# How Do Scientists Develop and Use Scientific Software?

Jo Erskine Hannay
Dept. of Software Engineering
Simula Research Laboratory
Dept. of Informatics, Univ. of Oslo
johannay@simula.no

Hans Petter Langtangen
Center for Biomedical Computing
Simula Research Laboratory
Dept. of Informatics, Univ. of Oslo
hpl@simula.no

Carolyn MacLeod
Dept. of Computer Science
University of Toronto
cmacleod@cs.utoronto.ca

Dietmar Pfahl
Dept. of Software Engineering
Simula Research Laboratory
Dept. of Informatics, Univ. of Oslo
dietmarp@simula.no

Janice Singer
Software Engineering Group
National Research Council of Canada
janice.singer@nrc-cnrc.gc.ca

Greg Wilson
Dept. of Computer Science
University of Toronto
gvwilson@cs.utoronto.ca

## Abstract

*New knowledge in science and engineering relies increasingly on results produced by scientific software. Therefore, knowing how scientists develop and use software in their research is critical to assessing the necessity for improving current development practices and to making decisions about the future allocation of resources. To that end, this paper presents the results of a survey conducted online in October–December 2008 which received almost 2000 responses. Our main conclusions are that (1) the knowledge required to develop and use scientific software is primarily acquired from peers and through self-study, rather than from formal education and training; (2) the number of scientists using supercomputers is small compared to the number using desktop or intermediate computers; (3) most scientists rely primarily on software with a large user base; (4) while many scientists believe that software testing is important, a smaller number believe they have sufficient understanding about testing concepts; and (5) that there is a tendency for scientists to rank standard software engineering concepts higher if they work in large software development projects and teams, but that there is no uniform trend of association between rank of importance of software engineering concepts and project/team size.*

## 1. Motivation

There is probably not a single scientist who has not, at some point in time, used a software system to analyze, visualize, or simulate processes or data. Many scientists use such software daily, while others develop it for their own use or for a wider community.

As many researchers have pointed out [2, 3, 6, 8], there is a wide chasm between the general computing community and the scientific computing community. As a result, there has been little exchange of ideas relevant to scientific application software.

One reason for this is that in scientific computing, a developer must have intimate knowledge of the application domain (i.e., the science), whereas in "regular" software development (of, say, an enterprise resource planning system), developers are much less likely to need to be domain experts. It follows that a scientific software developer is likely to be among the end-users, whereas a developer in "traditional" software engineering is most likely not. Scientific software is also often explorative: the purpose of the software is usually to help understanding a new problem, implying that up-front specification of software requirements is difficult or impossible [9]. This may inhibit initiation of fruitful collaboration between software engineers and scientists.

Whether these facts are the cause of differences in the development processes of scientific software versus those of other software, and whether one should apply software engineering processes to the development of scientific software, is the subject of active investigation. However, it is obvious that there *are* differences in development process, and in the roles of those who develop and use different kinds of research software.

The aim of this study was therefore to investigate how the majority of working scientists develop and use scientific software in their day-to-day work. Our overall research questions are listed below.

RQ1. How did scientists learn what they know about developing/using scientific software?
RQ2. When did scientists learn what they know about developing/using scientific software?
RQ3. How important is developing/using scientific software to scientists?
RQ4. How much of their working time do scientists spend on developing/using scientific software?
RQ5. Do scientists spend more time developing/using scientific software than in the past?
RQ6. On what scale of hardware do scientists develop/use scientific software?
RQ7. What are the sizes of the user communities of scientific software?
RQ8. How familiar are scientists with standard concepts of software engineering?
RQ9. Does program size, time spent on programming, or team size influence scientists' opinions about the importance of good software development practices?

## 2. Research Method

From the research questions, we formulated our expectations with regards to each question as a set of hypotheses. These hypotheses, in turn, gave rise to questionnaire items that were given on an online survey, which we advertised through mailing lists, bulletin boards, word of mouth, and with advertisements in both the online and print editions of *American Scientist* magazine [1].

## 3. Results

### 3.1. Demographics

1972 usable responses were collected between October and December 2008. Most scientists classified themselves into the age ranges 18 to 30 years (649 responses—33%) or 30 to 40 years (681 responses—34%). The remaining age ranges (40 to 50 years, 50 to 60 years, over 60 years) received 343, 187, and 97 responses repectively. Fifteen respondents did not disclose their age.

Valid responses were received from a total of 40 countries. More than 50% of all responses came from five countries: United States, Canada, United Kingdom, Germany, and Norway, with 579, 136, 136, 117, and 99 responses, respectively. Grouped by continents, most responses came from Europe and North America with 725 responses (including Russia and Turkey) and 715 responses, respectively, followed by Australia/New Zealand and Asia with 66 and 57 responses, respectively. The number of responses from South America, Central America, and Africa was below 50 for all three geographic regions taken together.

About two thirds of the respondents stated that their highest academic degree is a Ph.D. (or equivalent). About 20% stated they have an M.Sc. degree (or equivalent), and about 10% stated they have an B.Sc. degree (or equivalent).

About 50% of the respondents stated that they are academic researchers (professors, post-docs, or similar), followed by graduate students (25%), programmers (20%), government research scientists (16%), engineers (15%), software engineers (13%), teachers (12%), managers/supervisors (8%), industrial research scientists (7%), system administrators (7%), laboratory technicians (3%), and clinicians (1%). The percentages do not add up to 100% because scientists were given the opportunity to check more than one option to describe their current occupation.

### 3.2. Main Findings

For each research question, we present our associated expectations in the form of hypotheses, and then give the relevant survey results.

We refined RQ1 into the following two hypotheses:

H1a. Most scientists learn most of what they know about *developing* software on their own or informally from their peers, rather than through formal training.
H1b. Most scientists learn most of what they know about *using* software on their own or informally from their peers, rather than through formal training.

*Survey results*:

H1a: Using a five-point scale ('not at all important', 'not important', 'somewhat important', 'important', 'very important'), 96.9% of the responses state that informal self study is important or very important for developing software. 60.1% state that informal learning from peers is important or very important. Only 34.4% state that formal education at an educational institution is important or very important and only 13.1% state that formal training at work is important or very important.

H1b: 96.5% of the responses state that informal self study is important or very important for using software. 69.4% state that informal learning from peers is important or very important. Only 26.6% state that formal education

at an educational institution is important or very important and only 17.1% state that formal training at work is important or very important.

We refined RQ2 into the following two hypotheses:

H2a. Most scientists learn what they know about *developing* software early in their careers (undergraduate and graduate degrees, or equivalent years in industry).

H2b. Most scientists learn what they know about *using* software early in their careers (undergraduate and graduate degrees, or equivalent years in industry).

*Survey results*:

H2a: One finding related to hypothesis H1a was that formal education is important, or very important, for only 34.4%. On a more detailed level, the survey results indicate that the importance of graduate studies is clearly greater than that of undergraduate studies which, in turn, is clearly greater than that of high school studies. Another finding related to hypothesis H1a was that formal training at work was considered as important or very important for only 13.1%. Independently of whether scientists received formal training at the workplace or learned informally through self-study or from peers, there is the following trend: The last five years during professional work are more important for learning about developing scientific software than the periods six to ten years ago, eleven to 15 years ago, or longer than 15 years ago. However, this trend is not as pronounced as the effect of recency (and thus higher level) of formal education.

H2b: Similar to the results for H2a, the importance of graduate studies is clearly greater than that of undergraduate studies which, in turn, is clearly greater than that of high school studies. Also, the importance of learning about software development during professional work increases slightly for more recent professional work.

We refined RQ3 into the following two hypotheses:

H3a. *Developing* scientific software is important to the majority of scientists.

H3b. *Using* scientific software is important to the majority of scientists.

*Survey results*:

H3a: 84.3% of the responses state that developing scientific software is important or very important for their own research. 46.4% state that developing scientific software is important or very important for the research of others.

H3b: 91.2% of the responses state that using scientific software is important or very important for their own research. (We did not ask whether the use of scientific software might be important for others.)

We refined RQ4 into the following two hypotheses:

H4a. Most scientists spend less than ten hours (or one day) per week (or less than 20% of their working hours) *developing* software.

H4b. Most scientists spend less than ten hours (or one day) per week (or less than 20% of their working hours) *using* software for research purposes.

*Survey results*:

H4a: On average, scientist spend approximately 30% of their work time developing scientific software.

H4b: On average, scientist spend approximately 40% of their work time using scientific software.

We refined RQ5 into the following two hypotheses:

H5a. In the past, scientists and engineers used to spend less time *developing* software than today.

H5b. In the past, scientists and engineers used to spend less time *using* software than today.

*Survey results*:

H5a: Using a five-point scale ('much less time', 'less time', 'same amount of time', 'more time', 'much more time'), 53.5% of the responses state that scientists spend more or much more time developing scientific software than they did 10 years ago. 44.7% state that scientists spend more or much more time developing scientific software than they did 5 years ago. 14.5% state that scientists spend more or much more time developing scientific software than they did 1 year ago.

H5b: 85.9% of the responses state that scientists spend more or much more time using scientific software than they did 10 years ago. 69.5% state that scientists spend more or much more time using scientific software than they did 5 years ago. 19.8% state that scientists spend more or much more time using scientific software than they did 1 year ago.

We refined RQ6 into the following two hypotheses:

H6a. Most scientists use a desktop computer (or laptop) for their software development work, very few develop scientific software on intermediate computers (or parallel small-size clusters), and less than 10% ever develop scientific software on a supercomputer.

H6b. Most scientists use a desktop computer (or laptop) to run scientific software, very few use scientific software on intermediate computers (or parallel small-size clusters), and less than 10% ever use scientific software on a supercomputer.

*Survey results*:

H6a: *Desktop computers*: 42.9% of the scientists develop scientific software using exclusively desktop computers, and 77.9% of the scientists spend 60.0% or more of their time developing scientific software using desktop computers. Only 4.3% of the scientists never develop scientific software on a desktop computer. *Intermediate computers*: 55.2% of the scientists never use an intermediate computer to develop scientific software, and 81.3% of the

scientists spend 20.0% or less of their time developing scientific software using intermediate computers. Only 0.7% of the scientists always develop scientific software on an intermediate computer. *Supercomputers*: 75.2% of the scientists never use a supercomputer to develop scientific software, and 91.6% of the scientists spend 20.0% or less of their time developing scientific software using a supercomputer. Only 0.3% of the scientists always develop scientific software on a supercomputer and only 2.3% spend 50.0% or more of their time developing scientific software on a supercomputer. *Average time*: Scientists state that they spend on average 5.6% of their time developing scientific software using supercomputers, 12.8% of their time developing scientific software using intermediate computers, and 79.1% of their time developing scientific software using desktop computers. Note that the averages do not add up to 100% because some respondents either did not provide data that adds up to 100% or did not provide responses to all three hardware categories. In the latter case, we assumed that not providing data for a hardware category implies that 0% of the time developing scientific software is spent on hardware of that category.

H6b: *Desktop computers*: 48.5% of the scientists use scientific software exclusively on desktop computers, and 81.7% of the scientists spend 60.0% or more of their time using scientific software on desktop computers. Only 2.3% of the scientists never use scientific software on a desktop computer. *Intermediate computers*: 58.0% of the scientists never use scientific software on an intermediate computer, and 85.5% of the scientists spend 20.0% or less of their time using scientific software on intermediate computers. Only 0.2% of the scientists always use scientific software on an intermediate computer. *Supercomputers*: 79.9% of the scientists never use scientific software on a supercomputer, and 93.2% of the scientists spend 20.0% or less of their time using software on a supercomputer. Only 0.2% of the scientists always use scientific software on a supercomputer and only 1.7% spend 50.0% or more of their time using scientific software on a supercomputer. *Average time*: Scientists state that they spend on average 4.5% of their time using scientific software on supercomputers, 10.4% of their time using scientific software on intermediate computers, and 83.1% of their time using scientific software on desktop computers. Note that the averages do not add up to 100% because some respondents either did not provide data that adds up to 100% or did not provide responses to all three hardware categories. In the latter case, we assumed that not providing data for a hardware category implies that 0% of the time using scientific software is spent on hardware of that category. The standard deviations were larger for H6a than those for H6b. More scientists use, than develop, scientific software.

We refined RQ7 into the following hypothesis:

H7. Most scientific software is used by either a very small number of people or a very large number.

*Survey results*: 56.2% of the respondents believe that the most important scientific software they use has more than 5000 users worldwide. In contrast, 10.7% of the respondents believe that the most important software they use has less than 3 users worldwide. Similar patterns apply to the second, third and fourth important scientific software used.

We refined RQ8 into the following two hypotheses:

H8a. Most scientists are not familiar with standard software engineering concepts.

H8b. Most scientists do not consider standard software engineering concepts as important for their work.

*Survey results*: Scientists were asked to rank their understanding of the following software engineering concepts: software requirements (e.g., eliciting, analyzing, specifying and prioritizing functional and non-functional requirements), software design (e.g., specifying architecture and detailed design using design by contract, design patterns, pseudo-code, or UML), software construction (e.g., coding, compiling, defensive programming), software verification (e.g., correctness proofs, model-checking, static analysis, inspections), software testing (e.g., unit testing, integration testing, acceptance testing, regression testing, code coverage, convergence analysis), software maintenance (e.g., correcting a defect, porting to new platforms, refactoring), software product management (e.g., configuration management, version control, release planning), software project management (e.g., cost/effort estimation, task planning, personnel allocation). Then scientists were asked to assign to each of these software engineering concepts an importance rank. The question about the scientists' understanding of software engineering concepts used the following five-point scale: 'No idea what it means', 'Vague understanding', 'Novice understanding', 'Understand for the most part', 'Expert-level understanding'. The question about the scientists' judgment of the importance of each of these concepts for their work used the following five-point scale: 'Not at all important', 'Not important', 'Somewhat important', 'Important', 'Very important'. The related question had the following formulation: 'How well do you think you understand each of the following software engineering concepts, and how important is each to your work?'.

Figure 1 summarizes results related to H8a and H8b. The black and the white bars show the relative frequencies (percentages) of the answers in the two top categories for understanding and importance, respectively. (The label 'good/expert understanding' represents answer categories 'Understand for the most part' and 'Expert-level understanding'.) The gray 'gap' bars represent the differences between the combined percentages of the two top categories of assumed understanding of a software engineering concept

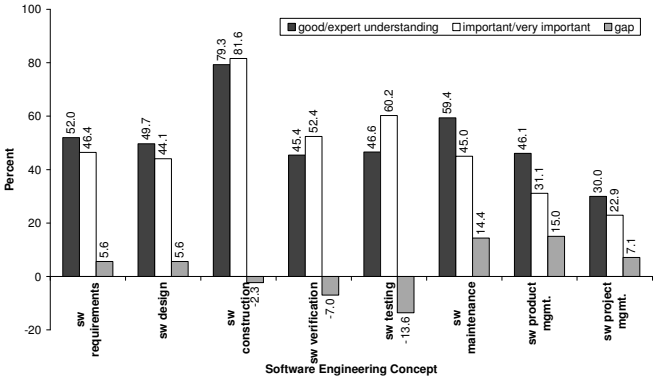


**Figure 1. Perceived Importance versus Understanding of Standard Software Engineering Concepts**

and perceived importance of that concept. We used only the responses in the two top categories because we wanted to check whether scientists have at least as much good or expert-level understanding of a standard software engineering concept as they believe this concept is important or very important for their work. The survey results show that this is not the case for the concepts 'software construction', 'software verification', and in particular for 'software testing'.

We refined RQ9 into the following hypothesis:

H9. How scientists rank the importance of standard software development practices for their own software development is independent of the size of the program they are working on, and the amount of time they spend developing scientific software, but the importance rank increases with the size of the team they are part of.

*Survey results*: We used one-way analyses of variance (ANOVA) to compare average rank of importance scores between groups defined by project size and groups defined by team size. Since we were interested in all pairwise comparisons between groups, we employed Tukey's HSD method for post-hoc comparisons when the ANOVA null hypotheses of no differences between groups were rejected at $\alpha = .05$.

*Importance of software requirements*: Respondents who are involved in small projects and who work with small teams are more likely to rank software requirements as somewhat less important than respondents who are involved with larger projects and larger teams. The one exception to this trend is that respondents working with very large teams also rank software requirements as less important. The relationship between time spent working on software development and rank of importance of software requirements is less clear. Nevertheless, the data show that people who reported the lowest rank for software requirements spend significantly less time developing software than do people who reported the highest rank.

*Importance of software design*: Results of the ANOVA show that the mean rank of the importance of software design differed between respondents engaged in different project sizes and in different team sizes. In general, scientists working on small projects with less than 5000 LOC and in teams of one or two people ranked software design as less important than did people working on larger projects in bigger teams. Scientists who ranked the importance of software design highest spend significantly more time on developing software than those who reported lower ranks of importance.

*Importance of software construction*: The ranking of this concept was generally high overall. Compared to large and small projects, scientists working on middle-sized projects ranked this concept higher (rank 'very important') but there was no difference in rank by team size. Time spent developing software tended to increase with importance rank of this concept—in particular, scientists who ranked this concept 'very important' spend significantly more time developing software than do other scientists.

*Importance of software verification*: Most scientists ranked software verification as somewhat important and this did not differ significantly by either project size or team size. Differences in time spent developing software between rank scores were only marginally significant. There is some evidence that those who ranked verification as 'important' or 'very important' report a higher proportion of their time developing software than those who ranked verification as 'very unimportant'.

*Importance of software testing*: Testing is generally considered important, and differences across groups are statistically significant but small. Importance ranks are significantly lower (but only slightly) among scientists working on smaller projects (less than 5000 LOC) and ranks are slightly higher among scientists in mid-sized teams than larger or smaller teams. Scientists who gave testing a rank of 'not important' (the second lowest rank) spend significantly less time developing software than those who gave testing the highest ranks ('important' and 'very important').

*Importance of software maintenance*: Software maintenance was generally ranked as moderately important. The relationship with project size is somewhat unclear, but the data seems to suggest that rank is lower among scientists who are involved with smaller, rather than larger, projects. Similarly, higher importance ranks are associated with scientists who work with larger, rather than smaller, teams. There is a general, but small, trend where an increase in time spent developing software associates with an increase in rank of importance.

*Importance of software product management*: Although not all pair-wise comparisons are statistically significant

and they are difficult to summarize, the analysis shows a fairly clear trend where an increase in importance of this concept is associated with an increase in both project size and team size. Likewise, the time spent on developing software generally increases with increasing rank of importance for this concept; i.e., scientists who spend more time developing software ranked this concept higher than those who spend less time developing software.

*Importance of software project management*: Software project management was generally ranked as moderate to low importance. The relationship between the ranking and project and team size are very clear. Scientists working on small projects (less than 5000 LOC) and with teams of less than three people ranked this concept low ('not important'), while others ranked this concept slightly higher ('somewhat important'). The relationship between the rank of importance and the time spent developing software is statistically significant but a clear trend is not evident. While the ranked importance of the concept increases with the time spent on developing software for the ranks 'not at all important', 'not important', 'somewhat important', and 'very important', the average time spent on developing software of those who assigned the rank 'important' to this concept is almost as low as that of those who assigned the lowest rank ('not at all important').

## 4. Discussion

We here discuss what we find to be the most relevant implications of our findings. We view the implications from the perspectives of both the scientific computing community and the software engineering community. We also discuss the most pressing threats to validity of our survey.

### 4.1 Importance of Scientific Software

The results for RQ3 of the study show that both developing and using scientific software is of very high importance for scientists' own research. Almost half of the respondents stated that they believe developing scientific software is important for other scientists. While the questions we asked do not allow to draw direct conclusions about the hypothesis that the importance of developing and using scientific software has been increasing over time, the stated belief of scientists that today much larger amounts of data are generated and archived than five and ten years ago may be interpreted as an indirect support for this hypothesis.

Corroborating this evidence, our findings for RQ4 indicate that scientists spend on average about 50% more of their total work time for developing scientific software and 100% more of their total work time using scientific software than expected. Further, our findings for RQ5 clearly support the hypothesis that today, scientists spend more or much more time for for both developing and using scientific software than they did five and ten years ago.

### 4.2 Software Engineering Practices

The results for RQ8 indicate that there is a great deal of variation in the level of understanding of standard software engineering concepts by scientists. The level of importance that scientists assign to a standard software engineering concepts is mostly consistent with their understanding of this concept. We found, however, that in particular for the concepts 'software testing' and 'software verification' scientists assign on average a higher level of importance to these concepts than they judge their level of understanding of these concepts.

Software testing is particularly challenging for scientific software because the answers are known to contain mathematical approximation errors of unknown size. More specifically, the challenge consists in separating software bugs from model errors and approximation errors. Moreover, the correct output from running a simulation code is seldom known. This makes standard testing procedures in software engineering (e.g., regression testing and black-box testing) less appropriate for scientific software in many situations. These facts may be the reason why scientists have instead focused on testing techniques that are based on mathematical insight in the scientific problem being solved. It is quite common, as soon as evidence for a correct code is provided, to use the verified output in regression tests. However, we postulate that scientists have mostly reinvented this concept rather than having imported the technique from software engineering. Thus, although it should be beneficial to teach scientists about software engineering testing techniques, one must be aware that scientific software testing raises issues that have not yet been addressed sufficiently by the software engineering community.

The rationale behind RQ9 was that larger projects and larger development teams might increase the perceived importance of various software engineering practices. The results support RQ9 to some extent (large projects and large teams are associated with higher perceived importance of certain software engineering concepts than are small projects and small teams), but there is no consistent trend of association that links an increase of project or team size to perceived importance of software engineering concepts.

### 4.3 Education and Training

The results for RQ1 and RQ2 support the hypotheses that for both developing and using scientific software, informal self-study and informal learning from peers is clearly more important than formal education at an academic institution or formal training at the work place. Our findings also

suggest that both learning at an educational institution and learning during professional work become more important the more recent they occur.

We postulate that these observation are due to the following: First, there is a general lack of formal training in programming and software development among scientists. Second, the training that scientists do receive is often supplied by a computer science department, which gives general software courses that scientists might not see the relevance of. A third aspect is that scientists may not see the need for more formal training in programming or software development. Codes often start out small and only grow large with time as the software proves its usefulness in scientific investigations. The demand for proper software engineering is therefore seldom visible until it is "too late". As modern scientific software tends to be more complex, there is an increasing awareness among scientists of the need for better development tools and more formal training. However, as suggested by our findings, this awareness arises primarily in larger projects and is difficult to experience in projects met in basic education. Training targeted at practitioners in science and engineering is therefore important.

However, many engineering programs have, in fact, removed programming courses and formal training in, e.g., numerical methods from the curriculum. Since our findings clearly suggest that scientific software is becoming increasingly more important and that an increasing amount of time is spent on both developing and using scientific software, one can ask if such moves are in the right direction. The answer lies in what further in-depth studies can tell us about how scientists and engineers use and develop software.

### 4.4 Scale

Our findings for RQ7 partly support the hypothesis that most scientific software is either used by a very large number of people (more than 5000 users) or by a very small number of people (less than three). The size category 'more than 5000 users' received clearly the largest number of responses (i.e., consistently greater than 50% of all responses for the four top most important pieces of software used).

That scientists rely mostly on software with a large user base may come as a result of an increasing number of (1) commercial packages, (2) open source projects, and (3) community efforts in establishing common software bases. Scientists' need for programming in such contexts is often restricted to smaller problem-dependent code for problem specification. The mix of personal, problem-dependent code interacting with a larger, more general software framework brings forward some challenges with testing and debugging (see above), since the scientist does not have complete control of all details. This contrasts to the past when scientists often had complete control when writing complete codes themselves. This, by the way, is still a dominating principle in university education.

Regarding RQ6, the results of the study support the hypothesis that the majority of scientists use desktop computers most of the time for developing and using scientific software. The results of the study did not confirm the hypotheses that less than 10% of scientists ever use a supercomputer for either developing or using scientific software. Actually, almost a quarter of scientists use supercomputers for developing scientific software sometimes, and about a fifth use scientific sometimes on a supercomputer. However, very few spend more than a fifth of their time developing or using scientific software on a supercomputer.

Many supercomputer centers have a significant staff for helping scientists with software issues, and the competence transfer from such help cannot be underestimated. Nevertheless, taking into account the importance of desktop and small-cluster computing indicated by this survey, one should consider allocating more resources for improving the scientists' competence in development techniques and tools related to desktop and small-cluster computing.

### 4.5 Threats to Validity

Every empirical study will have shortcomings. Here, we discuss the most pressing issues for our study.

*Construct validity* pertains to how well the measures in an empirical study reflect the concepts under investigation, and also to how well-defined the concepts are. In our study, this translates to how meaningful our research questions are, how appropriate the derived hypotheses are, and to what extent the survey questionnaire items were appropriate for giving answers to the hypotheses and research questions. We made efforts to follow standard guidelines for designing survey questionnaires, e.g., [4], and we ran the questionnaire as a pilot study in the field as a means to validate the questionnaire items. Nevertheless, since this was a first attempt, there are several items that may be improved for future applications of this survey.

*External validity* concerns the extent to which conclusions drawn on the study's specific operationalizations transfer to variations of these operationalizations [10]. Here, this pertains to how well our conclusions transfer to other prospective respondents. As indicated in Section 3.1 scientists from certain regions were not represented in any great numbers. It is therefore, unclear how well our conclusions transfer to scientists in these, and other, regions.

*Statistical conclusion validity* pertains to the conclusions drawn from the statistical analyses, and the appropriateness of the statistical methods used in the analyses. With regards, to the latter, the assumption for ANOVA is that the population distribution is normal, with the same standard deviation, for each group. In our case, we have no *a pri-*

*ori* knowledge of how the subgroup population distributions are. Nevertheless, our large sample size justifies the use of the ANOVA in estimating the means of whichever population distribution the groups may have, and at the present state of knowledge, means seem to be a sensible summary expression of a group's response on a variable. With regards to the conclusions drawn, our large sample size gives the statistical analyses an adequate power to show effects. However, it should be noted that highly significant effects are not necessarily large effects. We took this into account in our reporting in Section 3.2.

## 5. Related Work

To our knowledge, there are few publications in software engineering that focus on the development of scientific software. Smith, who is a computational scientist, has addressed software engineering topics in several works, e.g., [11]. Basili et al. [2] discussed the potential interplay of scientists and software engineering in the context of codes for high-performance computing on supercomputers. Sanders and Kelly [7] interviewed scientists at two universities to better understand development practices. Greenough and Worth [5] reviewed scientific software development practices and based much of their information on an extensive questionnaire. Tang [12] conducted a survey quite similar to the present one to investigate factors such as educational background, working experience, group size, software size, development practices, and software quality.

## 6. Future Plans

Science is very diverse and different fields of science use and develop software in different ways. A more refined analysis, where various subgroups of scientists are addressed separately, constitutes an obvious and necessary improvement of the present analysis. In several instances, the underlying rationale for responses are not always clear, and we consider follow-up interviews with selected respondents to clarify such issues. For example, the interpretation of what a standard software engineering concept actually is, might vary among respondents. Furthermore, when we know the answers to our research questions, it becomes possible to change practices and migrate the relevant software engineering knowledge to the science field.

## Acknowledgments

The authors wish to thank Laurel Duquette, Statistical Consulting Service University of Toronto, for support in doing the statistical analyses. This work was funded in part by a grant from The MathWorks.

## References

[1] Advertisement for survey. *American Scientist*, page 445, November/December 2008.

[2] V. R. Basili, D. Cruzes, J. C. Carver, L. M. Hochstein, J. K. Hollingsworth, and M. V. Zelkowitz. Understanding the high-performance computing community: A software engineer's perspective. *IEEE Software*, 25(4):29–36, July/August 2008.

[3] 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 *Proc. International Conference on Software Engineering*, pages 550–559, 2007.

[4] F. J. Fowler, Jr. *Survey Research Methods*. Sage, third edition, 2002.

[5] C. Greenough and D. J. Worth. Computational science and engineering department software development best practice. Technical report ral-tr-2008-022, SFTC Rutherford Appleton Laboratory, 2008.

[6] R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July/August 2008.

[7] R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July/August 2008.

[8] J. Segal. When software engineers met research scientists. *Empirical Software Engineering*, 10(4):517–536, 2005.

[9] J. Segal and C. Morris. Developing scientific software. *IEEE Software*, 25(4):18–20, July/August 2008.

[10] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin, 2002.

[11] W. S. Smith, L. Lai, and R. Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing (Special Issue on Reliable Engineering Computation)*, 13:83–107, 2007.

[12] J. Tang. Developing scientific computing software: Current processes and future directions. Master's thesis, Department of Computing and Software, Faculty of Engineering, McMaster University, 2008.