A Case Study on a Quality Assurance Process for a Scientific Framework

Hanna Remmel and Barbara Paech | University of Heidelberg
Christian Engwer | University of Münster
Peter Bastian | University of Heidelberg

The following case study analyzes the feasibility and acceptance by developers of two parts of the quality assurance process: variability model creation and desk-checking. The study found that both parts are accepted and feasible for development, but a different variability modeling language is needed to represent all important aspects.

Our research concentrates on the quality assurance of scientific frameworks. The Distributed and Unified Numerics Environment (Dune; www.dune-project.org), the software we use, is a complex scientific framework for solving partial differential equations supporting a large variety of applications (for example, fluid mechanics or transport in porous media), mathematical models, and numerical algorithms.

Testing scientific software is a challenging task, because it must deal with special challenges such as missing test oracles, the need for high-performance parallel computing, and the high priority of nonfunctional over functional requirements.[1] When testing a scientific framework, we also need to find a way to deal with the large variability, which means the various possible uses, of a framework.

Our approach to meeting this challenge is to apply software product line engineering (SPLE) to handle the framework's variability. In SPLE, the idea is to develop a software platform during domain engineering and then, in application engineering, to use mass customization for the creation of a group of similar applications that differ from each other in specific predetermined characteristics.[2]

In earlier work, we proposed the design of a quality assurance process for scientific frameworks.[3] This process is based on a quality assurance process for SPL introduced by Ivan do Carmo Machado and his colleagues in the context of the RiSE Product Line Engineering Testing project (Riple-TE).[4] (See the "Related Work in Quality Assurance" for other work in this area.) We adjusted this quality assurance process to account for an SPL test strategy for scientific frameworks and special characteristics for scientific software development, which we introduced earlier.[3] The main adjustments in our quality assurance process regarding Riple-TE were that our process only covers domain testing, introduces system testing already in domain testing, includes scientific validation, and reduces the number of test roles.

Here, we report on a case study based on the method described by Per Runeson and his colleagues,[5] analyzing those parts of the design of the quality assurance process that aren't yet familiar to the Dune developers: variability model creation and desk-checking. The use of variability modeling for the systematic creation of system tests is a new method in scientific software development, which makes case study results interesting for the computational science and engineering community in general. Desk-checking, on the other hand, is a relatively well-known technique and has already been mentioned in scientific software engineering literature,[6] but no experimental results on the feasibility and acceptability of the method in the context of scientific software were available at the time of this study. The objective is to analyze the feasibility and the acceptance of these methods. Based on the results of the case study, we'll want to adjust

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

the design of the quality assurance process before establishing it completely.

Case Study Background

In previous work, we discussed a process for creating variability models for Dune.[7] We use SPL variability modeling for the systematic creation of system tests. Because it isn't feasible to create such a variability model for the whole framework covering a wide range of functionality, we start with the mathematical requirements for the framework (the mathematical problems, which the framework should solve) and create several variability models based on those mathematical requirements. Each variability model is associated with a system test application, a Dune application solving the corresponding mathematical problem. For example, one mathematical problem the Dune framework should support is solving the Poisson equation, an elliptic partial differential equation. A variability model for this problem covers, among other things, the different possible grid configurations and the discretization methods used.[7]

We propose using the orthogonal variability modeling language of Klaus Pohl and his colleagues.[2] Software characteristics that can vary are called *variation points* and a variation point's possible values are called *variants*. A *variability model* is described by variation points and their variants. This model also includes the constraints between the variation points and the variants.[2] Figure 1 shows an example variability model including the graphical notation used.

Figure 2 illustrates the design of the quality assurance process. Highlighted are those parts of the design that are necessary to understand the background of the case study: planning, including the creation of variability models, and review (desk-check). Only these parts are discussed here. The entire process is discussed in detail elsewhere.[3]

Planning

The activities in this step are critical for the success of the whole process. When developers change the source code or develop a new piece of code, they plan ahead for quality assurance issues such as creating or adjusting unit test cases.

If the mathematical requirements for the framework change (for example, when a new functionality is included in the framework), the developer might need to formulate one or more new variability models based on the requirements together with the associated system test applications. In other cases, existing variability models and system test applications must be adjusted (for example, by including new variation points or variants).

Review

The sooner a failure is found, the lower the cost of removing it. The earliest possible point for finding failures is right after developing the code. Taking the time to consciously read one's own code before checking it in (also called desk-checking[6]) can reveal failures before the code is even tested. At the same time, the developer can review the code's readability and structure. Because the software context is complex, developers should strive to write comprehensible source code with a sufficient

Related Work in Quality Assurance

Here, we consider other empirical studies on quality assurance processes for software product line engineering (SPLE). We couldn't find any empirical studies for quality assurance processes for scientific software in the literature.

The unit testing part of Riple-TE, the quality assurance process we based our process on, was initially evaluated in an experimental study by Ivan do Carmo Machado and his colleagues.[1] The goal was to analyze the effectiveness of unit testing in this process and to determine which professional skills impact the test activity results. In the experiment, 30 undergraduate students tested the same source code with and without the process. The authors admitted that the results of this experiment weren't very significant. This initial experiment serves as a baseline for future experiments.

Paulo Anselmo da Mota Silveira Neto and his colleagues propose a formal regression testing approach for the reference architecture of an SPL, which uses extensive documentation, many detailed process steps, and plenty of test roles.[2] Their approach concentrates on the commonality of the SPL and doesn't apply to system testing. The approach was evaluated to calibrate and improve it. Eight postgraduate students applied the approach in an experimental scenario. The approach showed its efficiency, although it wasn't tested in a real SPL context.

One major advantage in our case study was that we could conduct the case study with developers of scientific software who are the actual target audience instead of undergraduate or postgraduate students, making the results more significant.

References

1. I.C. Machado et al., "RiPLE-TE: A Process for Testing Software Product Lines," *Proc. 23rd Int'l Conf. Software Eng. and Knowledge Eng.*, 2011, pp. 711–716.
2. P.A. da Mota Silveira Neto et al., "A Regression Testing Approach for Software Product Lines Architectures," *Proc. 4th Brazilian Symp. Software Components, Architectures and Reuse*, 2010, pp. 41–50.

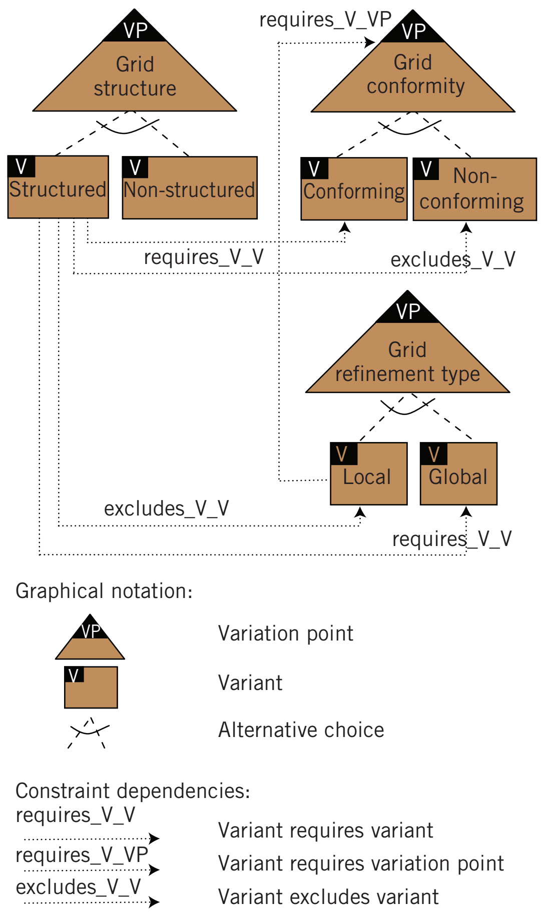

Figure 1. Example of the use of an orthogonal variability modeling language for supporting the systematic creation of system tests.

number of comments. This would also benefit new colleagues working with the same software, and it improves the code's maintainability.

In contrast to the technical review in the RipleTE quality assurance process, our process involves a review of the source code, not just SPL artifacts like the variability models. Certainly, developers should review all artifacts they created or changed: source code, unit tests, variability models, and system test applications.

If appropriate, the developer can ask a colleague to review the changes as well. The development team could also name developers responsible for different software modules who regularly review the changed source code.[6] We don't pursue a structured inspection or review process, as the goal is to keep the quality assurance process practical and simple.

Case Study Design

The case study's objective is to analyze the feasibility and acceptance of variability model creation and desk-checking as part of the designed quality assurance process. This objective was chosen as these aspects hadn't yet been familiar to the Dune developers. The advantage of analyzing the process before it's completely established is that we can still adjust the design according to the case study results without much overhead.

The case study was executed with a group of six Dune developers. They all work in the same academic group and have developed Dune for 2.5–3.5 years, with one developer having worked in the group for 10 years. Four of the developers are mathematicians and two are computer scientists.

Research Questions

According to the goal question metric approach (GCM),[8] to measure in a purposeful way, we first need to specify our goals and then define how we intend to collect and interpret data with respect to the stated goals. The following are the goals for the case study:

- Goal 1: Assess the feasibility of variability modeling.
- Goal 2: Assess the feasibility of desk-checking.
- Goal 3: Assess the acceptance of variability modeling.
- Goal 4: Assess the acceptance of desk-checking.

All of these goals were analyzed in the quality assurance context from the developer's viewpoint.

Feasibility. For the feasibility assessment, we reflected on possible advantages and disadvantages of variability modeling and desk-checking. We then formulated research questions that would check whether these assumptions apply. In addition, we wanted to find out if the Dune developers think that the advantages of these methods outbalance the effort required.

Acceptance. The acceptance part of our case study is based on the technology acceptance model (TAM).[9] This method was developed for software systems, but we use it for software engineering methods. Fred Davis and his colleagues found that during a one-hour hands-on introduction, people form a perception of a system's (method's) usefulness that is strongly linked to their usage intention.[9] Furthermore, the intention of use is significantly correlated with the future acceptance of the system (method). According to TAM, perceived usefulness and ease of use are of primary relevance for acceptance behavior.

Table 1 contains our research questions, the associated hypotheses, and the sources used to collect the data.

Research Methods

We designed and conducted the case study according to instructions by Runeson and his colleagues.[5]

The main author of this article took part in and moderated a two-hour meeting with the Dune developers. An external researcher also attended the meeting for validity reasons.

The Dune developers performed the given tasks together in a group. The researcher didn't take part in the tasks, but questions of comprehension to the researcher were allowed. The case study meeting was recorded, transcribed, and coded for analysis purposes.

Task 1 was to model different possibilities of how a grid can be defined.[7] The proposed approach was to first list possible variation points on a flip chart and then continue with the variability modeling on a poster board.

Task 2 was to adjust a proposed desk-checking checklist for the needs of Dune development. Which items are suitable and which are not? Which items are missing? The developers didn't try out desk-checking directly because it wasn't possible to simulate a realistic application of desk-checking. The "Proposed Desk-Checking Checklist" sidebar shows the checklist the Dune developers were given.

We used different data sources for the case study: observation, questionnaire, and discussion. The metrics used for observation and discussion were subjective notes by the researcher. For the open questionnaire questions, we used the Dune developers' subjective opinion metric. For the data aggregation, we counted how many times similar answers were given. For the closed questionnaire questions, we used a Likert scale and aggregated the answers with a median. If the median was between two values, we chose the side with the higher dispersal in the answers.

Results

We report the results of the case study using the research questions in Table 1.

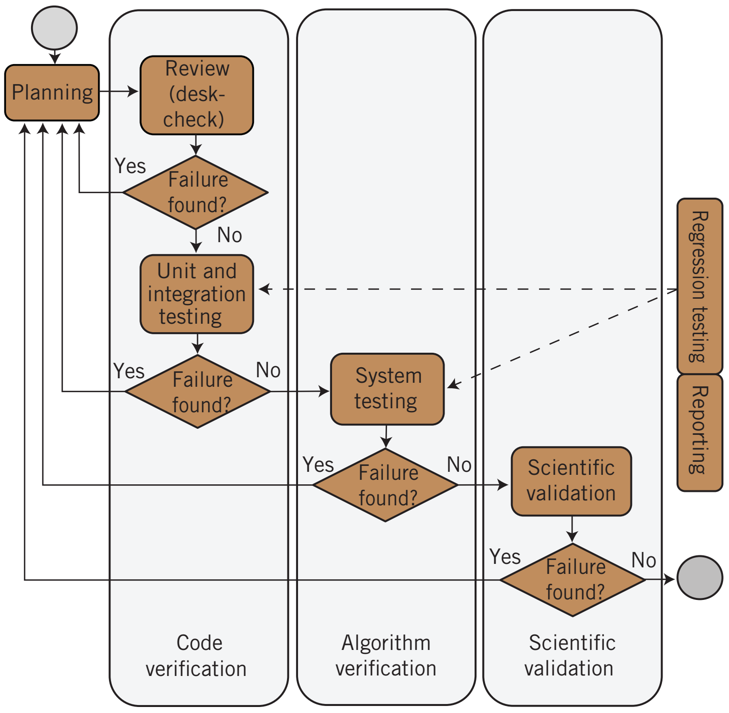

Figure 2. Quality assurance process for scientific frameworks. The parts needed to understand this case study, planning and review (desk-check), are highlighted. The whole process is introduced in detail elsewhere.[3]

Variability Modeling by Developers

When the Dune developers were working on their variability model, the working atmosphere was open and everyone took part in the discussion about variability model details (F_RQ_VM1). The developers were motivated to learn the method and many questions of comprehension were asked.

First, the developers collected possible variation points on a flip chart. Every proposal was thoroughly discussed right away and at the end accepted or rejected by the group. Possible variants were listed instantly for each variation point. Before drawing the variability model, the developers considered possible dependencies between the variation points and their variants. They wanted to draw the variation points with their dependencies close to each other.

The developers thought of the model with "levels" or a "hierarchy," although the proposed variability model doesn't have levels. They considered which variation point should be put on the top, meaning which variation point is most essential for the variability model. They wanted to put variation points that are built in a similar way "on the same

Table 1. Research questions.

Research question	Coding*	Hypothesis	Data source
How do developers perform variability modeling?	F_RQ_VM1	N/A	Observation
What do the developers believe are the advantages of variability modeling for the Dune development?	F_RQ_VM2	Variability modeling offers a systematic way to model different possibilities to solve a mathematical problem and a support for system test program development.	Observation, open and closed questionnaire questions
What do the developers believe are the disadvantages of variability modeling for the Dune development?	F_RQ_VM3	Variability model creation is a complex task and requires deep domain knowledge.	Observation, open and closed questionnaire questions
Can variability modeling be used to capture the variability of mathematical problems solved by the framework?	F_RQ_VM4	Yes, variability modeling can be used to capture the variability of mathematical problems.	Observation
How do developers perform desk-checking?	F_RQ_DC1	N/A	Observation
What do the developers believe are the advantages of desk-check for the Dune development?	F_RQ_DC2	Desk-checking helps developers find software failures even before testing, provides a reminder of creating tests and documentation, and leads to an increase in software quality, in particular readability and maintainability.	Observation, open and closed questionnaire questions
What do the developers believe are the disadvantages of desk-check for the Dune development?	F_RQ_DC3	Desk-checking leads to minor overhead.	Observation, open and closed questionnaire questions
Do the advantages of a variability model outbalance the effort of creating it?	E_RQ_VM	The advantages of a variability model outbalance the effort of creating it.	Closed questionnaire questions
Do the advantages of desk-checking outbalance the effort needed for it?	E_RQ_DC	The advantages of a desk-checking outbalance the effort needed for it.	Closed questionnaire questions
Do the developers think variability modeling/desk-checking is useful for them?	A_RQ_VM/DC1	The developers find variability modeling/desk-checking useful for them.	Closed questionnaire questions
Do the developers think variability modeling/desk-checking is easy to use?	A_RQ_VM/DC2	The developers think variability modeling/desk-checking is easy to use.	Closed questionnaire questions
Do the developers intend to use variability modeling/desk-checking in Dune development?	A_RQ_VM/DC3	The developers intend to use variability modeling/desk-checking in Dune development.	Closed questionnaire questions

*Legend for research question codes: F = feasibility, E = effort, A = acceptance, RQ = research question, VM = variability modeling, DC = desk-checking.

level." This seems to be an intuitive way of thinking of a variability model.

In Dune, there's a technical constraint when it comes to defining a grid: there are a handful of grid implementations a Dune user can choose from when implementing a Dune application. Each grid implementation has its own characteristics and constraints. It's only possible to use a grid with characteristics that suit at least one of these grid implementations. The used grid implementation is actually not a grid characteristic, but technically highly essential. Thus, the developers decided to select grid implementation as a central variation point.

Possible variants for the variation point “element type” depend on the variation point “grid dimension.” In 2D, possible element types are cubes and simplexes. In 3D, there are also prism, pyramid, and so on, because sometimes a grid can’t be defined correctly without these filling element types. The developers decided to use an “abstraction” for the variation point element type: they defined only the variants “cube(d)” and “simplex(d)” for it. Depending on the grid dimension, it’s clear which concrete element types are available.

Advantages of Variability Modeling for the Dune Development

The following list collects the most frequently mentioned advantages that Dune developers see in variability modeling (F_RQ_VM2 and E_RQ_VM). The numbers in parentheses indicate how many times each statement was mentioned in the open questionnaire questions about the advantages:

- Variability modeling offers a systematic way to cope with all different possible combinations of features and their dependencies (5).
- The process of variability modeling leads to a deeper reflection about the set of needed variants, concrete dependencies between the concepts in the software, or scope and goal of a test case (5).
- Variability modeling is the first step in the automatic generation of test cases for Dune: every valid combination of variants is a test case (2).

The summarized results of closed questionnaire questions reveal that the developers agree that variability modeling would be helpful in developing system test applications, and that the advantages of a variability model outbalance the effort of creating it.

Disadvantages of Variability Modeling for the Dune Development

The disadvantages some developers see in variability modeling (F_RQ_VM3) in general include the following:

- Creating a variability modeling is costly because of the complexity of the mathematical problems (1).
- It will be difficult to implement the automatic creation of test cases, because each set of variants must be implemented differently (1).

These are the disadvantages the developers see in the proposed variability model:

- The presentation becomes complex easily and therefore unclear, unreadable, and hard to maintain. Many lines (dependencies) make the model unclear (5).
- The developers miss the possibility to define a hierarchy between variation points (4).
- The modeling language can’t represent some important aspects. Some dependencies are more complex than can be modeled. For example, in some cases one variant should be excluded if a combination of two other variants is chosen. To model this kind of situation, the developers combined two variation points—grid implementation and grid dimension. This solution didn’t satisfy the developers (3).

Capturing the Variability of Mathematical Problems with Variability Modeling

While working on the variability model, the developers didn’t agree about how detailed the model should be (F_RQ_VM4). Some developers repeatedly came up with special cases and other developers argued that these cases weren’t really relevant. One developer noted that the variability model only needs to be as detailed as someone wants to define the different test cases for a system test application. Some minor features could be implemented as arbitrary parameters without being part of the variability model.

Proposed Desk-Checking Checklist

Before checking in, please read through the source code one more time:

- Is the desired functionality or change implemented correctly?
- Is the source code sufficiently documented?
- Does the source code follow the coding style?
- Were unit tests created for new functionality? Were existing unit tests adjusted for the changed source code?
- If new mathematical requirements were implemented, were system tests created?

All developers agreed that the viewpoint has a major influence on the created variability model. A variability model created for a specific system test application will be different from a variability model for a general case. Many developers agreed that it makes sense to choose a "test application" viewpoint, because the variability model would be more precise and less complex.

Acceptance of Variability Modeling

The developers agreed that variability modeling is useful for the Dune development, easy to learn and use, and that they intend to use variability modeling for the Dune development (A_RQ_VM1-3).

Desk-Checking by Developers

The developers found several items important in the proposed checklist (F_RQ_DS1):

- The developers distinguished between source code documentation and commit messages in the version control system. They thought both of these were important. The commit messages were distributed over a mailing list so other developers could review the changes.
- Several developers argued for the importance of creating and extending suitable tests, in particular for changes in the Dune base classes.
- A couple of developers thought that checking whether the source code follows the coding style is also important.

In addition, one developer noted that the items in the checklist are rather suitable for the development in the Dune base classes instead of source code for solutions of special mathematical problems.

There were also some ideas for additional checklist items:

- Several developers found it important to check the naming of variables for suitability before checking in the source code.
- Another point mentioned was that each developer should check that his or her source code is written comprehensibly.

The item in the checklist that most developers found redundant was whether the functionality was correctly implemented. They found that it's self-evident that only source code that works will be checked in. They didn't see any advantage in reading the source code one more time to review this. Rather, they check the functionality through testing or by looking at the output of the software.

Advantages of Desk-Checking for the Dune Development

The main advantage the developers see in desk-checking (F_RQ_DC2 and E_RQ_DC) is quality improvement, in particular of the documentation (5). Other quality improvements they expect are:

- a more careful check that an implementation is correct (2);
- a better chance that proper tests are developed (2); and
- more readable code that follows the coding style (2).

In the closed questionnaire questions, the developers answered that they agree that desk-checking helps to develop higher quality, more maintainable, and more readable source code, and that the advantages of desk-checking outbalance the effort required. They rather agree that desk-checking leads to a higher detection rate of software failures.

Disadvantages of Desk-Checking for the Dune Development

The disadvantages that the developers saw in desk-checking (F_RQ_DC3) include the following:

- There's an overhead before checking in source code, mainly because of the creation of new tests (4).
- Most of the items in the checklist are subjective. Developers have their own opinions on, for example, what is "sufficiently documented." Minimum requirements must be defined for each issue (2).

Acceptance of Desk-Checking

The developers agreed that desk-checking is useful for the Dune development and easy to learn, and they intend to use it for the Dune development (A_RQ_DC1-3). They clearly find desk-checking acceptable, although they only agree that it's easy to learn or personally important to them.

Support for Stated Goals

The results of our case study support the goals outlined earlier.

Goal 1: Feasibility of Variability Modeling

Dune developers recognized important advantages of variability modeling. A surprising result was that almost every developer brought up the positive effect of a deeper reflection about the variability in the examined concept. The disadvantages found were almost all associated with the presented variability modeling language, not with variability modeling in general.

The results of the case study reveal that variability modeling can be used to capture the variability of mathematical problems if the viewpoint is fixed first and the modeling task is clearly defined. This means that variability modeling is feasible for the Dune quality assurance, but we need to find a different variability modeling language that can represent all important aspects and enables the definition of a hierarchy.

Goal 2: Feasibility of Desk-Checking

The case study convinced the developers that desk-checking helps to develop higher quality source code. They could see many advantages in desk-checking. The main disadvantage they see—overhead—is mainly associated with the creation of tests. However, they're willing to accept this overhead, as they see it as an advantage that desk-checking reminds them of creating the tests.

The case study results indicate that desk-checking is feasible for the Dune development. As a next step, the developers should adjust the desk-checking checklist to better suit their needs and define the minimum requirements for each item in the checklist.

Goal 3: Acceptance of Variability Modeling

The acceptance of variability modeling for the Dune development was positive. We expect acceptance to increase with a more suitable variability modeling language.

Goal 4: Acceptance of Desk-Checking

The case study results in a clear acceptance of desk-checking. The developers see that desk-checking is useful for the Dune development and intend to use it.

Threats to Validity

In our analysis of the validity of the case study and its results, we distinguish between different aspects of the validity as presented by Runeson and his colleagues.[5]

Construct Validity

Construct validity reflects the extent to which the used research methods really represent what's investigated according to the research questions.[5]

We used different methods to increase the construct validity of our case study. To achieve a methodological triangulation, we combined different types of data collection methods: observation, questionnaire, and discussion. The case study design includes a chain of evidence on how the data of the different data sources are used to answer the research questions. Observer triangulation was achieved by an external observer during the case study meeting.

Several researchers checked the questions in the questionnaire for understandability. During the case study, external influence on the developers was kept to a minimum. The moderator didn't mention any advantages or disadvantages of the methods. The developers always answered the open questions about the advantages and disadvantages before reading the closed question that mentioned some possible advantages. During the case study, we found that it wasn't clear to all developers if they should report their opinion on variability modeling in general or on the proposed variability modeling language. Some problems with this specific variability modeling language might have negatively influenced the results on the variability modeling part of the case study.

The researcher who moderated the case study has been working with the Dune developers regularly over the past several years, and thus, there's a trusting relationship between them. Runeson and his colleagues call this a "prolonged involvement."[5] One positive effect of this is that the researcher can understand how the developers interpret the terms used in the study.

As a further step to increase the construct validity, the results of the case study were sent to the participating Dune developers in advance. They confirmed that the results reflect their opinion correctly.

External Validity

The analysis of external validity seeks to determine the extent to which it's possible to generalize the findings for other cases.[5]

The findings of this case study indicate that variability modeling and desk-checking could be found feasible and acceptable for other cases in the context of scientific software development. Further examination is necessary to confirm this.

Reliability

Reliability of validity relates to the extent to which the data and the analysis are dependent on a specific researcher.[5]

During the design, data collection, and analysis of the case study, the researcher continuously documented every step that was performed. A second researcher peer reviewed each step. Furthermore, an external researcher reviewed the case study design. This means there's a reproducible chain of evidence for the case study.

In our current and future work, we plan to adjust the design of the quality assurance process according to the results of the case study. Together with the developers, we'll adjust the desk-checking checklist to fit the need of the Dune development. Currently, we're adapting tool support for feature-oriented software development. This includes support for variability modeling and the development of test applications using the variability model. The variability modeling language used in the tool we plan to use, FeatureIDE (http://fosd.de/fide), is a feature tree that satisfies all our requirements. ■

References

1. J.C. Carver, "Report: The Second International Workshop on Software Engineering for CSE," *Computing in Science & Eng.*, vol. 11, no. 6, 2009, pp. 14–19.
2. K. Pohl, G. Böckle, and F. Linden, *Software Product Line Engineering: Foundations, Principles, and Techniques*, Springer Berlin Heidelberg, 2005.
3. H. Remmel et al., "Design and Rationale of a Quality Assurance Process for a Scientific Framework," *Proc. 5th Int'l Workshop Software Eng. Computational Science and Eng.* (SECSE 13), 2013, pp. 58–67.
4. I.C. Machado et al., "RiPLE-TE: A Process for Testing Software Product Lines," *Proc. 23rd Int'l Conf. Software Eng. and Knowledge Eng.*, 2011, pp. 711–716.
5. P. Runeson et al., *Case Study Research in Software Engineering*, John Wiley & Sons, 2012.
6. D. Kelly, and R. Sanders, "Assessing the Quality of Scientific Software," *Proc. 1st Int'l Workshop on Software Eng. for Computational Science and Eng.*, 2008; http://secse08.cs.ua.edu/Papers/Kelly.pdf.
7. H. Remmel et al., "Supporting the Testing of Scientific Frameworks with Software Product Line Engineering: A Proposed Approach," *Proc. 4th Int'l Workshop Software Eng. Computational Science and Eng.*, 2011, pp. 10–18.
8. V.R. Basili, G. Caldiera, and H.D. Rombach, "The Goal Question Metric Approach," *Encyclopedia of Software Eng.*, John Wiley & Sons, 1994, pp. 528–532.
9. F.D. Davis, R.P. Bagozzi, and P.R. Warshaw, "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models," *Management Science*, vol. 35, no. 8, 1989, pp. 982–1003.

Hanna Remmel is a graduate student at the Institute for Computer Science, University of Heidelberg, Germany. Her research interests include software engineering for computational science and engineering, particularly testing scientific frameworks. Remmel has a master's degree in computer science from the University of Jyväskylä, Finland. Contact her at remmel@informatik.uni-heidelberg.de.

Barbara Paech is the chair of software engineering at the Institute for Computer Science, University of Heidelberg, Germany. Her research interests include achieving quality with adequate effort, requirements engineering, and rationale management. Paech has a PhD in computer science from the Ludwig-Maximilians-Universität München, Germany. Contact her at paech@informatik.uni-heidelberg.de.

Christian Engwer is a junior professor at the University of Münster, Germany. His research interests include numerical methods for the solution of partial differential equations, their application to complex or coupled problems, and the development of efficient and maintainable numerics software. Engwer has a PhD in mathematics from University of Heidelberg, Germany. Contact him at christian.engwer@wwu.de.

Peter Bastian is the chair of scientific computing at the Interdisciplinary Center for Scientific Computing, University of Heidelberg, Germany. His research interests include the numerical solution of partial differential equations, parallel algorithms, and simulation of flow and transport in porous media. Bastian has a PhD in mathematics from the University of Heidelberg, Germany. Contact him at peter.bastian@iwr.uni-heidelberg.de.