SOP Propsal for Software Maintainence and Development Strategies

Joshua Cragun *
Kris Koford, WRTG 3014

October 3, 2018

*u1025691

1 Introduction

It is well understood that since the creation of the first computer in 1936, the fields of computer science and engineering have been consistently revolutionizing themselves, and the industry of software development has grown exponentially. Indeed, computer systems and software Application Programming Interfaces (APIs) for commercial use have become among the most complex technologies mankind produces and must maintain– particularly since any organization can only dedicate a relatively small proportion of individuals involved with these systems to maintain them in order to prolong their efficacy. Further, the complexity of software management and development can increase as the amount of personnel working on the project does as well. This means that, due to the complex and open nature of software development, there is no single procedure for designing all and any program. However, it does indicate an urgency for creating an open, dynamic procedure for planning what software practices will be utilized in order to develop code smoothly and efficiently on large-scale projects where many people may be developing the same components of a program simultaneously.

2 Background and Rationale

In addition to being complex, commerical software can be quite costly in net value, development time, and also when runtime problems occur. The demands upon modern software include a need to ensure that both new features and bug fixes are brought to the consumer level as quickly and efficiently as possible. Hence, it is of great concern to ensure that expenses are cut wherever possible. These expenses are largely determined by the practices the developers employ when building the program.

Traditionally computer science students are first taught how to improve efficiency on the algorithmic level. Although this is important knowledge to have, recent developments in computer technologies have shifted the importance of source code optimization in the professional software development setting. The reason the importance has shifted is due to the compilation process for most if not all mainstream languages. Modern compilers engage in low level and high level assembly code optimizations, causing runtimes to differ very little between a program that was made quickly without concern for source code optimization and one that was meticulously crafted in or-

der to satisfy efficiency considerations, for example. This is because when the compiler translates the source code into assembly code, it will change both programs to run as efficiently as possible, performing the same modifications to the "sloppy" program as was implemented in the other program, and possibly even more. Although it might have been possible to argue that people are better at optimizing code than machines ten years ago, this is no longer the case and over time language compilers are only getting more sophistocated and efficient. Therefore, spending time rewriting source code to be more efficient is often a waste of time and can, in the worst case scenario make the code harder for other developers to understand if they do not uphold excellent software practices. As a result, when software projects are discussed on a professional level, it is usually expected of developer to know which algorithms and data structures perform best for the situation, and instead the greatest area of concern is generally focused on how these algorithms and data structures should be implemented on a more abstract, organizational level.

The reason an SOP becomes necessary is from considerations about the unique position the software industry is in presently, in combination with the advent of the world wide web. Proficiency with many scripting and programming languages can easily be attained without attending any sort of post-secondary education. In fact, many people of all ages and backgrounds can easily begin learning the basics of software development on their own, and many do. When combined with the fact that the labor supply in data science and software development has not been growing at the same rate as the industry and their openings, it is entirely possible –and even reasonable– for businesses to be willing to employ individuals into developer positions even if they do not have a degree in computer science. Further, even among those with post-secondary education, different institutions will have different standards expected from their students. As a result when people of different levels of coding experience and standards gather to work on a problem, progress can be hindered since now developers will not only need to work on their own additions to the code, but also need to read and modify the code of others as well.

Although these difficulties can be managable, a simple way to reduce the deceptively expensive costs associated with these difficulties would be for the company to create a standard operating procedure that details and educates about what software practices must be upheld while writing the organization's software. Doing so would standardize code organization and decrease

the amount of confusion and bugs in the development of the project and allow developers to easily work other distant sections of the same project with ease. The objective of this standard operating procedure would therefore be to provide a template to companies so that they can implement their own design patterns for their employees and cut costs while also educating individuals learning to code so that they can not just a language but improve as a programmer as a whole, improving their adaptability and code quality substantially– therefore also improving their hireability.

3 Methodology

The SOP will be divided into at least three different sections based on the type of software practice: design patterns, debugging and test conventions, and overall code formatting. Each section will detail the varieties of methods, including their strengths and weaknesses, and how to best decide what software practices will work best for the reader's objectives. As research continues, the categories may be revised with new ones being added and others being removed if there isn't enough evidence to show that they make a significant difference. The evidence will be consistently evavluated and drawn from experiences software developers via digital correspondance and faculty in the University of Utah's School of Computing. The SOP will also be informed by whatever reputable literature there is on the topic.

To evaluate the impact the SOP has once it has been completed, volunteering companies could implement the SOP and over a six or more trial period record the number of bugs and time spent on each project for a time period of at least six months. Once the trial period is over, the source code would be assessed to see how much of it actually changed and then see if the number of bugs and time spent on projects decreased in any notable manner whenever the SOP significantly altered the way the developers were originally developing.

4 Expected Results and Application

By providing a standard means of deciding which software practices should be used to complete large scale projects, both companies and inexperiences programmers can improve their efficiency once the developers have fully adapted

to the advised changes. Specifically, it will reduce the number of bugs in the code and will allow programmers to complete projects faster by spending less time planning how to integrate new features into the software and trying to understand what their colleagues have written. For businesses this will correspond to a decrease in expenses, providing a net benefit to all parties involved thanks to a procedure that is easy to implement into the workplace.