Lack of Software Engineering Practices in the Development of Bioinformatics Software

Dhawal Verma, Jon Gesell, Harvey Siy, Mansour Zand
Department of Computer Science
University of Nebraska at Omaha
Omaha, Nebraska 68182
Email: {dverma,jgesell,hsiy,zand}@unomaha.edu

***Abstract*—Bioinformatics is a growing field in the software industry. However there is very little evidence that sound software engineering practices are being applied to bioinformatics software development. As bioinformatics is a merging of the disciplines of biology and computer science, it would appear very odd that this, particularly important aspect of the computer science field would be absent, however that is the case. This paper will attempt to go into the reasons for this, as well as propositions that others have put forward to remedy this issue. We finally propose an approach towards resolving the software / requirement engineering challenges by comparing four methodologies Agile, SSADM, UP and Domain Engineering, and select the best approach that can help resolve the software / requirement engineering issues while developing bioinformatics software.**

***Keywords*—Requirements engineering; bioinformatics; agile; UP; SSADM; domain engineering**

I. INTRODUCTION

The field of bioinformatics is a relatively recent; however, rapidly growing field, and one that spans both computer science and biology. It is focused on making cutting-edge scientific discoveries through sophisticated analysis of biomolecular data such as DNA, RNA, and protein sequences. It is an interdisciplinary field in which computer technology and computer science techniques, including software, hardware and algorithms, are applied to solving problems arising in biology. The primary stakeholders are biologists rather than computer scientists. As such, it presents a unique situation for the field of software engineering, as it presents challenges and opportunities that are not typically present during the normal engineering process [1]. For instance, stakeholders may be more inclined to sacrifice program structure to get something that works. While the field itself is indeed very different from the typical software engineering situation, with generally much tighter restraints on budget and timetables, as well as less time allotted for verifying and testing, the end goal of creating accurate and reliable scientific software is no less critical since incorrect results would greatly compromise the validity of the discovery. Furthermore, developing an easily maintainable and functional, as well as well-documented piece of software is still ever-present. Just as in the more general field of computer science, the practices of requirements and software engineering should be introduced in the academic lives of those involved. As this field is part of computer science, and is included in the computer science departments of many universities, requirements engineering would be assumed to be part of the curriculum that these students would enroll in; however, as Umarji, et al. [1] discuss, this is not the case at all. In their studies of the syllabi of over 50 universities and colleges across the United States, Umarji et al. found that, while the students were always well instructed in general computer science topics, such as "design and analysis of algorithms, databases and programming languages in all the bioinformatics programs, however little or no training given to these students on basic software engineering principles." [1]. Part of this may stem from the fact that "only recently have studies on end-user programming and information activities in bioinformatics started to emerge; there is still a large gap in our understanding problems in bioinformatics software development" [2]. Their particular study into this lack of application of the software engineering discipline is particularly compelling, as it includes roughly 50% of persons from the computer science discipline, and 50% from the molecular biology and biochemistry disciplines, so that it shows the influences that both sides have had on each other in this process of discovery.

A. Problem Statement

Bioinformatics, as a field is almost unique in regards to how requirements engineering is concerned. As Umarji, et al. [3] point out, within this field, unlike most others, there is a relatively large percentage of the practitioners of the field (i.e., bioinformaticians) who are doing the programming themselves, and have been left to their own devices in terms of software development and documentation. While science itself, and the experimental model teaches that documentation of every detail is important, this would appear to stop when it comes to the development of the tools that these scientists would use, and documentation is very limited, if it exists at all. This would appear to match up with their findings that, when asked about where these scientists learned what they knew about the software development process, "84% of [126] respondents indicated that they had learned through self-teaching alone, or that self-teaching was one of their main modes of learning." [1]. This statistic in itself shows a considerable void in the formal training of the scientists involved with bioinformatics

 ISBN: 978-1-61208-283-7

or computational biology, let alone the formal training in software engineering.

II. Literature Survey

The study from Umarji, et al. [1] clearly suggests a lack in implementation of software engineering and requirements engineering methodologies in the development of bioinformatics tools. It would fit in with their earlier findings in regards to the number of university programs where bioinformatics students who had taken software engineering or requirements engineering as part of their degree program, and this also would make sense given that "70% [of those same respondents] responded that they had taken some computer science courses. Ten percent of the respondents completed certification programs to gain proficiency in software development." The fact that so few of these respondents have any formal training in requirements engineering is indeed a disquieting figure: the implications are that there is a lot of software that is developed by an individual, and that once that particular individual is gone, the maintenance and efficiency of that software, even just the simple understanding of its function, will also be lost. While, unlike in a traditional business setting, this loss may not be financial in nature, the loss still could have far-reaching repercussions on the advancement or understanding of a particular experiment. In addition, the majority of these same respondents also identified themselves as bioinformatics specialists with some programming ability (57%), as opposed to programmers first (35%). Finally, most of these respondents indicated that they worked on mid-sized projects of about 5-20 KLOC, and that there was a broad range of team sizes. In short, this is clearly a discipline that covers a very broad range of experience in terms of programming ability, but which shows a very inconsistent level of education in the merged field, and one which is primarily graduate-level. This would indicate that though they completed many years of advanced computer science courses, in addition to those in biology, one of the most fundamental of these graduate courses, software engineering, requirements engineering or architecture were never covered.

To address this, Umarji, et al. propose a curriculum that fits both the demands of the field, and which emphasizes proper software engineering disciplinary techniques. Instead of adding a new class or requirement, their approach spreads the importance of disciplined software engineering in current classes, avoiding adding time to an already lengthy training process. Chilana, et al. [2] also bring up the lack of standardization since bioinformatics is such a new field. As software engineering is in a similar state when compared to other engineering disciplines, this should be a problem that software engineers themselves are familiar with.

Another problem pointed out by Chilana, et al. is that while bioinformatics is a cross-disciplinary field, it is one in which the two disciplines do not even speak remotely the same language. For example, they discuss how the two groups in their study, the computer science-oriented and the molecular biology and biochemistry-oriented, do not even approach the problems in the same manner: the computer science group "often used command-line interfaces and programming languages that they were comfortable with and did not find it challenging to locate any related technical information." [2]. Meanwhile, the more bioinformatics-oriented group, developed applications which "were simple at the beginning to match their research purposes the participants in this category had primarily self-taught programming skills and often sought information ... or obtain additional help from colleagues in implementing a solution."

Most bioinformaticians and computational biologists believe that good bioinformaticians build up their own toolbox, and are aware of and use existing tools to do more powerful work. The software development practices mostly surround the notion of "Don't reinvent the wheel" which essentially refers to the use of existing frameworks and to take advantage of large existing projects like BioPython [4], which contains a lot of ready to go code for practically everything.

This would indicate that while those in the bioinformatics background are less likely to have a software engineering or requirements engineering course to teach them the necessity of requirements specification, software maintenance and development, they are more willing to branch out than those who were educated in a far more rigid, single discipline environment. An ideal solution would, therefore, include a cross-over: better training of those in bioinformatics with regards to requirements engineering, and a cross-discipline research course in the computer science field.

III. Software and Requirements Engineering Challenges

The term bioinformatics has a range of interpretations, but the core activities of bioinformatics are widely acknowledged: storage, organization, retrieval, and analysis of biological data. A bioinformatician works to provide services to the scientic community in the form of databases and analytical tools. The serious challenges facing bioinformatics over the next decade include integration and presentation of the enormous and ever expanding size data. In order to make use of the relatively weak signals present in a single data source, it is necessary to integrate data from different views of the same system. Of course, integrated data is still just a mountain of data. Researchers need tools that present the data in a comprehensible fashion, annotated with context, estimates of accuracy and explanation. Another challenge that bioinformaticians face while developing bioinformatics software is that in most cases the tool or collection of tools was written in an ad hoc manner to be used for an experiment. If the experiment is successful, this tool evolves into a large scale project and sometimes even considered to be commercialized. In such situations, little emphasis is paid on the organization and requirement gathering process in the early stages of the software. A bioinformatician gathers information from the biologists and takes an evolutionary approach towards building the software and in most cases the

 ISBN: 978-1-61208-283-7

approach is so raw in the sense that absolutely no importance is given to the software and requirement engineering process simply because the main objective is to create a tool/software "quickly", that only a selective set of users who are involved in the research can use [5].

We put forward an approach to resolve the aforementioned challenges. We considered four different methodologies, Agile [6], SSADM (Structured Systems Analysis and Design Method) [7], UP (Unified Process) [8] and DE (Domain Engineering) [9]. Table I compares the four methodologies in terms of how they address the technical challenges of bioinformatics software engineering. After an exhaustive study of the pros and cons of each methodology for software development in the field of bioinformatics we came to the conclusion that an integrative methodology of UP, SSADM and DE would be the best candidate for software development in the field of bioinformatics. The following section explains the reasons why.

IV. Evaluation of Methodologies

A. Evaluation: Agile methodology

One of the fundamentals of agile methodology is user involvement. However, in our case, user involvement was expected to be low for the following reasons:

- Most of the projects are research oriented, the users are mainly the biologists of the research group and they are usually unavailable due to scheduling constraints and only play a small part in the knowledge transfer of the domain to the development team.
- The development team members are mostly themselves part of the research group, so they perceive (correctly or not) that they understand the system requirements and do not feel the critical need to involve the users.

Requirements although expected to change along with the changes in research objectives for this system, they are still stable for a phase due to the following reasons:

- There are not many stakeholders involved (only the research group).
- Existing work process is to be automated
- User stories are not that complex since most bioinformatics tools developed are for analytics and hence it is the algorithm that takes preference.
- Existing process/work-cycle can be demonstrated easily in short span of time thereby giving us enough insight into how the system should work and create plan. On an average, it takes bioinformaticians about 3-4 hours to obtain a small subset of results using theoretical methods/algorithms.

In practice, the lightweight nature of agile methods affords a lot of flexibility to the development process, but makes agile methods difficult to implement in a disciplined manner without coaching [10]. An undisciplined application of agile methods leads to a "patch and go" attitude. Agile methods are commonly used in the development of scientific software (e.g., [11], [12]). However, the use of agile methods by programmers with no formal training in software engineering increases the likelihood of undisciplined application of the process, leading to lack of documentation, and code that is difficult to maintain or reuse. Moreover, the common quality assurance practice employed in agile processes, test-driven development, is very difficult to apply to scientific software, where it is not always clear what the expected output should be. In addition, the fact that the stakeholders, developers and users are generally all the same person does not help in this regard, as they are less likely to seek outside help, and to view documentation as a waste of time, as they are the only persons involved. For these reasons, agile is not a good fit for creating maintainable scientific software.

B. Evaluation: Unified Process

Unified Process (UP) [8] views software development as consisting of four iterative phases:

1) *Inception*: the need for the software is justified and cost-benefit and risk analyses are conducted.
2) *Elaboration*: this is where requirements are elaborated and the software architecture is created.
3) *Construction*: the design is further detailed and the software is incrementally built.
4) *Transition*: the developed software is delivered to the client.

UP provides an extensible framework to manage software development: one of its major features is that it considers software development to be an iterative and incremental process whereby each stage has a specific area of focus, which is just the kind of approach required in a research-oriented field of bioinformatics.

The iterative and incremental process divides the project into smaller chunk called increment and each increment refines the functionality of intended system by undergoing several iterations, following the complete process of traditional waterfall method [6]. In UP, the system development starts with little knowledge about the objective and as projects continues the knowledge about the system increases, which is exactly the kind of needs a research project calls for. While agile processes have a similar iterative development cycle, a key distinction is the discipline imposed by the heavyweight process associated with UP, particularly the explicit requirements engineering activities carried out during the elaboration phase.

Unified Process handles risk well throughout the development process. With deliverables at each stage of the development there is a go/no-go decision to be made analyzing each deliverables. [13]. This helps in identifying and analyzing the risk at early stage making it possible to decide the steps to be taken in course of time.

 ISBN: 978-1-61208-283-7

TABLE I. COMPARISON OF METHODOLOGIES

Methods	Challenges		
	Explosion in Data Sources	*User Centric Design*	*Other*
Agile	Lack of documentation makes it difficult to understand how code deals with data formats. Lack of upfront design modeling makes it harder to adapt to unanticipated formats.	Requires more customer participation time than users (biologists) can commit to.	Advantage in dealing with rapid change is not applicable due to relatively stable requirements.
UP		Well-documented incremental development makes it easier for users to add more feature requests over time.	Risk management process facilitates review at every stage of development.
SSADM	Strong data modeling emphasis fits well with managing diverse data needs.	Too rigid for incremental development.	
DE	Domain analysis provides understanding of the commonalities and variabilities of the various data formats.	Reduces dependency on domain expert in later phases of project or future similar projects.	Develops infrastructure, generic architecture and common "assets". Facilitates development of taxonomy and ontologies.

C. Evaluation: SSADM

We consider SSADM [7] to be an important candidate for software development in the field of bioinformatics because of the three most important techniques used in SSADM:

- Logical data modeling: the process of identifying, modeling and documenting the data requirements of the system being designed; the data are separated into entities (data required to record information) and relationships (the associations between the entities).
- Data Flow Modeling: the process of identifying, modeling and documenting how data moves around a system; this process examines processes (activities that transform data from one form to another), data stores (the holding areas for data), external entities (what sends data into a system or receives data from a system), and data flows (routes by which data can flow).
- Entity Behavior Modeling: the process of identifying, modeling and documenting the events that affect each entity and the sequence in which these events occur.

Since dealing with huge amount of data is the major part of the problem in bioinformatics, the explicit data modeling techniques inherent in SSADM are suitable for the problems bioinformaticians deal with the most.

While the overall SSADM process is too rigid [14] for the incremental user-centered design approach that is ideal for development of bioinformatics software, there are certain aspects that are still applicable. Of the different stages involved, we select the ones that are essential for software development in the field of bioinformatics. These stages are:

1) *Feasibility study*: A feasibility study is effectively a condensed version of a fully blown systems analysis and design, to investigate the goals and implications of a research project before committing resources to it.
2) *Investigation of the current environment*: This is one of the most important stages in the software development methodology; although the new environment may be radically different, the concepts underlying bioinformatics software will remain the same and hence it is critical to investigate some form of the current system.
3) *Requirement Specification*: This is probably the most complex stage in the methodology. Using the requirements developed in stage 2 and working within the framework of the selected module, the analyst must develop a full logical specification of what the new system must do.
4) *Technical system options*: This is the first stage towards a physical implementation of the system and a large number of options for the implementation are generated such as the hardware architecture, software to use, cost of implementation, staffing required, physical limitations and constraints such as a space occupied by the system, the distribution including any network which that may require and the overall format of the human computer interface.
5) *Logical design*: This level concentrates on the requirements for the human computer interface, which is something important when it comes to research projects having potential of being widely used, such as a Genome Browser or an interface for querying a disease database. The logical design specifies the main methods of interaction in terms of menu and command structures.
6) *Physical design*: This is the final stage where all the logical specifications of the system are converted to descriptions of the system in terms of real hardware and software.

D. Evaluation: Domain Engineering

Domain engineering [9] provides a systematic process for analyzing a family of similar applications in order to produce a common extensible framework. It consists of the following activities:

1) *Domain analysis*: In this stage that is unique to domain engineering, the domain is studied to understand what the different applications have in common and how they vary. This also leads to the definition of the scope of the domain, i.e., which applications to consider part of the domain and which ones to exclude.

 ISBN: 978-1-61208-283-7

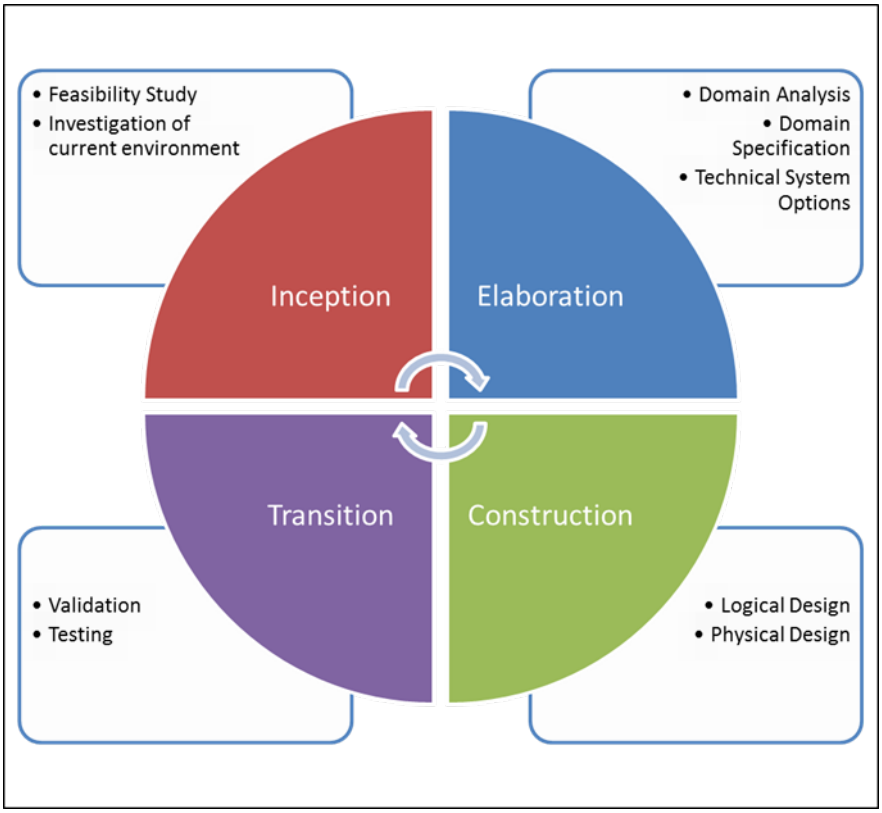

Fig. 1. An integrative approach towards a suitable Software/Requirement Engineering Methodology for Bioinformatics Projects

2) *Domain specification*: In this stage, the artifacts are refined, producing a domain-specific requirements document followed by a reference architecture that defines the common components and reusable assets as well as where application-specific components and modifications can be added.
3) *Domain implementation*: This stage uses existing software development processes to take the domain-specific requirements and architecture and create the framework and other core assets that can be used for developing additional applications.

The domain engineering process is well-suited to analyze and model the diverse data sources and organize the multiplicity of scripts that perform similar analyses on bioinformatics data. Domain analysis studies the commonalities and variabilities of the various data sources and formats, enabling developers to understand how they vary and anticipate additional variations.

The outcome of domain engineering activities facilitates further development of a product line of similar bioinformatics applications. Development of a product line tool or workbench makes it possible to easily develop add-ons to existing products. It makes every function and feature of the family of products available for use, extensions, and possible adaptation of developed/abstracted features. This can be accomplished with the creation of bioinformatics-specific designer tools that simplify many of the programming tasks for bioinformaticians.

E. Integrative Approach

We present here an integrative software and requirement engineering approach for the domain of bioinformatics where applied research is the brainchild for development of software in this unique discipline. Research can be perceived differently depending on viewpoint, academic or industrial. In this paper, we define applied research as an activity that gains intellectual leadership that ultimately leads to commercial reward for a company by combining the classical viewpoints. We measure this in terms of the combination of intellectual leadership (knowledge) together with the ability to demonstrate new ideas through proof of concept prototypes.

From our survey, we observed that since a number of the methodologies used in software development today have their origins in the waterfall process and tend to separate activities into distinct phases of design, coding, testing and integration. These activities have been found to occur repeatedly on each cycle or iteration. Iterative design processes such as the Unified Process (UP) have become widely used over the past decade. Since, research is considered to be an incremental and iterative process; we chose UP

 ISBN: 978-1-61208-283-7

to be the backbone of our methodology and incorporated different stages of SSADM and domain engineering into the four stages of UP methodology. Figure 1 shows the integration of these methodologies. The feasibility study and investigation of current environment activities are adopted from SSADM to guide the inception phase, justifying the business case. During the elaboration phase, domain analysis and specification are adopted from DE to guide requirements analysis activities that can be shared across several related bioinformatics applications. A study of the technical systems options are used to understand the requirements from the systems engineering standpoint. In the construction phase, logical and physical design activities model the structural and behavioral aspects of the software. Finally, the transition phase includes system testing and validation.

V. Conclusion and Future Work

As was stated before, the lack of any formal training in requirements engineering principles has led to a major problem in the field of bioinformatics: many programs with no requirements specifications, no maintenance plans, and which, more often than not, have only a single user who will ever understand these issues. While general education in the field of requirements engineering and architecture would certainly assist in making sure that these principles are reinforced while in academia, the uniqueness of the field in practicum presents challenges that are not seen in most other areas. To this end, several techniques that are discussed in a typical requirements engineering class are on the table. The first of these, agile, is actually very common in the field itself, however, as it is practiced almost to the point of exclusion, it has contributed to the problem almost as much as the lack of education in the requirements engineering. Instead, the more practical option would be a combination of the Unified Process, SSADM or Domain Engineering methods that were discussed, as they would ensure a more robust architecture thru requirements specification, an actual architectural design, and a maintenance plan put in place to ensure that others actually know what the developed software is for and how it is to be used, beyond simply those using it in the here and now.

Future work includes empirical validation of the effectiveness of the integrated approach. Validation of any new process or methodology is always difficult, particularly during the formative years, partially due to the subjective nature of evaluation but primarily due to the limited experience in using the technique. As every research project is different, it is difficult to exactly compare productive gains; however many approaches for examining the credibility, reusability and the efficiency of performing research have been discussed at length [14], [15].

In addition, we will investigate aspects of other software development processes that can be used to further support informally trained software developers in building reliable systems.

References

[1] M. Umarji, C. Seaman, A. G. Koru, and H. Liu, "Software engineering education for bioinformatics," in *Proceedings of the Conference on Software Engineering Education and Training (CSEET 09)*, 2009, pp. 216–223.

[2] P. K. Chilana, C. L. Palmer, and A. J. Ko, "Comparing bioinformatics software development by computer scientists and biologists: An exploratory study," in *Proceedings of the ICSE Workshop on Software Engineering for Computational Science and Engineering*, 2009, pp. 72–79.

[3] M. Umarji, M. Pohl, C. Seaman, A. G. Koru, and H. Liu, "Teaching software engineering to end-users," in *Proceedings of the 4th International Workshop on End-User Software Engineering*, 2008, pp. 40–42.

[4] P. J. Cock, et al., "Biopython: freely available Python tools for computational molecular biology and bioinformatics," *Bioinformatics*, vol. 25, no. 11, pp. 1422–1423, 2009.

[5] K. Pavelin, J. A. Cham, P. de Matos, C. Brooksbank, G. Cameron, and C. Steinbeck, "Bioinformatics meets user-centered design: A perspective," *PLOS Computational Biology*, vol. 8, no. 7, pp. 1–4, 2012.

[6] I. Somerville, *Software Engineering (9th Ed)*. Addison-Wesley, 2010.

[7] J. S. Hares, *SSADM for the Advanced Practitioner*. John Wiley & Sons, Inc., 1990.

[8] Rational Software, *The Rational Unified Process, version 5.0*, Cupertino, CA, 1998.

[9] D. M. Weiss and R. Lau, *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.

[10] M. M. Muller and W. F. Tichy, "Case study: Extreme programming in a university environment," in *Proceedings of the International Conference on Software Engineering (ICSE 01)*, 2001, pp. 537–544.

[11] O. Chirouze, D. Cleary, and G. G. Mitchell, "A software methodology for applied research: extreme researching," *Software: Practice and Experiences*, vol. 35, no. 15, pp. 1441–1454, 2005.

[12] W. A. Wood and W. L. Kleb, "Exploring XP for scientific research," *IEEE Software*, vol. 20, no. 3, pp. 30–36, 2003.

[13] B. W. Boehm, "A spiral model of software development and enhancement," *IEEE Computer*, vol. 21, no. 5, pp. 61–72, 1988.

[14] B. Kitchenham, S. L. Pfleeger, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[15] C. Robson, *Real World Research*. Backwell Publishers: Oxford, 2002.

 ISBN: 978-1-61208-283-7