EPiC Series in Computing

Volume 88, 2022, Pages 8–17

Proceedings of 31st International Conference
on Software Engineering and Data Engineering

Challenges of Software Engineering in Computational Biology and Bioinformatics Scientific Software Projects

Tamer Aldwairi[1,2*]
[1] Mississippi State University Starkville, MS 39762, USA
[2] Temple University, Philadelphia, PA 19122, USA
aldwairi@temple.edu

Abstract

Computational biology scientific software projects are continuously growing and the volume and the task of analyzing, designing, implementing, testing, and maintaining these projects to ensure high-quality software products are only getting harder and more complicated. Conventional software development methodologies are not sufficient in ensuring that scientific software is error-free or up to the standard or comparable to the software designed in the industry. For this reason, it is important to investigate projects that utilized the best software engineering practices during their development and find and understand the problems that arise during the development of those projects. Such understanding will serve as the first step in the process of developing high-quality software products and will enable us to design and propose solutions to the problems that commonly occur during the development of such projects. In this paper, we will discuss different studies that applied software engineering practices and approaches in their computational biology projects. The challenges they encountered and the benefits they gained from employing software engineering quality assurance and testing techniques. In addition, we will demonstrate some of our own experiences when designing, developing, and testing computational biology projects within academic settings. We will also present, based on our experience, some solutions, methodologies, and practices that when adopted will benefit the scientific computational biology software community throughout the process of designing and testing the software.

* Corresponding Author

1 Introduction

The advancement of next-generation sequencing, microarrays, and other aspects of technologies within molecular biology has had a major effect on the computational field in both industry and academia. If we take a closer look at the recent development in sequencing technologies, we notice a huge jump in the volume and size of data which increased the complexity of dealing with its various features. The consequence of these changes indicates that we should find new ways to extract important and useful information out of these massive volumes of data. Trying to control these trends and shifts in the past and now is tantamount to trying to control the flow of water in a small waterpipe compared to a waterfall. The size of data that is produced on an annual basis is growing at an exponential rate. Most of the software systems available to deal with these data are barely able to keep up with the task of processing all this information [7, 13, 16]. Yet extracting the most useful information from these large volumes of data in an efficient manner might seem like a far-fetched end to achieve.

The large availability of data requires a paradigm shift in the way we think and process the data. Small software systems designed in labs, or a small script written in Perl by a student would not be able to manage such large datasets. The integration of efficient methodologies for data processing and retrieval of large amounts of data within these tools is a major problem in computational biology and in bioinformatics [6, 7]. Data integration will allow for better management and facilitate the system's ability to store and retrieve data in addition to interacting with outside sources within the same field [8].

To cope with such a large amount of data, the development of large biological-oriented software systems (LBOSS) is a must. Developing LBOSS systems comes with a lot of challenges and finding solutions to such problems has been delayed for several reasons. One major reason that has not been addressed until recently is the employment of software engineering techniques in the development of scientific projects, especially in biological-related fields. This view coincides with the study in [12] which showed that software engineering is rarely adopted in computational biology projects. This is because these systems were developed in small labs or written by a student in an academic setting to handle a very specific task, especially software that was developed to cope with a large amount of data and lacks any involvement of software engineering techniques and approaches.

The tests that are usually applied to those projects are not enough to verify and validate that the code works as it should. A very common example is the bugs and errors that can be easily detected when browsing through some of the major websites that were developed originally as part of a research project within academia. These errors should have been detected early in the software process if proper testing methodologies were applied to verify the accuracy of the integration between the software and its data before the site was launched. Some studies reviewed software testing methods and their usage in the bioinformatics field [14] but without comparing the effectiveness of these methods.

In this paper, we will address the challenges software developers face during working on computational biology projects within an academic setting and the different steps required to ensure that those software projects are on par with the software quality level produced by their counterparts in an industrial setting. We will review some studies that employed software engineering practices in their projects. We will also present based on our experience solutions to the challenges that we discussed.

2 Computational Biology Projects that applied Software Engineering Practices

In this section, we will present three papers that took advantage of software engineering practices in developing their computational biology software. The first paper we present dealt with the chaste (Cancer Heart and Soft Tissue Environment) project [1] which used the agile approach in the development of their project. The authors found that when agile approaches were used in scientific projects new properties emerged to the surface that was not there before. These properties represent some of the challenges faced by developers when trying to employ software engineering practices in scientific programming. For example, a test oracle should be well defined before the test, in a normal business setting, but are not always predefined, when it comes to scientific settings. This problem was handled through the use of small unit tests as well as utilizing test-driven development and acceptance test, adding to that the frequent refactoring throughout the whole software development.

The authors argue that unit testing in a scientific setting is more difficult than its counterparts in an industrial setting. They stated that programs can be in one of these three categories numerical algorithms, structures, or models. The programs based on models are the most difficult to test because of the limited types of tests that can be performed on those programs. The problem was dealt with by carrying out heavier tests covering a wider range of components, in addition to testing the final model of the whole system.

The second paper discusses the employment of software engineering approaches in multiple scientific projects [2]. It shows the benefits of involving new and improved practices like quality assurance and testing at six biomedical organizations of various types and with different applications. A strong aspect of the results in this paper is the diversity in team structures that was a consequence of the different types of organizations, commercial, academic, and government involved in these projects.

The projects targeted a wide range of users and had teams of different sizes. The teams also had experience with a set of various approaches. Such a variation in the context among the projects provided the authors with the ability to compare the results achieved when agile practices were used. One important aspect that the agile practices brought to the table was allowing close communication between the developers and the biologists which solved the communication problem.

Communication between teams was common in all six projects, even though these projects applied different approaches and practices of software engineering. They all emphasized the importance of incorporating testing and quality assurance practices such as automated unit testing, continuous integration, and refactoring throughout the development process. Incorporating such testing techniques while using agile methods allowed for continuous change in the projects while preserving and maintaining software quality. The issue of communication between team members will be discussed in detail in the next section.

The third paper describes the process of integrating software engineering testing and quality assurance practices on an ongoing computational biology project using agile development techniques [3]. In this project, an incremental approach combining Extreme Programming [10] and Scrum [11] was used. Software engineering practices such as automated builds and configuration management facilitated the development process and reduced the time needed to integrate and demonstrate the code. They wrote automated tests using JUnit and during integration used MaxQ for their system tests.

Refactoring, continuous integration and code reviews were all integrated into the process. Some of the challenges faced during the process were related to the problem of database synchronization which occurs whenever there are changes to be made to the test data. In addition to that, the User Interface was perceived as hard to test because of the different number of software programs used in the UI layer.

3 Challenges that Face Software Engineering in Computational Biology

The significance of developing accurate scientific software is growing rapidly within different disciplines, especially in fields like computational biology. The implication of verifying and validating the correctness of the results provided by these software programs is of the utmost importance. Incorporating software engineering testing and quality assurance techniques is still relatively new in the development of computational biology projects for the following reasons.

Some computational biology developers come from a biological background rather than a computer science one and as a result, lack knowledge about the software engineering best practices and the different scenarios where they can be applied accurately [15]. The other reason is attributed to the exploratory nature of scientific projects. Researchers in a scientific environment prefer getting a piece of software working to produce fast results so they can explore new paths that might lead to newer insights. Because of this nature scientists spend more time developing software that produces fast results rather than spending the time producing a program that follows software engineering best practices which could have saved them a lot of future development time and would have led to more accurate results.

The importance of software engineering testing and quality assurance practices within scientific programming can be seen clearly when developing computational biology projects. These projects usually involve some sort of string manipulation, the string could be DNA, RNA, or a protein. What really matters is the meaning behind such a string. From a programmer's point of view, a string of random characters has no practical meaning unlike a number representing certain values or any sort of output related to other sciences.

The problem of undefined test oracle is one of the most known problems that are faced in scientific computing. In a normal project when a user inserts an input into a program, he expects to get a certain output, based on such output the user decides whether the program is performing correctly. Such tests are not always applicable to scientific programming especially when we consider certain bio-fields like bioinformatics and computational biology [4]. This is important because it makes the use of automated test oracle classes [5] unknown in advance. Meaning that the programmer does not know in advance what the expected output is and that makes it harder for him to design test cases that suit the project. Designing test cases for scientific software that have nondeterministic behaviors can be supported using machine learning and deep neural network algorithms [5, 25]. This can be of help in cases where designing test oracles is a difficult task.

Another problem that we face in scientific computing, especially in computational biology projects is the processing of information from large Databases through extracting, comparing, or manipulating the data. Not only that but the process of testing those numerous cases is time-consuming especially since we always need to verify the correctness of the data after each step of any procedure that involves data processing. For example, computational biology projects that are concerned with finding certain string patterns and comparing them with each other can be one of the most difficult tasks to verify or

validate the accuracy of such patterns. Since one shift in a single character, during the process of reading or extracting a specific sequence, will change the whole meaning of the extracted sequence and will eventually, affect the whole comparison procedure and any conclusions reached based on that data. In addition to the aforementioned reasons, it is also hard to enumerate all of these tests or at least verify a minimum threshold for a certain number of tests that need to be executed in order to cover the most essential test cases needed to verify that the code is working properly.

One of the major challenges researchers face in scientific programming is the communication problem. Such a problem occurs due to the diverse backgrounds that each group member possesses. Computational biology projects require collaborations between biologists who have deep specialized knowledge in their field and computer scientist who understand how the inner mechanics of software programs are built and tested. It's due to this alliance between those fields that allow us to build software that serves the needs of biologists and the scientific community in general.

The problem occurs when computer scientists and biologists work together, they start finding that their understanding of certain terms used in both fields varies greatly. A worse scenario appears when such miscommunication is not detected early in the process. The solution to such a problem requires close communication between team members with different backgrounds on a daily basis where team members can sit together and observe the inner workings of another team member an approach that resembles pair programming in the agile development process, we discussed this approach in detail in [9] along with case studies that support using agile approaches in planning computational biology projects.

Test coverage is one of the main problems facing developers in computational biology projects. When designing scientific projects, it is important to cover the maximum number of cases possible. The problem that we are facing is that there are many cases in which testing them will take too long. In addition, one of the main characteristics of scientific projects is the lack of time because we need to produce results and then check the validity and significance of these results. Due to the time constraints and the large number of tests required, we believe that designing computational biology projects for scientific purposes should take a different route than the current way these projects are currently handled.

The coverage of the tests is a major problem in computational biology projects. The reason for that depends on the target customer that is benefiting from the software. Most computational biology software is designed to target a certain audience or to fulfill a specific task. This is especially true when designing a program with specific requirements without any consideration for changes in the future. The program will face many problems as it becomes available for public use or reused by another party or institution.

These types of problems occur since the program was tested only using certain specific and very stringent requirements. When the input of the program is modified, despite the change being a little and even if it maintains the general specifications required by the original developer. The program will not work as expected and will not result in the desired output and in many cases, it might even crash.

The challenges of software development in an academic setting can be viewed through the different purposes it serves in comparison to commercial or industrial software. In scientific software, the software is written by researchers for the purpose of exploring new scientific knowledge with scientific publications as the end result. This means that the main goal is to publish a new software with little or no emphasis on maintenance or updating the tools a common example can be seen in miRNA target prediction tools [24]. The main source of documentation for scientific software is usually the paper or

resources associated with the publications. In addition to that, there is a hidden assumption that the users of the software are end-users who are interested in that specific scientific field and are experts in their domain and should know how to use the tool [16]. This in fact could be the biggest contributor to getting wrong results through misusing the software or assuming that the software is going to give the right results without checking the current implementation and testing the software in every single possible way to cover as many cases as possible [13].

4 Challenges and Solutions from Our Experience

Nearly all the papers that described the involvement of software engineering in computational biology projects emphasize the importance of testing, verification, and validation, of the code beyond the normal setting used in standard software engineering projects. The reason for such emphasis lies in the fact that when we are dealing with scientific projects a small error could lead to drastic changes in the results.

Scientific research is all about results and proving that those results are reliable so the researcher can build hypotheses, models, and theories based on that. For such reason accuracy of the data is important within scientific programming projects. To facilitate the process of verifying and validating that our programs work correctly and reflect what the customer is expecting. We designed several small programs that are connected to each other where the output of one program acts as an input to the next program we adopted this approach while working on the following projects [17-22,26], such an approach will reduce or limit the number of tests required by the program itself.

One might suggest that this approach is like unit testing where small components of the program are tested individually before the whole program undergoes release testing. This approach is by far different in several ways and provides added benefits, especially to computational biology scientific projects. It provides scientists with releases much faster that are fully tested and an output that can be analyzed by researchers until other results are provided by the other programs.

This approach also simplifies testing in general and saves execution time. The reason for that can be simply demonstrated if you have a program that needs 3 hours to run, and you need to run 100 tests to ensure that the program is running correctly which is around 300 hours of execution time. While if the program was divided into 3 or 4 programs with the one requiring the longest execution time designed to require the least number of tests this could save a lot of testing time.

Many computational biology projects are designed specifically to serve a certain scientific purpose and are not designed to be reusable. On the contrary, we designed our code to be reusable by employing certain factors that help us to do so. One additional benefit that this approach provides is the ability to insert new parts into the project without having to repeat all the tests for the whole project.

The addition to the pipeline of programs could be addressed in two ways. The first one occurs when the added functionality is not significantly large, in that case, the additional functionality is addressed to one of the programs in the pipeline. In that case, the test cases will be built around that specific program [17-22, 26 - 28]. The second one occurs when the size of the added functionality is significantly large in that case, we design another program using the same factors and insert it within the pipeline [23].

When dividing the project into smaller subprograms this makes it easier to predict the output for a certain test oracle. This could represent some sort of solution to the oracle problem. In addition, it contributes to finding a solution to the boundaries problem since we have many test cases and it is hard to satisfy all those cases. Designing testing boundaries for such a program that will cover all the cases will be much easier when we are working with smaller, less complex, programs that are interconnected together through their input-output relationship.

Our approach also solves the coverage test problem, since the division of the programs facilitates fewer tests to cover that specific program. This ensures that not a single program has been left without tests to ensure coverage of all the basic functionality. In addition, it is more thorough since for the pipeline to execute correctly it requires taking the right output from one program before inputting it into the next one. Any error in one output will result in breaking the chain of the pipeline which serves as an additional testing mechanism for the correctness of the process.

Fault tolerance is a major issue in computational biology because many programs crash just due to a small insignificant change in the input which should be usually handled by the system. A typical example can be seen when trying to run some analysis using one of those programs or websites. The program might require several inputs in the form of string sequences and certain numbers that represent certain thresholds and filtering stringency mechanisms.

The system should be able to detect an error when the data is entered on the spot and not wait until the end allowing the user to continue to enter all the data into the website then failing to generate an output and, in some cases, failing to specify the cause of the error. Such a problem could be handled by applying certain requirements throughout the design and test phases of software engineering. For instance, making sure that the program is designed with the reusability factors so that it can be used by other people utilizing different input types [18]. In addition to including test and quality assurance approaches such as end-user testing, acceptance testing, and refactoring.

One would argue why Test-driven development was not used in our approach. We found that designing such tests is not suitable to verify the correctness of our implementation. The reason for that lies in the fact that no specific or expected pattern can be used to judge the output because extracting sequences that represent DNA, RNA, or a certain protein does not follow a specific pattern.

Test-driven development (TDD) requires programmers to write tests before they write the code which means that the expected output should be known by the programmer in advance before writing the test. The programmer knows that the output is a sequence of characters, but those characters are random and even though the programmer might know what the sequence would look like, they can only do so only when the sequence is compared to the original sequence from which it was originally extracted. This means that the programmer does not have in advance a mechanism to verify or validate the output.

Since we can not specify what our designated output will look like when we choose a random set of inputs. Designing a systematic test procedure will not be useful since we are required to choose several test cases that are uniquely different in their characteristics from their counterparts to cover the largest coverage of tests possible for that program.

5 Conclusion

In this paper, we introduced several approaches and best software engineering practices to implement computational biology projects within academic settings. We presented some papers that addressed this issue and the methodologies, approaches, and practices that they employed throughout the process. We also discussed some of the challenges encountered during the testing phase of those projects and the mechanisms used to solve those challenges. In addition, we discussed our own experience working with computational biology projects and the innovative approach we used to deal with the testing problem faced when employing software engineering practices in computational biology projects.

The approach we used divides the project into a small number of interconnected programs where the output of one or more programs serves as an input for another program. This approach facilitates the testing process of the project by minimizing the number of needed tests to only that specific part which saves a lot of testing time. We believe that there is more room for improvement regarding updating our approach through the employment of software engineering practices, especially the testing and quality assurance activities associated with the integration of such an approach.

References

[1] Pitt-Francis, Joe; Bernabeu, Miguel O.; Cooper, Jonathan; Garny, Alan; Momtahan, Lee; Osborne, James; Pathmanathan, Pras; Rodriguez, Blanca; Whiteley, Jonathan P.; Gavaghan, David J. “Chaste: using agile programming techniques to develop computational biology software,“ Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 366, No. 1878. (13 September 2008), pp. 3111-3136.

[2] D.W. Kane, M.M. Hohman, E. G. Cerami, M. W. McCormick, K.F.Kuhlmman, J. A. Byrd, “Agile methods in biomedical software development: a multi-site experience report”,BMC Bioinformatics, Vol. 7, No. 1. (1 May 2006), pp. 1-12.

[3] D Kane, "Introducing agile development into bioinformatics: an experience report," Agile Development Conference, 2003. ADC 2003. Proceedings of the , vol., no., pp. 132- 139, 25-28 June 2003.

[4] Tsong Chen, Joshua Ho, Huai Liu, Xiaoyuan Xie, “An innovative approach for testing bioinformatics programs using metamorphic testing”, BMC Bioinformatics, Vol. 10, No. 1. (2009), 24.

[5] V. A. de Santiago Júnior, "A Method and Experiment to evaluate Deep Neural Networks as Test Oracles for Scientific Software," 2022 IEEE/ACM International Conference on Automation of Software Test (AST), 2022, pp. 40-51.

[6] Maarala, I. "Scalable computational methods for high-throughput sequencing data analytics in population genomics." (2021).

[7] Srivastava, A., Naik, A. (2021). Big Data Analysis in Bioinformatics. In: Singh, V., Kumar, A. (eds) Advances in Bioinformatics. Springer, Singapore. https://doi.org/10.1007/978-981-33-6191-1_22

[8] Alfonsi, Tommaso, Pietro Pinoli, and Arif Canakoglu. "High Performance Integration Pipeline for Viral and Epitope Sequences." BioTech 11, no. 1 (2022): 7.

[9] Aldwairi, Tamer. "Planning Computational Biology Projects Using Agile Approach." *Proceedings of 34th International Conference on Computers and Their Appoications. 1-7.* (2019).

[10] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, Boston, 1999.

[11] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum, ISBN 978-0130676344, Prentice-Hall, 2001.

[12] M. Umarji, C. Seaman, A. Koru, H. Liu, "Software Engineering Education for Bioinformatics," Software Engineering Education and Training, 2009. CSEET '09. 22nd Conference on, vol., no., pp.216-223, 17-20 Feb. 2009.

[13] Meher, J. (2021). Potential Applications of Deep Learning in Bioinformatics Big Data Analysis. In: Prakash, K.B., Kannan, R., Alexander, S., Kanagachidambaresan, G.R. (eds) Advanced Deep Learning for Engineers and Scientists. EAI/Springer Innovations in Communication and Computing. Springer, Cham. https://doi.org/10.1007/978-3-030-66519-7_7

[14] K. H. Amir, et al. "How to test bioinformatics software?." Biophysical reviews vol. 7,3 343-352, 2015. doi:10.1007/s12551-015-0177-3

[15] Peter Georgeson, Anna Syme, Clare Sloggett, Jessica Chung, Harriet Dashnow, Michael Milton, Andrew Lonsdale, David Powell, Torsten Seemann, Bernard Pope, "Bionitio: demonstrating and facilitating best practices for bioinformatics command-line software", GigaScience, Volume 8, Issue 9, September 2019, doi.org/10.1093/gigascience/giz109

[16] Mangul, Serghei et al. "Challenges and recommendations to improve the installability and archival stability of omics computational tools." PLoS biology vol. 17,6 e3000333. 20 Jun. 2019, doi:10.1371/journal.pbio.3000333

[17] Aldwairi T, Chevalier DJ, Perkins AD. Exploring the Effect of Climate Factors on SNPs within FHA Domain Genes in Eurasian Arabidopsis Ecotypes. Agriculture. 2021; 11(2):166. https://doi.org/10.3390/agriculture11020166

[18] Aldwairi, T., Elam, B., Hoffmann, F., & Perkins, A. D. RepCalc: a Tool For Calculating Transposable Element Density within the Genome. Proceedings of the 34th International Conference on Computers and Their Applications (BICOB) 2018

[19] T. Aldwairi, B. Nanduri, M. Ramkumar, D. Gautam, M. Johnson, A. D. Perkins. "Statistical Methods for Ambiguous Sequence Mappings". In Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics (BCB'13), Association for Computing Machinery, New York, NY, USA, 674–675, 2013. DOI:https://doi.org/10.1145/2506583.2506678

[20] T Aldawiri, et al. "A Novel Approach for Mapping Ambiguous Sequences of Transcriptomes." In Proceedings of 14th International Conference on Bioinformatics and Computational Biology. Vol. 83, pp. 76-85. 2022.

[21] Aldwairi T, Hoffmann F, Perkins AD. Prediction Of Novel Pirna Rat Clusters Based On Mouse Pirna Clusters Using Downstream and Upstream Analysis. InProceedings of 11th International Conference 2019 Mar 18 (Vol. 60, pp. 190-199).

[22] Aldwairi, Tamer Ali, "Computational Methods for Solving Next Generation Sequencing Challenges" (2014). Theses and Dissertations. 1140.
https://scholarsjunction.msstate.edu/td/1140.

[23] Al-Agtash, Salem Y., et al. "Re-Engineering BLUE Financial System Using Round-Trip Engineering and Java Language Conversion Assistant." Software Engineering Research and Practice. (pp. 657-663) 2006.

[24] Aldwairi TA, Weerasinghe KS, Yan L, Perkins AD. Comparison of miRNA Target Prediction Algorithms Using Computational Methods. . In Proceedings of the 4th International Conference 2014.

[25] Maurizio Leotta, Dario Olianas, Filippo Ricca, A large experimentation to analyze the effects of implementation bugs in machine learning algorithms, Future Generation Computer Systems, Volume 133, 2022, Pages 184-200, ISSN 0167-739X, https://doi.org/10.1016/j.future.2022.03.004.

[26] Aldwairi, T., Perera, D., & Novotny, M. A. An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection. *Computer Networks*, *144*, 111-119, 2018.

[27] Aldwairi T, Perera D, Novotny MA. Measuring the Impact of Accurate Feature Selection on the Performance of RBM in Comparison to State of the Art Machine Learning Algorithms. *Electronics*. 9(7):1167, 2020. https://doi.org/10.3390/electronics9071167

[28] Aldwairi T, et al. An Investigation of the Role of Feature Selection on the Classification Performance of Machine Learning Algorithms. In Proceedings of the 33rd International Conference on Computers and Their Applications (CATA). 2018.