pp. 7–11, 2013.

[6] D. Kelly, D. Hook, and R. Sanders, "Five recommended practices for computational scientists who write software," *Computing in Science & Engineering*, vol. 11, no. 5, pp. 48–53, 2009.

[7] M. A. Heroux and J. M. Willenbring, "Barely sufficient software engineering: 10 practices to improve your cse software," in *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, May 2009, pp. 15–21.

[8] J. Segal, "Models of scientific software development," in *SECSE 08, First International Workshop on Software Engineering in Computational Science and Engineering*, Leipzig, Germany, 2008.

[9] D. Kane, "Introducing agile development into bioinformatics: an experience report," in *Agile Development Conference, 2003. ADC 2003. Proceedings of the*, 2003, pp. 132–139.

[10] D. Kelly, S. Thorsteinson, and D. Hook, "Scientific software testing: Analysis with four dimensions," *Software, IEEE*, vol. 28, no. 3, pp. 84–90, May 2011.

[11] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[12] About computing in science & engineering magazine. [Online]. Available: http://www.computer.org/portal/web/computingnow/cise/about

[13] L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," New York, New York, USA, 2010, p. 1.

[14] Y. Li, "Reengineering a scientific software and lessons learned," in *Proceedings of the 4th international workshop on Software engineering for computational science and engineering*, 2011, pp. 41–45.

[15] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, May 2009, pp. 1–8.

[16] L. Osterweil, "Software processes are software too , revisited: An invited talk on the most influential paper of icse 9," in *Proceedings of the 19th International conference on Software Engineering*, 1997, pp. 540–548.

[17] F. Pettersson, M. Ivarsson, T. Gorschek, and P. Öhman, "A practitioner's guide to light weight software process assessment and improvement planning," *Journal of Systems and Software*, vol. 81, no. 6, pp. 972–995, Jun. 2008.

[18] W. Humphrey, "Characterizing the software process: a maturity framework," *Software, IEEE*, vol. 5, no. 2, pp. 73–79, 1988.

[19] Software Engineering Institute, "CMMI for development, version 1.3," Carnegie Mellon University, Tech. Rep. CMU/SEI-2010-TR-033, 2010.

[20] J. Cannegieter, A. Heijstek, B. Linders, and R. van Solingen, "CMMI roadmaps," Caregie Mellon University, Tech. Rep. CMU/SEI-2008-TN-010, 2008.

[21] P. Kroll and B. MacIsaac, *Agility and Discipline Made Easy: Practices from OpenUP and RUP*. Pearson Education, Inc., 2006.

[22] B. Boehm and R. Turner, *Balancing agility and discipline: A guide for the perplexed*. Reading, MA: Addison-Wesley Professional, 2004.

[23] D. Raffo and W. Wakeland, "Moving up the CMMI capability and maturity levels using simulation," Carnegie Mellon University, Tech. Rep. CMU/SEI-2006-TR-028, 2007.

[24] D. Raffo, U. Nayak, and S.-o. Setamanit, "Using software process simulation to assess the impact of iv&v activities," in *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004)*, Edinburgh, 2004.

[25] R. Lipshitz, G. Klein, J. Orasanu, and E. E. Salas, "Taking stock of naturalistic decision making," *Journal of Behavioral Decision Making*, vol. 14, no. 5, pp. 331–352, Dec. 2001.

[26] G. Klein, "Recognition-primed decisions," in *Advances in Man-Machine Systems Research*, vol. 5, 1989, pp. 47–92.

[27] G. Klein and A. Clinton-cirocco, "Rapid decision making on the fire ground : The original study plus a postscript," *Journal of Cognitive Engineering and Decision Making*, vol. 4, no. 3, pp. 186–209, 2010.

[28] D. Kahneman, "A perspective on judgment and choice: mapping bounded rationality." *The American psychologist*, vol. 58, no. 9, pp. 697–720, Sep. 2003.

[29] D. Kahneman and G. Klein, "Conditions for intuitive expertise: a failure to disagree." *The American Psychologist*, vol. 64, no. 6, pp. 515–26, Sep. 2009.

[30] S. Adolph, W. Hall, and P. Kruchten, "A methodological leg to stand on : Lessons learned using grounded theory to study software development," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, pp. 1–13.

[31] D. Kelly, R. Gray, and Y. Shao, "Examining random and designed tests to detect code mistakes in scientific software," *Journal of Computational Science*, vol. 2, no. 1, pp. 47–56, Mar. 2011.

[32] software-carpentry.org. [Online]. Available: http://software-carpentry.org/

[33] K. Charmaz, *Constructing grounded theory: A practical guide through qualitative analysis*. London: SAGE, 2006.

[34] Nvivo 10 features and benefits. [Online]. Available: http://www.qsrinternational.com/products_nvivo_features-and-benefits.aspx