# A Literature Review of Agile Practices and Their Effects in Scientific Software Development

Magnus Thorstein Sletholt
University of Oslo
P.O. Box 1080
N-0316 Oslo, Norway
+47 934 97 004
magnusts@ifi.uio.no

Jo Hannay
Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
+47 488 94 138
johannay@simula.no

Dietmar Pfahl
Lund University
P.O. Box 118
SE-221 00 Lund, Sweden
+46 46 222 41 41
dietmar.pfahl@cs.lth.se

Hans Christian Benestad
Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
+47 982 57 805
benestad@simula.no

Hans Petter Langtangen
University of Oslo / Simula Research Laboratory
P.O. Box 134
N-1325 Lysaker, Norway
+47 995 32 021
hpl@simula.no

## ABSTRACT

The nature of scientific research and the development of scientific software have similarities with processes that follow the agile manifesto: responsiveness to change and collaboration are of the utmost importance. But how well do current scientific software development processes match the practices found in agile development methods, and what are the effects of using agile practices in such processes? In order to investigate this, we conduct a literature review, focusing on evaluating the agility present in a selection of scientific software projects. Both projects with intentionally agile practices and projects with a certain degree of agile elements are taken into consideration. In the agility assessment, we define and utilize an agile mapping chart. The elements of the mapping chart are based on Scrum and XP, thus covering two of the most prominent agile reference models. We compared the findings of the literature review to results of a previously conducted survey. The comparison indicates that scientific software development projects adopting agile practices perceive their testing to be better than average. No difference to average projects was perceived regarding requirements-related activities. Future work includes an in-depth case study to further investigate the existence and impact of agility in three large scientific software projects, ultimately aiming at a better understanding of the particularities involved in developing scientific software.

## Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General D.2.9 [**Management**]: Software process models, Software quality assurance K.6.3 [**Software Management**] Software development, Software process

## General Terms

Management, Measurement, Documentation, Experimentation, Human Factors, Theory

## Keywords

Agile software development, Literature review, Scientific software, SCRUM, XP.

## 1. INTRODUCTION

Software engineering (SE) research has traditionally focused on techniques, methods and concepts that are generally applicable. Scientific software, however, operates in very specialized domains. Diane Kelly suggested that the domain-specificity of science might explain why results of research in SE have only rarely been oriented toward scientific computing [14].

Recently however, researchers both within the SE and scientific software communities have started investigations into the nature of scientific software development. Scientists use their software to do complex calculations or simulations. In some scientific projects, the software may be used in order to test a scientific theory. These characteristics of scientific software entail that, in contrast to the development of, say, administrative or business enterprise software, the writers of scientific software cannot determine what the correct output of an application should be in the traditional sense. Also, the software may evolve through the combined effort of a number of scientists over the course of many years, continuously adding new functionality to the system [18]. This poses particular challenges from the software engineering point of view: First, requirements elicitation and specification will be highly dynamic. Due to the exploratory nature of many scientific projects, the elicitation and specification of requirements is problematic because they may be unclear, or even unknown, up-front. In fact, to the degree that any specification is perceived necessary, requirements are typically written near the completion of the software. Second, since requirements are of such a volatile nature, one should expect that testing the software with regards to such requirements would be problematic.

Thus, *a priori*, inherent characteristics of scientific software would seem to impede requirements handling and testing in the outset. In fact, the lack of knowledge about requirements and testing principles has been identified as problem areas in several studies [4, 10, 17]. In a recent survey conducted by some of the present authors, we identified that requirements activities are perceived as problematic in scientific software projects, especially when the teams are large or when scientists dedicate much time to developing software [10]. We also identified that the definition of test cases for validation and verification of the software is perceived as challenging. For example, it is often not obvious to stipulate whether an error lies within the scientific theory or in the implementation (numerical approximation) of that very theory. Among many of the participants in the survey in [10], testing-related activities were indeed regarded as an important part of the project. However, there was a considerable difference between the number of survey participants having said opinion and the number of survey participants having good command of such activities. Consequently, testing skills seems to be a clear weak point for scientists developing software.

In most aspects of the development of scientific software, the urge to conduct science is the primary motivation and goal. Scientists therefore have a different approach to developing software than software engineers; their mindset is to perform science, not to write software [7]. The development method one ends up with is usually one that has emerged as best practices based on local experience [4]. Also, the variation in domains and motivation found in scientific software projects are factors that influence development methods, and one would consequently, expect large variations in development methods both across and within domains.

Nevertheless, some common ground may be found, and due to the challenges with determining requirements up-front and the subsequent testing, scientific software development may lend itself more to agile-oriented practices than plan-driven practices. Sanders supports this notion by stating that most projects under investigation in her study had an iterative, rather than a plan-oriented, approach to development [17].

Therefore, our next step towards a better understanding of scientific software development and in order to propose SE practices for scientific software development, is to review current literature reporting on scientific software development having two goals in mind. First, we aim to investigate the extent to which agile practices have been used in scientific software projects. Second, we aim to investigate the impact on testing and requirements activities in projects with agile practices. We define two propositions, to be investigated as part of the literature review:

P1. Projects using agile practices have a better handling of testing-related activities.

P2. Projects using agile practices have a better handling of requirements activities.

The rest of the paper is structured as follows. In Section 2, we present the list of agile practices that will be used as a reference model for identifying agility in scientific software development projects. In Section 3, we present the literature review and its results. In Section 4, we discuss the findings and the limitations of the study. In Section 5, we describe future work. We conclude in Section 6.

## 2. LIST OF AGILE PRACTICES

To examine the use of agile practices in scientific software projects, we performed a literature review that extracted and critically appraised the available information on the subject. Both intentional, explicit use of agile methods and papers reporting agile-similar practices were included.

**Table 1. Agile Mapping Chart**

| # | Agile practices |
|---|---|
| 1 | Priorities (Product Backlog) maintained by a dedicated role (Product Owner) |
| 2 | Development process and practices facilitated by a dedicated role (Scrum Master) |
| 3 | Sprint planning meeting to create Sprint Backlog |
| 4 | Planning poker to estimate tasks during Sprint planning |
| 5 | Time-boxed sprints producing potentially shippable output |
| 6 | Mutual commitment to Sprint Backlog between Product Owner and Team |
| 7 | Short daily meeting to resolve current issues |
| 8 | Team members volunteer for tasks (self organizing team) |
| 9 | Burndown chart to monitor sprint progress |
| 10 | Sprint review meeting to present completed work |
| 11 | Sprint retrospective to learn from previous sprint |
| 12 | Release planning to release product increments |
| 13 | User stories are written (*) |
| 14 | Give the team a dedicated open work space (*) |
| 15 | Set a sustainable pace (*) |
| 16 | The Project Velocity is measured (*) |
| 17 | Move people around (*) |
| 18 | The customer is always available (*) |
| 19 | Code written to agreed standards (*) |
| 20 | Code the unit test first |
| 21 | All production code is pair programmed |
| 22 | Only one pair integrates code at a time |
| 23 | Integrate often |
| 24 | Set up a dedicated integration computer |
| 25 | Use collective ownership (*) |
| 26 | Simplicity in design (*) |
| 27 | Choose a system metaphor |
| 28 | Use CRC cards for design sessions |
| 29 | Create spike solutions to reduce risk (*) |
| 30 | No functionality is added early |
| 31 | Refactor whenever and wherever possible |

| | |
|---|---|
| 32 | All code must have unit tests |
| 33 | All code must pass all unit tests before it can be released |
| 34 | When a bug is found tests are created |
| 35 | Acceptance tests are run often and the score is published |

The projects described in the papers, were analyzed with respect to 35 agile practices, as listed in Table 1. The first twelve practices originate from the Scrum methodology [5], while the remaining 23 elements are XP practices [20]. We found that six XP practices from [20] overlapped with the Scrum practices and were therefore not included in the table. The six practices are:

EP1. Release planning creates the release schedule.

EP2. A stand up meeting starts each day.

EP3. Make frequent small releases.

EP4. The project is divided into iterations.

EP5. Iteration planning starts each iteration.

EP6. Fix XP when it breaks.

The elements marked with an asterisk in Table 1 are XP practices from [20], but are also recommended practices in the Scrum methodology [5]. In short, Table 1 covers both XP and Scrum, but does not duplicate similar concepts.

By merging Scrum and XP practices we do not rely too heavily on a single methodology. Also, the chosen methodologies are well-established, they are both accessible and there is general consensus with regards to their content.

Scrum and XP are complementary in the sense that Scrum focuses on practices for management and organization, while XP focus more on technical development practices. The combined set of practices addresses a large number of concerns in general software development, while simultaneously capturing the essence of the agile mindset.

## 3. LITERATURE REVIEW

### 3.1 Research Method

The literature review is performed in a similar fashion to the method described in [8]. Due to the sheer number of research fields where scientific software development can be found, multiple literature databases had to be included for a sufficiently comprehensive result set to be returned.

The search query consisted of the following sub-queries:

SQ1. XP AND scientific AND software,

SQ2. Agile AND scientific AND software,

SQ3. Agile AND scientific AND research,

SQ4. XP AND scientific AND research,

SQ5. Scrum AND scientific,

SQ6. Crystal AND scientific,

The pattern of the complete query was then: Q1 or Q2 or Q3 or Q4 or Q5 or Q6.

Agile terms like *lean development* or *feature-driven development* could have been included in the search queries. However, research on less renowned agile methodologies (compared to Scrum and XP), would likely be included in either Q2 or Q3. The query yielded a great number of results; in some databases there was a three-figure number of hits. Most of these papers were clearly irrelevant; many of them originating from Q6 concerning chemistry objects of crystal-nature. There were also a large proportion of clearly irrelevant papers describing the apparent lack of scientific foundation for agile practices, as well as papers on how to execute scientific software on the Windows XP platform.

The papers were collected from the ACM, IEEE Xplore, ScienceDirect and ISI Web of Science databases. After the search in the databases, a search for the keywords was performed in Google Scholar to collect any relevant papers falling short of the original search. This search identified one additional paper (number 9 in the list in Section 3.2). In Table 2, the statistics of the literature search and filtering of results are presented.

A large number of the retrieved papers could be excluded purely based on the title. This was the case when the focus was on aspects other than software development; for example when the science rather than the development was presented, or when the paper did not portray software development at all. For papers where this distinction could not be made based on the title, the abstracts were thoroughly examined. In the IEEE Xplore and ISI Web of Science databases, some fine-tuning of sub-query 6 was necessary due to an overwhelming amount of results (due to chemistry-papers describing objects with crystalline structure). This refinement was based on publication year (we filtered out papers published prior to year 2000), as well as on filters for publication title and subject provided by the respective database search engines.

**Table 2: Summary of search results and filtering**

| | ACM | IEEE | ScienceDirect | ISI Web |
|---|---|---|---|---|
| SQ1 | 5 | 7 | 11 | 11 |
| SQ2 | 14 | 21 | 3 | 24 |
| SQ3 | 12 | 18 | 4 | 27 |
| SQ4 | 3 | 1 | 1 | 7 |
| SQ5 | 2 | 3 | 1 | 3 |
| SQ6 | 26 | 3014 | 305 | 1004 |
| SQ6 (refined) | 26 | 114 | 305 | 4 |
| | | | | |
| Total unique | 49 | 145 | 320 | 59 |
| Excluded title | 36 | 133 | 313 | 59 |
| Excluded abstract | 11 | 8 | 6 | 42 |
| Relevant | 2 | 4 | 1 | 6 |
| | | | | |
| **Total unique relevant** | **8** | | | |

## 3.2 Relevant Papers

The literature search, and subsequent filtering, resulted in the following list of candidate papers for full review:

1. Engineering the Software for Understanding Climate Change [9]
2. An empirical characterization of scientific software development projects according to the Boehm and Turner model: A progress report [6]
3. Test driven development and the scientific method [15]
4. Chaste: using agile programming techniques to develop computational biology software [16]
5. Agile methods in biomedical software development: a multi-site experience report [13]
6. When software engineers met research scientists: A case study [19]
7. Exploring XP for scientific research [21]
8. Is Scrum and XP suitable for CSE Development? [3]
9. Introducing Agile Development into Bioinformatics: an Experience Report [12]

After examining the above papers in detail, four of them (number 2, 3, 6 and 8 in the list) could be excluded from any further review: Paper 2 focuses on a future, planned study where the objective is to obtain an empirical characterization of scientific software. One of the aims of performing the study is to assess how suitable agile and plan-driven approaches are in scientific software projects, and the study, when completed is relevant to our current research. In paper 3, the techniques and practices of XP are compared with the manner of conducting scientific inquiries. Some similarities are investigated (for instance how test-driven development resembles theory building and exploration), but the study is not directly related to scientific software projects and the applied processes therein. A scientific software project was described in paper 6, but the applied process was plan-driven. Due to project issues and largely unsatisfactory development, the authors discuss whether agile practices could be introduced and whether these would resolve (at least to some extent) the problems they encountered. Paper 8, focused on whether it is possible, or sensible, to use agile in scientific software projects. They investigate the constituents of agile methods and assess how each of them aligns with the desiderata of scientific software development. None of the four above mentioned papers reported on experiences with agile practices, and they were therefore excluded from our review.

Summaries of the remaining five relevant papers are presented in the following sub-sections.

### *3.2.1 Paper 1 – Engineering the Software for Understanding Climate Change*

In “Engineering the Software for Understanding Climate Change“, the results of a case study investigating the development practices exercised by climate researchers at the Met Office Hadley Centre are presented. The paper is a collaborative effort between a climate scientist from the research center and a software engineer from the University of Toronto, Canada. The empirical evidence collected were twenty-four interviews with participating scientists, direct observations of meetings and workshops and quantitative data extraction from the code base.

The aim of the study was to investigate the current practices employed by the scientists at Met Office Hadley Centre. The high degree of agility present in the development process was a surprising result for the software engineer. One of the most significant differences, when compared to other scientific software projects, was the emphasis put on verification and validation activities. The authors also claim that requirements handling followed a (semi-)agile approach. As there was no explicit agile method enforced in the project, the high level of agility identified was an important result.

The authors discuss some of the validity threats, and the actions undertaken to handle these, in a separate chapter. One of the threats mentioned by the authors are terminology issues, as some terms are not easily mapped when discussing with scientists, who may have different understanding and recognition of software engineering terms and concepts. Follow-up interviews and feedback sessions with the interviewees were organized to reduce such threats.

The research questions for this study were not directly related to the effects of agile practices. Agile practices were employed, but no explicit agile method was used, and it is difficult to know whether it was aspects of agility in the process that caused the good testing practices. Perhaps the notable testing achievements present here were caused by other factors, such as the level of correctness required in the domain of climate change. Also, the development process had some discrepancies in relation to a proper agile process model, making it questionable whether testing activities indeed were executed in an agile manner.

As the authors of paper 1 report, there are some external validity issues. There was only one project under examination. Although it might be a representative case for the specific domain of climate change, it does not necessarily represent scientific software in general. Some alternative explanations are then plausible, as is also indicated in the paper; the specific way the project adapted and tailored agile practices might have been influenced by the climate change domain’s testing requirements, which may not be as strict in other domains.

The connection between presented evidence and the claims/results is not explicit. This can be due to the focus of the paper, which was to characterize the development practices found in the project. Its main projective was not to assess the suitability of agile methods, and even less to emphasize on the effects of an agile approach. Therefore, the paper is mostly relevant for analyzing the presence of agile practices, not so much for examining their effects.

The project is referred to as Project 1 in Table 3.

### *3.2.2 Paper 2 – Chaste: Using Agile Programming Techniques to Develop Computational Biology Software*

Chaste is a computational biology project with a large number of scientists involved. The aim of the project is to provide a library for cardiac modeling and cardiac electrical activity simulation. The paper is written by a total of ten researchers, stringing together efforts from both computer scientists and biologists. A case study regarding the use of agile methods is the topic of the paper.

Introduction of XP in the Chaste project was claimed to be a massive success. They found the basic agile principle of being responsive to change to be very much in the natural spirit of general scientific research. Consequently they favored the responsive ability imposed by adopting XP in the project. The authors also emphasized that the agile approach to testing was a valuable asset, both concerning the testing of new functionality and regression testing of existing functionality.

The evidence is presented in a reasonably comprehensive manner, although the composition of the teams and organizational aspects are not described in much detail. The structure within and across the teams, as well as the number of teams and members within a team, remain unknown. It is suggested that there was a large number of scientists involved and it would have been interesting to know more about the organizational aspects to be better able to discuss the potentials for generalizing the results from the paper.

The project investigated is referred to as Project 2 in Table 3.

### 3.2.3 Paper 3 – Agile Methods in Biomedical Software Development: A Multi-Site Experience Report

Paper 3 is another study with origins from the field of bioinformatics. The paper reports on experiences from multiple sites and projects. A total of six projects, all incorporating key agile practices, are examined by the authors. The multidisciplinary group of authors represents different universities and research centers, all based in the United States.

Agile methods were deemed very suitable for this type of scientific research and software development. The developers regarded the agile approach to be a key success factor. In this line of biomedical software development, the software has to be responsive to change at two levels. Both progress in the scientific domain and specific customer demands can enforce changes to the software.

Subjective experiences are the primary source of evidence in the paper. The group of projects under examination was selected during meetings and biomedical conference discussions. To collect and elicit the tacit knowledge and experience from the involved parties, the authors initiated a basic mapping survey. Thereafter they conducted open-ended interviews with key developers across the projects. To ensure the quality of the collected data, several rounds of feedback sessions were arranged. The authors extensively described the evidence and the method with which it was collected.

A notable strength compared to the other studies in this report is that data was collected from six different projects. Similar effects were reported in all cases, which strengthen the claims of positive effects due to agile practices.

However, the cases under examination have some obvious similarities, restraining the scope for external validity:

1. They were all of small size (a single team with 2-5 members)
2. They were all in the domain of biomedical software development.

The first issue is consistent with the notion that small size projects are more inclined to succeed with agile methods, than larger projects with multiple teams. It has been suggested in some studies that XP does not scale well to extensive projects. The second issue pertains to whether some common attributes present in biomedical software development make them more prone to embark on and succeed with agile development methods. It would be interesting to observe whether a contrast case shared the same results as the ones investigated.

The six projects described in the paper are referred to as Projects 3.1to 3.6 in Table 3.

### 3.2.4 Paper 4 - Exploring XP for Scientific Research

In this paper the authors reported from an attempt to apply XP to a project at a NASA research center. The two authors worked closely together, and were the only developers on the project. They aimed at assessing the suitability of XP in a scientific software project, and reported that XP was successfully adopted in their project. In particular, they reported that code quality improved, more bugs were caught, development was more focused, maintenance was easier and productivity increased.

Although the use of XP was reported as promising, there are some limitations to the study's generalizability. First, the software project was very small, consisting of only two members. Also, some of the applied practices, such as pair programming, are only possible to perform when the scientists are co-located and available concurrently, which is not always the case for scientists. The paper is still relevant for small scientific project teams, albeit more valuable on a small-scale team basis than on a managerial level. The results might not be transferable at all to a scientific project of larger size.

Another limitation to the relevance for the scientific community at large is that the developers could work exclusively with this project during the 2-month life span of the project. For generalizability, there are two problems with this situation: First, most scientific software projects have a life span of several years, sometimes even decades, meaning that any long-term effect of using XP is not addressed by this study. Second, even though an increasing part of work time is devoted to writing software, most scientists are engaged in other research and education activities. There is seldom opportunity for full-time dedication to software development for all team members in a large project. Therefore, the project context reported in paper 4 is somewhat remote from the regular settings of scientific software projects.

The researchers had no experience with agile methods prior to the experiment. No software engineers were involved in the planning or execution of it. The assessment of how well the new development methodology was implemented is highly subjective. The authors, being excellent researchers and capable scientific programmers, still lack software engineering knowledge. The reader might question the degree to which agile principles were carried out properly, and thereby question the validity of the reported results.

Also worth noting is the fact that the effects of applying XP may be confounded with effects of other new elements brought into the project. Ruby and object-oriented design represented two completely unfamiliar concepts. The usual language for the two researchers, Fortran, is quite different from Ruby. Perhaps Ruby with its object-orientation and testing frameworks were as significant to the alleged improvements as the use of XP. Consequently, the reasons for the team's apparent preference of XP to prior practices may have been connected to other elements introduced, or a combination of these and XP. There is no substantial evidence presented indicating that the testing improvements are exclusively caused by the use of XP in the project.

The project is referred to as Project 4 in Table 3.

### 3.2.5 *Paper 5 - Introducing Agile Development into Bioinformatics: an Experience Report*

One of the authors of Paper 3 [13] also wrote a report on the experiences of introducing agile techniques in a bioinformatics project. This project was not among the six in Paper 3. The author presents an application of an agile method to their process, as an answer to their need for being more responsive to changing requirements. Various agile practices, adopted from a combination of SCRUM and XP, were then incrementally incorporated into the development process.

The authors report positive experiences with implementing agile practices in increments, focusing on testing and requirements activities. They found the agile practices to be beneficial in dealing with flexible requirements. The agile testing practices also facilitated the scientific setting where correct and reproducible results are of the utmost importance.

The evidence consists of the subjectively reported experiences of the involved project members. The incremental fashion of introducing the agile method is well documented in the paper, while the effects of each increment are documented to a lesser degree. It is therefore difficult to find evidence on the effects of agile practices from the paper. In conclusion, the findings in this study cannot be particularly emphasized when conclusions are made in the synthesis.

The project is referred to as Project 5 in Table 3.

## 3.3 Mapping of Projects to Agile Practices

Once the final set of relevant papers was defined, we used a simple yes/no indicator when assessing each reported project against each individual practice. Table 3 shows the result of this mapping. Fields are left blank if we were not able to determine from the available information whether (or not) a practice was followed. This was the case in particular with papers more focused on the effects of the agile approach, rather than naming or describing the employed practices in much detail. Nonetheless, even with no explicit mentioning that a particular practice was not used, it was sometimes possible to assign a "No". For example, for Project 1 in Table 3, we could assess that practice 1, i.e., "Priorities (Product Backlog) maintained by a dedicated role (Product Owner)", was not used, because the paper stated that all project members identified, specified and prioritized new features, hence we could deduce that prioritization was not a centralized responsibility of a Product Owner.

**Table 3: Agile mapping for the examined projects**

| | Projects | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| # | *1* | *2* | *3.1* | *3.2* | *3.3* | *3.4* | *3.5* | *3.6* | *4* | *5* |
| **1** | No | | | | | | | | No | |
| **2** | No | | | | | | | | No | |
| **3** | | | | | | | | | No | |
| **4** | No | | | | | | | | No | |
| **5** | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **6** | No | | | | | | | | No | |
| **7** | No | Yes | Yes | | | Yes | Yes | Yes | Yes | Yes |
| **8** | Yes | | Yes | Yes | No | Yes | Yes | Yes | Yes | |
| **9** | No | Yes | | | | | | | Yes | |
| **10** | | | | | | | | | No | |
| **11** | | | | | | | | | Yes | |
| **12** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **13** | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | | Yes |
| **14** | Yes | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | |
| **15** | | | | | | | | | Yes | |
| **16** | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **17** | No | Yes | | | | | | | | Yes |
| **18** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **19** | Yes | | | | | | | | Yes | Yes |
| **20** | | Yes | | | | | | | | |
| **21** | | No | No | No | No | No | No | No | Yes | |
| **22** | | | | | | | | | Yes | |
| **23** | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **24** | | | | | | | | | | |
| **25** | Yes | Yes | Yes | Yes | | Yes | Yes | Yes | Yes | Yes |
| **26** | | | | | | | | | Yes | |
| **27** | | | | | | | | | Yes | |
| **28** | | | | | | | | | | |
| **29** | | | | | | | | | | |
| **30** | Yes | | | | | | | | | |
| **31** | | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| **32** | | Yes | | | | | | | Yes | |
| **33** | | | | | | | | | | |
| **34** | | Yes | | | | | | | | |
| **35** | Yes | Yes | | | | | | Yes | | |

## 3.4 Synthesis of Findings

The papers all indicated positive effects of agile practices in scientific software developments. A tentative conclusion is that agile methods can effectively handle the special characteristics of requirements and testing in scientific software development. The evidence in favor of such a conclusion is stronger for small projects with relatively few team members.

Almost all of the studies reported on improvements to the testing activity. Testing was performed more extensively, and the approaches to adding tests of new modules improved [16, 21]. The rigor of the testing approaches seemed to satisfy the need for having reproducible, correct results [12]. For requirements activities, a degree of mismatch was identified between scientific software projects and the agile-assumed context of a clear customer-developer relationship. However, the responsiveness and flexibility of agile methods proved valuable for the requirements activities. Elicitation and specification of tasks were perceived as easier and more focused with agile methods [16, 21]. Good practices regarding requirements prioritization were also observed [9].

In conclusion, the literature review supported proposition P1, i.e., that projects using agile practices have a better handling of testing-related activities. The review gives some support for proposition P2, i.e., that projects using agile practices have better handling of requirements activities, but the findings are not as substantial as for P1.

## 4. DISCUSSION OF FINDINGS AND LIMITATIONS

The papers were all experience reports. Consequently, the evidence presented in the papers was personal experience typically gathered from interviewing key project members. Some authors also used direct observations and multiple feedback interview sessions [9, 13]. The authors of the papers were often participants in the systems development themselves, relying on a combination of their own expert opinion and personal experience when arguing for their claims. This may open up for both researcher and personal bias, especially in papers with few authors. Some of the studies [9, 12, 13] did not have the effects or results of applying agile methods as the primary focus. The claim-evidence relationship is less obvious in these studies than in studies focusing on applying an explicit agile method and reporting on the results of doing so [16, 21].

Internal validity issues were most salient in [9, 21], and especially evident in [21] where they introduced several new elements besides XP. Only in one study was there any consideration as to the long-term effects of applying agile methods [16]. It is hard to make generalized assumptions based on this single study, particularly when the study only examined a single project. Other internal validity issues were present in [9, 13], due to not having a defined agile methodology applied. It is difficult to know how much of the results can be attributed to the agility in the processes.

For external validity, we cannot automatically transfer the results of these studies to the general scientific software community. First, the settings of the projects and the composition of the teams were not necessarily representative for scientific software development. Although there were a considerable number of projects investigated, nearly all of these were of small size (generally 2-5 participants). As mentioned in characterization studies of scientific software [17], there are often a large number of people contributing to the development, which provide extra obstacles for such projects. None of the projects were of a distributed nature, which is also quite common in scientific software development.

Second, the domain of most of the projects was bioinformatics, which is tightly connected to general informatics and computer science. Bioinformaticians may have an adequate understanding of software engineering concepts incorporated in their formal education, since a common career path is to take a bachelor degree in computer science and then proceed with further education in bioinformatics. It may be that the increased level of software engineering knowledge makes scientists in such projects more prone to apply software engineering concepts and practices and to succeed when doing so.

Limitations to the reviewed papers also limit the degree to which we can draw conclusions in our literature review. We have made efforts to take this into account when reporting our findings. Additional limitations to our review include *reliability* threats due to single-reviewer assessment, *publication bias* due to papers possibly being submitted and published more readily when they report positive findings, and *selection bias* due to reviewer reliability threats and search engine mechanics. Reliability threats and selection bias will be lessened by adding multiple reviewers in the next round of this line of research. Publication bias can be ameliorated to some extent by also including “gray literature” (technical reports, unpublished material etc.), but we consider the cost of retrieving such literature to be high compared to how this might benefit our goal in this paper, which is to prepare for a more thorough investigation into agile practices in scientific software development projects.

## 5. FUTURE WORK

In continuation of our questionnaire based survey [10] and the literature review reported in this paper, our next step will be to conduct a case study [22] that investigates three large scientific software projects; FEniCS, Dalton and OLGA. The case study will be exploratory in the first part and confirmatory in the second part. Research methods will include techniques for eliciting and externalizing practitioners’ tacit knowledge [1, 2, 11].

The purpose of the case study is to:

1. Analyze and conceptualize core process elements in the software development processes in the three projects.
2. Investigate to what extent these elements map to elements in agile methodologies, i.e. evaluation according to the agile mapping chart in Table 1.
3. Compare the agile mapping charts from the three projects to the findings of the literature review.

In each project, 3 to 4 key developers will be interviewed. We have gained access to the projects through the network of one of the authors of this paper. The projects are all international collaborations, but the interviewees work in sub-projects located in Norway.

The first case under investigation is FEniCS, which is a mutual project joining together participants from several universities and research institutions in the computational mathematics domain. The aim is to facilitate automatic solution of differential equations. Although in a constantly operational state, there is still no version 1.0 of FEniCS. The program is open source and available for everyone and even distributed through software managers in Ubuntu and Debian.

FEniCS is no traditional software application per se, but rather a collection of (more or less) separate packages that form a framework for automated solution of differential equations. Researchers write applications, typically relating to a fairly specific scientific problem, on top of the FEniCS framework/interface. The components are written in C++ and Python. There is an international community of developers who contribute with coding and documentation, thus implying a fairly distributed development effort.

The Dalton project is an older scientific software project, in the molecular electronic structures sub-domain of chemistry, aiming to automate computation of such molecular properties. The software was first released in 1997, with several versions in the years to follow; the latest dating to the first quarter of 2010. Like FEniCS there is an international community of scientists involved in the development of the program.

The program is written in FORTRAN.77 and C, and the authors recommend a UNIX platform. The program consists of seven components, with more or less independent development cycles. The program is distributed free of charge, as long as the user signs a personal license agreement.

The third case is OLGA. Contrary to the other cases, this is a commercial project, developed by the SPT Group. OLGA is a simulator tool for accurate flow modeling of oil, water and gas in wells and pipelines. Being a commercial system that must stay competitive, OLGA has a more traditional type of developer-customer relationship.

These cases will complement the projects investigated in the literature review. They will represent different types of scientific software than the projects investigated in the review, as they are much larger in terms of size, life-cycle and participants. By the selection of cases, we will investigate projects in multiple domains, and domains other than bioinformatics. In the event of detecting agile practices in the cases investigated, the combined analysis of these and the projects examined in the literature review will enhance the evidence base, and hopefully, increase the potential for generalizing findings about scientific software projects employing agile practices.

## 6. CONCLUSION

Following previous and current investigations into the practice of scientific software development, we consider it valuable to analyze and conceptualize the core process elements of such development, in terms of modern software engineering best practices.

In this paper, we reported the results of a literature review to investigate the presence of agile practices in scientific software development, and to summarize evidence on the effects of such practices. This is the second study in a three-step research plan, succeeding a survey on software engineering practices in scientific software development, and preceding an in-depth case study on the same topic.

A likely outcome of such investigations will be more explicit and deliberate scientific software development practices, and also a timely updating of software engineering methodology to include domains other than business-administrative software.

Overall, the literature review indicated that agile techniques had positive effects in the projects investigated. None of the studies displayed any particular negative side effects. Generally, there are some validity issues, both internal and external. Hence, in order to be conclusive on the pros and cons of agile in scientific software, more research on the matter is needed. Nevertheless, the initial results from the literature review are indeed promising, so a preliminary conclusion may be that the agile approach can be valuable to scientific software development, especially for smaller-sized teams and projects.

## 7. ACKNOWLEDGMENTS

Parts of this work were funded by the Excellence Center at Linköping – Lund in Information Technology (ELLIIT), and by the Simula School of Research and Innovation.

## 8. REFERENCES

[1] Argyris, C. 1993. *Knowledge for Action*. Jossey-Bass Publishers, San Francisco, CA.

[2] Argyris, C. and Schön, D. A. 1996. *Organizational Learning II. Theory, Method, and Practice*. Addison-Wesley Publishing Company, Reading, MA.

[3] Blom, M. 2010. Is scrum and XP suitable for CSE development? *Procedia Computer Science* 1, 1 (May 2010), 1511-1517.

[4] Carver, J. C., Kendall, R. P., Squires, S. E. and Post, D. E. 2007. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. In *Proceedings of the 29th International Conference on Software Engineering* (Minneapolis, MN, May 20-26, 2007). ICSE'07. IEEE Computer Society Washington, DC, USA, 550-559.

[5] Cohn, M. 2009. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, Boston, MA.

[6] Crabtree, C. A., Koru, A. G., Seaman, C., and Erdogmus, H. 2009. An Emprical Characterization of Scientific Software Development Projects According to the Boehm and Turner Model: A Progress Report. In *Proceedings of the Second International Workshop on Software Engineering for Computational Science and Engineering* (Vancouver, Canada, May 23, 2009). SECSE'09. IEEE Computer Society Washington, DC, USA, 22-27. DOI= 10.1109/SECSE.2009.5069158.

[7] Decyk, V. K., Norton, C. D. and Gardner, H. J. 2007. Why Fortran? *Computing in Science and Engineering* 9, 4 (Jul-Aug. 2007), 68-71. DOI=10.1109/MCSE.2007.89.

[8] Dybå, T., Dingsøyr, T. and Hanssen, G. K. 2007. Applying Systematic Reviews to Diverse Study Types: An Experience Report. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement* (Madrid, Spain, Sep. 20-21, 2007). ESEM'07. 225-234. DOI=10.1109/ESEM.2007.59.

[9] Easterbrook, S.M. and Johns, T.C. 2009. Engineering the Software for Understanding Climate Change. *Computing in Science and Engineering* 11 (6), 64-74. DOI= 10.1109/MCSE.2009.193

[10] Hannay, J.E., Langtangen, H.P., MacLeod, C., Pfahl, D., Singer, J. and Wilson, G., How Do Scientists Develop and Use Scientific Software? in *Second International Workshop on Software Engineering for Computational Science and Engineering* (Vancouver, Canada, May 23, 2009). SECSE'09. IEEE Computer Society Washington, DC, USA, 1-8.

[11] Jarvis, P. 1999. *The Practitioner-Researcher*. Jossey-Bass Publishers.

[12] Kane, D. 2003. Introducing Agile Development into Bioinformatics: An Experience Report. In *Proceedings of the Agile Development Conference* (Salt Lake City, USA) ADC'03, IEEE Computer Society Washington, DC, USA, 132-139.

[13] Kane, D. W., Hohman, M. M., Cerami, E. G., McCormick, M. W., Kuhlmman, K. F. and Byrd, J. A. 2006. Agile methods in biomedical software development: a multi-site experience report. *BMS Bioinformatics* 7 (273), 1-12.

[14] Kelly, D.F. 2007. A Software Chasm: Software Engineering and Scientific Computing. *IEEE Software* 24, 6 (Nov./Dec. 2007), 118-120.

[15] Mugridge, R. 2003. Test Driven Development and the Scientific Method. In *Proceedings of the Agile Development Conference* (Salt Lake City, USA) ADC'03, IEEE Computer Society Washington, DC, USA, 47-52.

[16] Pitt-Francis, J., Bernabeu, M.O., Cooper, J., Garny, A., Momtahan, L., Osborne, J., Pathmanathan, P., Rodriguez, B., Whiteley, J.P. and Gavaghan, D.J. 2008. Chaste: using agile programming techniques to develop computational biology software. *hilosophical Transactions of the Royal Society - Series A: Mathematical, Physical and Engineering Sciences 366* (1878). 3111-3136.

[17] Sanders, R. 2008. The Development and Use of Scientific Software, *MSc Thesis*, Queen's University.

[18] Sanders, R. and Kelly, D. 2008. Dealing with Risk in Scientific Software Development. *IEEE Software* 25, 4 (Jul.-Aug. 2008), 21-28.

[19] Segal, J. 2005. When software engineers met research scientists: A case study. *Empirical Software Engineering* 4, 10 (Oct. 2005), 517-536.

[20] Wells, D. 1999. The Rules of Extreme Programming. URL: http://www.extremeprogramming.org/rules.html. Accessed on January 19, 2011.

[21] Wood, W. A. and Kleb, W. L. 2003. Exploring XP for Scientific Research. *IEEE Software* 20, 3 (May-June 2003), 30-36. DOI=10.1109/MS.2003.1196317.

[22] Yin, R.K. 2003. *Case Study Research: Design and Methods*. Sage Publications.