Software Engineering Education for Bioinformatics

Medha Umarji, Carolyn Seaman,
A. Gunes Koru
Department of Information Systems,
University of Maryland Baltimore County
Baltimore, MD 21250 USA
{medha1, gkoru, cseaman}@umbc.edu

Hongfang Liu
Department of Biostatistics
Bioinformatics and Biomathematics
Georgetown University Medical Center
Washington, DC 20007 USA
hl224@georgetown.edu

Abstract

As software engineering educators, it is important for us to realize the increasing domain-specificity of software, and incorporate these changes in our design of teaching material. Bioinformatics software is an example of immensely complex and critical scientific software and this domain provides an excellent illustration of the role of computing in the life sciences. To study bioinformatics from a software engineering standpoint, we conducted an exploratory survey of bioinformatics developers. The survey had a range of questions about people, processes and products. As software engineering educators, we realized that the survey results had important implications for the education of bioinformatics professionals. We also investigated the current status of software engineering education in bioinformatics, by examining the curricula of more than fifty bioinformatics programs and the contents of over fifteen textbooks. We observed that there was no mention of the role and importance of software engineering practices essential for creating dependable software systems. Based on our findings and existing literature we present a set of recommendations for improving software engineering education in bioinformatics.

1. Introduction

Bioinformatics can be formally defined as, "any application of computational methods to biological problems" [1]. Due to the critical and complex nature of bioinformatics software and its growing volume, there is a strong need to support bioinformatics professionals in developing reliable and maintainable software systems. In addition, this domain is fundamentally different in some aspects to the general software engineering community. First, in bioinformatics software projects, the need to investigate sophisticated research questions is often the main driver of software requirements, rather than a more generic business function [2]. As a result, requirements can be complex, vague, and volatile, which presents an important risk for bioinformatics software efforts. Secondly, the tight budgets and schedule constraints of typical research projects can create additional constraints for development; for example, the resources for appropriate verification and validation can be limited [3]. Finally, bioinformatics developers, who may lack a formal software engineering background, are often in a position to create and maintain their own programs, i.e. there is a high proportion of end-user programmers in bioinformatics [4].

In an effort to better understand and thus to study the "bioinformatics software development community", we conducted an exploratory survey of bioinformatics developers. The findings of the survey study especially those pertaining to requirements engineering, extreme programming and documentation, led to a curiosity about how these professionals are taught to create and maintain such complex software. Hence we studied the syllabi and contents of bioinformatics programs in universities across the

United States, using a list compiled by the International Society of Computing Biology (ISCB) [5]. We observed that there was a consistent effort to include the study of computer science courses such as design and analysis of algorithms, databases and programming languages in all the bioinformatics programs [6-8]. There was, however, little or no training given to these students on basic software engineering principles.

Considering the importance of quality and maintainability of bioinformatics software, it is important to teach software engineering principles to bioinformatics programmers. The results of our survey, a study of existing learning material, and an extensive literature review inform the design of a learning unit with specific software engineering knowledge areas that would be useful to include in a typical bioinformatics curriculum.

2. Survey of bioinformatics software developers

As described above, we created and deployed a survey to gain an understanding of the bioinformatics software development community. It is important to note, at this point, that this survey was not intended to explore education-related issues in bioinformatics, but was aimed at characterizing software development and informing further software engineering research in this domain. The insights about education that resulted, however, were striking, and hence we discuss them here.

The survey instrument was created and distributed online. We contacted bioinformatics developers subscribed to Open Source Software mailing lists found at the Open Bioinformatics Foundation website (www.open-bio.org). The survey details can be found at http://userpages.umbc.edu/~medha1/bio/bioinfswengsurvey.htm. We posted the survey on the mailing lists in August 2005. It was removed after two months, when the flow of additional responses appeared to stop. After blank responses were deleted, we had a valid sample of 126 respondents. It is indeed quite difficult to calculate the response rate as the survey was sent out to mailing lists of unknown size, and unknown overlap. We have done several analyses intended to approximate the response rate. We have also compared early and late respondents, and performed other analyses to investigate the possibility of bias in the sample. For a complete discussion of these analyses please refer to: http://userpages.umbc.edu/~medha1/bio/RR.htm. The final conclusion of these analyses is that the approximate response rate was around 28% for our survey.

Background information was solicited to help us understand the variety of developers and projects in this domain. When asked about the method by which they gained software development skills, 84% of respondents indicated that they had learned through self-teaching alone, or that self-teaching was one of their main modes of learning. About 70% responded that they had taken some computer science courses. Ten percent of the respondents completed certification programs to gain proficiency in software development. This result is interesting because it seems that self-teaching is prevalent in this domain. The percentage of respondents having a graduate or professional degree was also 84%. This finding indicates a high level of education among bioinformatics software professionals and mirrors the fact that there is a high entry barrier in this domain as until recently only graduate level classes were offered. Another survey item asked respondents whether they would describe themselves as computer programmers working in bioinformatics-related projects, or as bioinformatics professionals doing software development. About 57% of the respondents identified

themselves primarily as bioinformatics professionals, while another 35% identified as professional programmers.

While most respondents (85%) had worked on projects larger than 5KLOC, 63% had never worked on anything larger than 20KLOC and 15% had worked only on projects smaller than 5KLOC. Respondents were also asked about the number of people they typically worked with on a project. Just over half (51%) of the respondents worked in teams with less than 5 people. Also, 22% work in groups of 5 to 20 people and 20% work alone. Thus, although there is a strong indication of teamwork, there is also a sizable chunk of respondents who work alone. (The rest of the survey results are discussed in Section 4).

3. The current role of software engineering in bioinformatics education

After several years of academic programs that included a few bioinformatics topics, Russ Altman illustrated the need for a formal bioinformatics educational program in 1998 [9]. Until recently the entry barrier for students in this domain was quite high, due to the availability of only graduate level courses and the need for a strong background in at least two disciplines [7]. Some studies such as that by Doom et al [7] proposed a syllabus for a baccalaureate option in bioinformatics, that would help to churn out professionals faster, and meet the growing demand for bioinformatics professionals in the government and industry. However, there continues to be a lot of discussion about the composition of a bioinformatics program that would enable students to keep up with the advances in bioinformatics [6]. Ranganathan [10] points out that it is almost impossible to design an educational program that satisfies the "wish list" of skills that a bioinformatics professional should possess.

We acquired a list of bioinformatics programs within the United States from ISCB (http://iscb.org/univ_programs/program_board.php), and browsed through each program's website. We considered all the bioinformatics, computational biology, biomedical informatics and related programs on the list. We focused on the computer science-related coursework and biology-related courses were beyond the scope of our work. Currently, each bioinformatics program includes at least one computer science course, and some require a major or minor in computer science. The most common categories of computer science related courses that we observed were: a) design and analysis of algorithms, b) object-oriented programming, c) database programming, and d) data mining and visualization. From prior literature, other bioinformatics educators also mention that skills in programming, databases, artificial intelligence and algorithms are absolutely essential [6, 7].

Next, we studied the presence or absence of software engineering courses. In all out of a total of 79 program offerings, there were only 2 instances where a software engineering related course was a required part of the curriculum. As an elective, software engineering featured 13 times in the degree-program combinations. Also, there was no mention of the role and importance of software engineering in the curricula. The detailed analysis of program listed on the ISCB website can be found at: http://userpages.umbc.edu/~medha1/bio/BioinformaticsPrograms.htm.

We also examined closely the tables of contents of 15 bioinformatics textbooks that focused on computing approaches to solving bioinformatics problems. All the tables of contents were examined in detail online, as it was not feasible to go through each and every book. Again, we made note of the topics commonly covered in such textbooks. It was evident that there was no information about software engineering concepts within

the bioinformatics textbooks. There was some emphasis on design patterns in the object-oriented programming-related books, but there was nothing about, for example, techniques for prototyping, testing and quality assurance in bioinformatics projects. The entire list of textbooks can be found at the ISCB website: http://www.iscb.org/bioinformaticsBooks.shtml.

4. Design of a learning unit based on survey results

Due to the interdisciplinary nature of bioinformatics, students and professionals are overloaded with information. For this reason, we do not recommend a separate course in software engineering, but instead suggest the inclusion of selected software engineering principles in a bioinformatics programming textbook, so that self-taught programmers can have access to this knowledge. Also, it would be advisable to incorporate these concepts into assignments and design class projects around them. We would estimate that the amount of time devoted to such a learning unit would be at least 10 percent of the total course time (in terms of both lecture and lab time). A similar space allotment would be reasonable for a textbook. Over and above teaching all these topics, self-learning should be encouraged, as it is the culture of this domain. In this section we present details of the survey results pertaining to specific software engineering topics, the supporting literature, and suggested content for a learning unit.

4.1 Approaches to Software Development

Survey results: When we asked the respondents to indicate all the general software development processes they followed, the most common responses were prototyping (40%) and extreme programming (29%), followed by evolutionary model of development (23%). As well, 25% of the respondents did not use any specific software development methodology. We had also asked respondents to indicate whether or not they were using a range of XP practices (which we derived from the XP literature). The most popular ones were *starting with a rough project plan* (49%) and *keeping designs as simple as possible* (46%). It was interesting that XP practices were not all used as is recommended by the literature [11] and this finding points to the possibility that the XP label may not be fully understood, or that a different mix of XP practices are applicable to bioinformatics. Respondents were asked to rate the frequency with which they used inheritance, encapsulation, and polymorphism. Professional programmers used encapsulation and polymorphism significantly more than bioinformatics programmers (Mann-Whitney U test $p=.00003$ and $p=.014$, respectively). We did not observe such a difference in the use of inheritance (a similar test gives $p=.23$). The power of object-oriented programming is based on using inheritance "together with" encapsulation and polymorphism. This result indicates that bioinformatics programmers may not be taking full advantage of the benefits of OO programming (e.g. modifiability). Therefore, we suggest that they can be better trained to use these two fundamental OO principles.

Supporting literature: Experience reports on developing bioinformatics software by Kane et. al. [25] emphasized that extreme programming and the agile philosophy were indeed well suited to their development practices. A similar finding was reported Kendall et al [12] in their development of weather forecasting software. Bioinformatics toolkits such as BioJava [13] are based on the object-oriented framework, and are very popular, but there is little information on how programmers leverage these toolkits in their work.

Suggested content: Each software development paradigm can be discussed in some detail along with advantages and disadvantages so that bioinformatics programmers can

make an informed choice. For example, rapid application development or prototyping is a great way to get a tangible solution to the customer, but the drawback is that the prototype ends up being used as the actual system, which later results in problems. As discussed earlier, future bioinformatics software developers should have complete knowledge of XP and related paradigms such as Test-Driven Development (TDD), and how to use the right mix for a successful project. Object-oriented concepts should be taught with rich examples and plenty of exercises.

4.2 Importance of Documentation

Survey results: We asked whether developers would perceive writing an *increased number of comments and documentation* to be beneficial, in terms of maintenance. For increased comments, 61% of opinions were in the range "slightly beneficial" to "extremely beneficial". On the other hand, for increased documentation, 90% were in that range. According to a Mann-Whitney U test between the two distributions, increased documentation was perceived to have significantly more potential benefits compared to increased commenting ($p=0.000$ and $Z\text{-value}=-6.303$). Next, we asked developers to indicate all types of documentation they used, from: Design Documentation, Requirements Specification, User guides, Test Cases, Systems Description, and an "other" category was provided. The most frequently chosen type of documentation was User Guides (almost 70%), followed by 56% using System Descriptions.

Supporting literature: Bioinformatics software is complex, has a high proportion of end-user programmers [4], and is constantly evolving. Therefore, documentation is very important to developing software for bioinformatics. Documentation also performs the function of embedding domain knowledge within source code [14]. The importance of documentation was stressed by the researchers of Bioconductor project [15]. Baxter et al. [16] also cite documentation as a key practice to be strengthened in scientific software development.

Suggested content: In addition to explaining the costs and benefits of commenting, alternative successful techniques could be taught, such as: descriptive variable names, refactoring in lieu of comments, and functional descriptions. Tips for creating and maintaining documents such as user guides and system descriptions should also be discussed in detail. Advantages of having updated documentation such as clear system understanding and better maintainability can be emphasized, along with a cost-benefit analysis of immediate time spent on documentation and how it is time well invested for future maintainability.

4.3 Quality Assurance Practices

Survey results: Forty percent of respondents were of the opinion that their software was not tested adequately and about 25% did not agree that their software was maintainable. These results point to a need for improvement in quality and maintainability practices in bioinformatics software projects.

Supporting literature: Bioinformatics research and practice has critical implications for life sciences, and it is absolutely essential to have strong quality assurance (QA) practices such as code reviews and testing to ensure software quality. Heusden suggests that the open source development model used for many bioinformatics products can have weaknesses in terms of quality assurance [17] because it is often the case that only some portions of bioinformatics source code are reviewed frequently. Also, in a survey done by Koru, El-Emam et al [18] testing and code review practices were found

severely lacking, particularly in smaller sized projects. Other researchers of scientific software development [19] have also suggested strengthening the QA practices of scientific software development in general.

Suggested content: It would also be useful to have a step-by-step tutorial about how to write a test case, as well as how to perform a code review. Class exercises and projects could also be used to train students in QA practices. Students could write their programs complete with documentation and then interchange their code with another person in the class and perform a thorough code review. A grade incentive could be tied to number of bugs found. It would be worthwhile to have them join an open source community of their choice and contribute test cases to a project. This would strengthen their understanding of test cases.

4.4 Software Evolution and Maintenance

Survey results: A majority (65%) of developers agreed that complexity of software modules increases, as age of the software product increases. When asked whether complex modules tend to be more change-prone, and if change-proneness of a module increased with age, more than half of the participants responded Neutral for both these questions (61% and 58%, respectively). We interpret the reporting of "Neutral" to indicate that many of the respondents were unsure about the existence of such relationships, either due to lack of experience or due to lack of awareness of the theoretical basis behind such relationships [20].

Supporting literature: Bioinformatics is still a very young field, and software developed in this field has not yet matured enough to be studied from an evolutionary perspective. As their applications move into legacy status, bioinformatics programmers need to know more about the complex relationships between software size, complexity and age, so that they can take preventive measures in advance. Bioinformatics software development is a *means to an end*, not an end in itself, like software developed in the IT industry [21]. Hence developers care less about the future of the software they are writing, and more about the science-related problem at hand. The same is true of other types of scientific software [22].

Suggested content: It would be beneficial to educate bioinformatics students about the causes and effects of these relationships. Additionally, through class assignments designed to span multiple software modules, students could be shown how a single change could impact a large number of modules. In addition to a lecture on relationships between software aging, complexity, etc., we recommend bringing in an outside speaker who can recount "war stories" of software bugs difficult to catch due to system aging, complexity or churn.

4.5 Requirements Engineering/ Management

Survey results: Respondents were asked to indicate all the requirements gathering techniques they used. Responses show a high use of interviewing (78%) and reading related documents (70%). What is somewhat surprising is that 40% of the respondents indicated that they often have sufficient bioinformatics-related information beforehand that they do not have to spend any time gathering requirements. Also, 80% of respondents indicated that they examined similar applications to gather requirements.

Supporting literature: Managing requirements in the bioinformatics domain is a challenging task [21]. In bioinformatics, requirements cannot simply be "handed off" from the domain experts to the degree that is possible in other disciplines. Close cooperation between domain scientists and professional programmers is necessary in

order to keep up with changing hypotheses, new algorithms and new methods for handling vast quantities of data [4]. Based on these and other factors discussed earlier, we conclude that traditional software engineering approaches to requirements management might not be appropriate as-is for bioinformatics. A deeper study of their current requirement gathering practices needs to be done, in order to be able to teach bioinformatics students the right set of practices.

Suggested content: Agile requirements gathering techniques such as user stories could be taught, since user stories model the changing requirements well. Also, standard techniques such joint application development type of requirements sessions could be included in the curriculum. It would be useful to provide training in interviewing techniques that can elicit both domain and system knowledge.

5. Conclusions and limitations

To our knowledge, this is the first survey study to inquire about the software development practices of a domain-specific scientific software community. As such a first effort, it has several limitations. One is that mailing lists of OSS projects were used, which can bias the sample. However, our analysis of responses indicates that a significant number of responses were from developers on proprietary software projects. Also, it is to be noted that most of the literature in bioinformatics indicates that the OSS model is widely used, so our sampling strategy is likely to characterize this domain accurately. In our study we considered only bioinformatics programs in the US and an analysis of bioinformatics programs in other countries is outside the scope of our work.

Our proposed learning unit may be missing important elements because the survey did not address those issues. For example, other software engineering challenges faced by bioinformatics are integration and presentation of data, as discussed by Barker and Thornton [23]. Software reuse is also an important issue as different scientists may be working on different (or similar) aspects of the same problem [24]. Although we did not include these topics in our survey, bioinformatics educators should be mindful of these concepts while teaching software engineering.

This paper compares bioinformatics software education and practice and identifies some areas where they can be better bridged. It is incumbent upon us to make available the right software engineering material and market the importance of software engineering and its impact on bioinformatics software. Our learning unit is meant to act as a prototype/precedent for including software engineering education in other domains. The next step would be to collaborate with bioinformatics educators and pilot these recommendations in a real bioinformatics programming class.

References

[1] D. Counsell, "A review of bioinformatics education in the UK," *Briefings in Bioinformatics*, vol. 4, pp. 7-21, 2003.

[2] J. Segal and C. Morris, "Developing Scientific Software," *IEEE Softw.*, vol. 25, pp. 18-20, 2008.

[3] R. Sanders and D. Kelly, "Dealing with Risk in Scientific Software Development," *Software, IEEE*, vol. 25, pp. 21-28, 2008.

[4] C. Letondal and W. Mackay, "Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment," in *Proceedings of the eighth conference on Participatory design: Artful integration: interweaving media, materials and practices - Volume 1*. Toronto, Ontario, Canada: ACM, 2004.

[5] International Society for Computational Biology, "ISCB listing of degree/certificate programs worldwide," International Society for Computational Biology, La Jolla (California) October 5, 2005.
[6] D. T. Burhans and G. R. Skuse, "The role of computer science in undergraduate bioinformatics education," *SIGCSE Bull.*, vol. 36, pp. 417-421, 2004.
[7] T. Doom, M. Raymer, D. Krane, and O. Garcia, "A proposed undergraduate bioinformatics curriculum for computer scientists," in *Proceedings of the 2002 ACM Special Interest Group on Computer Science Education (SIGCSE 2002)*. Covington, Kentucky, 2002.
[8] B. Hemminger, T. Losi, and A. Bauers, "Survey of bioinformatics programs in the United States," *Journal of the American Society for Information Science and Technology*, vol. 30, pp. 12-13, 2005.
[9] R. B. Altman, "A curriculum for bioinformatics: The time is ripe.," *Bioinformatics*, vol. 14 pp. 549–550, 1998.
[10] S. Ranganathan and e52., "Bioinformatics education—Perspectives and challenges," *PLoS Comput Biol*, vol. 1, 2005.
[11] K. Beck, *Extreme programming explained: embrace change*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 1999.
[12] R. Kendall, J. C. Carver, D. Fisher, D. Henderson, A. Mark, D. Post, C. E. Rhoades, and S. Squires, "Development of a Weather Forecasting Code: A Case Study," *Software, IEEE*, vol. 25, pp. 59-65, 2008.
[13] M. Pocock, "BioJava Toolkit Progress," in *Bioinformatics Open Source Conference*, 2002.
[14] K. S. Ackroyd, S. H. Kinder, G. R. Mant, M. C. Miller, C. A. Ramsdale, and P. C. Stephenson, "Scientific Software Development at a Research Facility," *Software, IEEE*, vol. 25, pp. 44-51, 2008.
[15] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry, K. Hornik, T. Hothorn, W. Huber, S. Iacus, R. Irizarry, F. Leisch, C. Li, M. Maechler, A. J. Rossini, G. Sawitzki, C. Smith, G. Smyth, L. Tierney, J. Y. H. Yang, and J. Zha, "Bioconductor: open software development for computational biology and bioinformatics," *Genome Biology*, vol. vol. 5, 2004.
[16] S. M. Baxter, S. W. Day, J. S. Fetrow, and S. J. Reisinger, "Scientific Software Development is Not an Oxymoron," *PloS Computational Biology*, vol. 2, 2006.
[17] P. v. Heusden, "Applying software validation techniques to Bioperl," in *Bioinformatics Open Source Conference*, 2004.
[18] A. G. Koru, K. El-Emam, A. Neisa, and M. Umarji, "A survey of quality assurance practices in biomedical open source projects," *Journal of Medical Internet Research*, vol. 9, pp. e8, May, 2007.
[19] S. M. Baxter, S. W. Day, J. S. Fetrow, and S. J. Reisinger, "Scientific Software Development is Not an Oxymoron," *PloS Computational Biology*, vol. 2, September, 2006.
[20] A. G. Koru and J. Tian, "Comparing High-Change Modules and Modules with the Highest Measurement Values in Two Large-Scale Open-Source Products," *IEEE Transactions on Software Engineering*, vol. 31, pp. 625-642, 2005.
[21] J. Segal, "When Software Engineers Met Research Scientists: A Case Study," *Empirical Softw. Engg.*, vol. 10, pp. 517-536, 2005.
[22] M. Vigder, N. G. Vinson, J. Singer, D. Stewart, and K. Mews, "Supporting Scientists' Everyday Work: Automating Scientific Workflows," *Software, IEEE*, vol. 25, pp. 52-58, 2008.
[23] J. Barker and J. Thornton, "Software Engineering Challenges in Bioinformatics," in *International Conference on Software Engineering (Keynote address)*. Edinburgh, Scotland, UK, 2004.
[24] L. Stein, "Bioinformatics: Gone in 2012.," in *O'Reilly Bioinformatics Technology Conference (Keynote Address)*. San Diego CA, 2003.
[25] D. Kane, "Introducing Agile Development into Bioinformatics: An Experience Report," *Conference on Agile Development*, Washington DC, USA 2003.