Essential Design Modeling for Scientific Software Solutions Development

Zeeshan Ahmed$^{1,2,3}$
$^1$The Jackson Laboratory for Genomic Medicine, CT, USA
$^2$University of Massachusetts, School of Medicine, MA, USA
$^3$University of Wuerzburg, Department of Bioinformatics, USA

Abstract

Software design and its engineering is essential for bioinformatics software impact. We propose a new approach ‘Butterfly’, for the betterment of modeling of scientific software solutions by targeting key developmental points: intuitive, graphical user interface design, stable methodical implementation and comprehensive output presentation. The focus of research was to address following three key points: 1) differences and different challenges required to change from traditional to scientific software engineering, 2) scientific software solution development needs feedback and control loops following basic engineering principles for implementation and 3) software design with new approach which helps in developing and implementing a comprehensive scientific software solution. We validated the approach by comparing old and new bioinformatics software solutions. Moreover, we have successfully applied our approach in the design and engineering of different well applied and published Bioinformatics and Neuroinformatics tools including DroLIGHT, LS-MIDA, Isotopo, Ant-App-DB, GenomeVX and Lipid-Pro.

Introduction

Computer Science has revolutionized almost all other fields of life. Common man including engineers, doctors, artists, technicians and scientist etc., somehow, every one’s life is now partially depending on the usage of informatics. In the past (1980s), the informatics (IT) issues were related to the development of the large sized but small-scaled applications. Later on (1990s), with the passage of time systems started becoming complex but smaller in size, especially with the evolvement of the concept i.e. Component Based Systems (CBS)$^1$ and the innovations of advanced programming tools and technologies e.g. Enterprise Java Beans, Microsoft COM and CORBA etc. So far the focus of the last decade (2000s) was to develop smart, intelligent and robotic applications.

Particularly in life science, with the front-runner field bioinformatics, the world has been changed by small, efficient, fast, logical, embedded and intelligent software, databases and management systems. Even this year’s (2013) Nobel Prize winners (Arieh Warshel, Martin Karplus, Michael Levitt)$^3$ in the field of Chemistry relied on powerful computational programs to understand and predict biochemical processes and molecular dynamics, giving testimony to the novelty and innovation of bioinformatics.

Software Engineering Principles

To establish and expedite the processes of scientific software engineering (SSE), many Software Development Life Cycle (SDLC)$^4$ models have been introduced e.g. Waterfall Model, V-Model, Spiral Model, Iterative and Incremental Model, Rapid Prototype Model, Extreme Programming Model, Evolutionary Model, Agile Development Model, Code and Fix Model etc., and some other Process improvement models$^5$.

SDLC is a goal-oriented approach toward the software development. Almost all of the proposed SDLC models provide distinct processes for the software implementations. Depending upon the nature of the end product, the right model has to be chosen and applied. Based on the process’ artifacts and logical steps for developing a software project (e.g. time, quality, size, development effort etc.), it is not easily possible to compare different SDLCs$^6$, but doing so reveals differences$^7$. Own efforts did focus on quality improvement of software$^8,9,10,11,12$.

Depending upon the observed commonalities, in general, we state that the software engineering is an integrated, cyclic and product line combination of following independent modular approaches: requirements engineering$^3,14,15$, design modelling$^{16,17,18,19,21,22,23,24,25,26,27,28}$, programming, testing and deployment.
These five modular approaches (Figure 1) follow the procedures of some life cycle management approaches, which can help them in performing their individual functionalities as well as regulating tasks in cyclic chain processes.

**Figure 1.** Traditional Software Development; consisting of integrated and cyclic combination of the following independent modular approaches: requirements engineering, design modeling, programming, testing and deployment 65,66.

**Scientific Software Engineering**

Testing of integrated and individual modules becomes time consuming (Figure 2), as new test cases have to be rewritten all the times or the application exists with a high expectation of ripple effects 29 (i.e. unidentified logical or syntax errors in the system which arise while fixing the identified logical or syntax errors). The quality of a software application decreases with an increase in the ripples a change in software creates. Moreover measured optimum software maintenance can only be achieved with the accessibility of the concrete information about the ripples effect in the system [30]. Depending upon the nature of the system, many approaches have been proposed to improve the software quality measurement processes (e.g. 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, etc.), towards the traditional software development but one can also use these in the scientific software solution’s quality assurance and for improvements as well.
Figure 2. Scientific Software Engineering (SSE)\textsuperscript{65--66}. SSE integrates and combines in a development cycle the following independent main modular approaches: requirements engineering, design modeling, programming, testing and deployment. Each approach consists of its own sub-modular, integrated and cyclic combination of internal phases: requirement engineering consists of specification, functional, non-functional, and analysis; design modeling consists of use cases, system flows, data flow and source code; programming consists of languages, tools and technologies, development, and debugging; testing consists of test cases, modular, integrated and quality; finally, deployment consists of installation, configuration, training, feedback. Iterative cycles lead to continuous improvements, achievements translate the goals into good software.

Database manipulation and management system: If the target scientific software solution has this focus, then it requires to properly modeling the database schema (entity relationship model) by reducing the levels of data redundancy and dependency, using data normalization. There are five data normalization forms: 1NF, 2NF, 3NF, 4NF and 5NF, conceptual procedures for comprehensive database designing\textsuperscript{44}. These data normalization forms help in shaping the data types (1NF), developing relationships between non-key and key fields (2NF, 3NF)\textsuperscript{45, 46}, and deals with multi-valued facts corresponding to the many to many relationships (4NF and 5NF)\textsuperscript{47, 48}.

Human Computer Interaction and Scientific Applications: The Human Computer Interaction (HCI), also known as Human Machine Interaction (HMI)\textsuperscript{49--51}, has to correlate with the Scientific Application development. HCI defines the implementation of the mechanisms to establish the efficient communication protocols between human
and machines. These protocols are based on the textual, visual, sensory, audio and event based information, provided by both the user and the machine (computer).

**Butterfly Workflow Design and Software Examples**

To implement the Butterfly model \(^{65, 66}\), we have designed a three-layered architecture (Figure 3), going from abstract planning (gray) to designers and developers (yellow) to implementation and user (green).

**Figure 3.** Butterfly three-layer model \(^{65, 66}\). Shown in grey is the abstract layer, the basis for design and development (yellow), followed by implementation and testing by the user (green) so that the software is released including installation and training.

*Abstract planning:* Scientific software solution planning is the first step towards a new scientific application development. It requires good knowledge of the field (e.g. biochemistry, neurobiology, genetics, metabolomics, proteomics etc.) as well as project related information (e.g. what could be the end product, in-put to the system, expected output from the system, methodology, ideas, user opinions etc.). The next important phase is to perform requirements engineering and analysis. The third phase is the conceptual software design and modeling. Before moving ahead, first go for some abstract designs based on functional requirements and discuss these in your team. The last phase is software solution planning. It concerns the design of a user-friendly graphical interface.

*Software design:* This layer involves the designers and developers. It consists of four steps: design and modeling and analysis, tools and technology selection, design implementation and graphical user interface implementation.

*Implementation:* The last layer concerns implementation and programming and involves in house testing and debugging (by the developers and tester). Steps include scientific software solution testing, debugging and creation of new versions, users involved in testing and feedback and finally installation and training.

The Butterfly workflow design accentuates experience from previous software developments including a number of larger efforts (Table 1). Most of these are team efforts that simply have come close to the Butterfly paradigm, but by chance and pressure, not by explicitly following a scientific approach. With rapid development of new software applications, the need to formalize the software solution development principles increases to ensure that all scientific applications follow the standard scientific paradigms.

Adopting the concepts of Butterfly model, some new scientific software applications have already been proposed, designed, implemented, tested and are in use (LS-MIDA \(^{52, 53}\) (Figure 4), DroLIGHT \(^{54, 55, 56}\) (Figure 5), Isotopo \(^{57}\) (Figure 6), Lipid-Pro \(^{60}\) (Figure 7), App Ant Database \(^{67}\) (Figure 8) and GenomeVX \(^{68}\) (Figure 9).
Figure 4. Graphical user interface of LS-MIDA

Figure 5. Graphical user interface of DroLIGHT
Figure 6. Graphical user interface of Isotopo

Figure 7. Graphical user interface of Lipid-Pro
Figure 8. Graphical User interface of Ant-App-DB

Figure 9. Graphical User interface of GenomeVX
To determine in a more objective way potential gains from scientific software design following our proposed butterfly paradigm we have performed a short comparative analysis of some bioinformatics software applications (C13, Metatool, BioOpt, FiatFlux, ReMatch, Biolayout, LS-MIDA, DroLight, Isotopo), describing their type, methodology, implementation, user friendliness, configuration etc., based on the provided, published information (summarized in Table 2).

Table 1. Comparative analysis of different Bioinformatics software applications.

<table>
<thead>
<tr>
<th>Butterfly area Software</th>
<th>Engineering Approach</th>
<th>Scientific Methodology</th>
<th>Scientific Application</th>
<th>Human Computer Interaction</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPASI</td>
<td>Traditional</td>
<td>C++</td>
<td>General software design</td>
<td>Copasi GUI Command Line</td>
<td>Hoops S., Sahle S., Gauges R., Lee C., Phil J, Simus N., Singhal M., Xu L., Mendes P. and Kummer U.</td>
</tr>
</tbody>
</table>
### Table 2. Comparative analysis of different scientific software applications.

<table>
<thead>
<tr>
<th>Application / Comparative Measures</th>
<th>C13</th>
<th>Metatool</th>
<th>BioOp t</th>
<th>Fiatlu x</th>
<th>ReMatc h</th>
<th>Biolayou t</th>
<th>LS-MIDA</th>
<th>Dro-LIGHT</th>
<th>Isotopo</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>App. Type</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Web</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
</tr>
<tr>
<td>Script or Prototype</td>
<td>Script</td>
<td>Script</td>
<td>Prototype</td>
<td>Script</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td></td>
</tr>
<tr>
<td>Algorithm Type</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Sequential</td>
<td></td>
</tr>
<tr>
<td>Algorithm / Methodology</td>
<td>Isotopic Labeling</td>
<td>Schuster Algorithm</td>
<td>Mass Balance Equation</td>
<td>Isotopic Labelling</td>
<td>Carbon Mapping</td>
<td>Markov Clustering</td>
<td>Least Square</td>
<td>Circadian Rhythm</td>
<td>Partial Least Square</td>
</tr>
<tr>
<td>Running Mode</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Batch</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
<td></td>
</tr>
<tr>
<td>Publishing, licensing</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C13</th>
<th>Metatool</th>
<th>BioOp t</th>
<th>Fiatlu x</th>
<th>ReMatc h</th>
<th>Biolayou t</th>
<th>LS-MIDA</th>
<th>Dro-LIGHT</th>
<th>Isotopo</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>App. Type</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Web</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
<td>Desktop</td>
</tr>
<tr>
<td>Script or Prototype</td>
<td>Script</td>
<td>Script</td>
<td>Prototype</td>
<td>Script</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
<td>Prototype</td>
</tr>
<tr>
<td>Algorithm Type</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Parallel</td>
<td>Sequential</td>
<td>Sequential</td>
</tr>
<tr>
<td>Algorithm / Methodology</td>
<td>Isotopic Labeling</td>
<td>Schuster Algorithm</td>
<td>Mass Balance Equation</td>
<td>Isotopic Labelling</td>
<td>Carbon Mapping</td>
<td>Markov Clustering</td>
<td>Least Square</td>
<td>Circadian Rhythm</td>
</tr>
<tr>
<td>Running Mode</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Batch</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
<td>Interactive</td>
</tr>
<tr>
<td>Publishing, licensing</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
<td>Published, Free</td>
</tr>
</tbody>
</table>
Conclusions

Conscious adaptation of SSE principles as exemplified here by the suggested butterfly design and its multilayered architecture, might look like an increase in developmental workload in comparison to many current bioinformatics programming methods. However, on the long run, it will reduce the burden by making the scientific application well designed, flexible, structured and reusable. It allows a product line development, is analytical and allows qualitative software improvement. Furthermore, HCI concepts make it user friendly, easy to learn and to deploy.

Conflict of Interests

The authors declare no conflict of interest.

References
