Chapter 7

Software Engineering

Software engineering is the practice of applying engineering principles to the entire project cycle of software applications. This is best accomplished by developing and utilizing standards to accomplish product development. Today, the term software engineer is still not properly defined, but is generally applied to computer programmers or those with that title and more formally or traditionally educated, as well as people in management positions of software projects.

Other engineering fields (e.g. civil engineering, electrical engineer, mechanical engineer, etc.) are far more established and the fundamentals of which rarely change, as compared to software engineering. Coding languages are frequently, every decade, coming and going; ever changing to the demands of the day's hardware systems and customer needs.

In the early days, when computers were first making their way into business environments and later into peoples' households, software was developed and at times was not able to operate or transfer to other systems like it can today. The difference is that there were not many standards back then that were applied to the field of computing to ensure application to application compatibility and cross-platform interoperability. To address issues such as these, several committees, groups, societies, and organizations were created to develop widely recognized standards.

An organization of note is IEEE (Institute of Electrical and Electronics Engineers). It is the world's largest professional organization related to advancing technology. To create standards for computing technology, the IEEE consists of and consults with professionals in fields that may be applicable to whatever piece of technology is being scrutinized. By acquiring input by many individuals from different backgrounds, standards are developed to

best fit the market. By having standards for hardware and software components, innovation and advancement of technology is accelerated because everyone starts with the same base components and is able to fit their creations to similar structure, which are used to create a more developed base for future endeavors. Without working from the same or similar base materials, individuals and businesses would go their own route resulting in many issues when attempting to combine separate components or complete inability to interact with others using completely different applications.

Software Development and Product Life Cycle

- Planning, Defining and Analysis
- Designing, Building, and Implementation
- Testing
- Staging and Deployment
- Monitoring

Planning and Analysis

The first step of the software development cycle is the planning and analysis phase. During this phase, requirements are gathered from the client(customer). If there is no specific client, and a software package is being built as a general purpose application to be used by anyone – known as off-the-shelf software – requirements may be thought up by the development team or by conducting one or more studies of possible users where the product may be marketed.

After needs for the project are identified, a process is started to identify and prioritize requirements. Requirements are scrutinized against several factors: cost, time, size, urgency(is it a want or a need) are but a few. Once the requirements are agreed to between those developing the software and those who are purchasing the product, if there is a client specifically, the proposal is finalized and becomes the fundamental outline of what the software is to be. Throughout the project's life cycle, however, changes may be made and often are. For the project team to create the best product that may be deliverable at the agreed upon price estimation and date, frequent communication must be made between the client and development team and any changes need to be agreed to and addressed as soon as possible.

Designing, Building, and Implementation

This stage consists of designing, building, and the implementation of software and is continually evolving within the project. Software design is the process of creating the aesthetics of the finished product. At this stage, the software development team will collaborate with the client to help determine where components of the application will reside and possibly how they are accessed. While outlining what is included in a piece of software, developers must be sure to try to create an ergonomic feel to using the product. The best way to do this is by talking and working with the client before and during development, and during the testing phase. Engineers will assess how their software program will fit into their client's computer ecosystem and how user will interact with it.

When it comes to building a piece of software, details of the client's ecosystem will be gathered to determine how the software will be built. Some factors that go into this are the type of computer

systems the software will be deployed on: desktops, servers, hand-held devices, which operating systems are running on those computers, and which languages would be best suited for the particular environment, and possibly the consideration of future updates to the software. Besides which type of coding language to use, developers must consider the security aspects of their project and the hostility of the environment the software is being deployed to. Knowing what the environment is going to be like before and during the development process, will allow developers to implement security features before encountering major compatibility issues when combining security into the core of the program. If utilizing any coding components that have been created already and being repurposed, it is important to understand any security risks that come with them.

Implementing the components of the program into the design is what ties everything together to create a finished product. During development, coders will build the application's components, usually separately and concurrently, and implement them one at a time. As each component is implemented into the main branch of the program, errors may be discovered and fixed to ensure proper integration.

Testing

After completion of a product, extraneous testing is conducted to ensure everything is working and that the product is acceptable for the client or consumers. Testing is done at most every stage during the product life cycle, but more extensively during this phase; pre-shipment. To be sure a software product will work as desired, it is ran on multiple differing environments, if needed, and under different conditions. Several tens or hundreds of hours will be put into executing, running, and operating software by many people on the development team to identify bugs for quality control.

Even though a development team will test software for many hours, some programs may be released as alpha and beta releases prior to a general release. During an alpha release, software is usually tested two different ways: white-box testing and black-box testing. When a program is undergoing white-box testing, that means the developers are focused on each component at the code level – the software's source code. They are testing each chunk of code by running it, inputting any data or instructions that may be required, and analyzing the output and comparing against what is expected. While reviewing the code for errors, programmers will also use this time to identify bottlenecks or slower operating code and update it run more quickly and efficiently. Once completed, the product advances to black-box testing. Here, the development team takes it a step up and tests the software how any user is expected to use it. While black-box testing a product, the test group may consist of the main development team, colleagues, or a small test group similar to how beta testing occurs. Test cases(scenarios) are created of how users are expected to use the product and are functional in practice.

When software is released for beta testing, it is near completion and functional. Beta testing will either be released openly or closed; meaning open to the public or closed to a select group of people. By releasing software in a beta stage, the software will be ran on many more different computer systems and possibly tested in ways not thought of by the development team. Having many more people test the product will increase chances of bugs to surface and the development team will have an opportunity to acquire feedback of how the software functions for its users. Becoming a beta tester will permit early access to software and may come with financial benefits or other perquisite sometime in the future. Perhaps a great example is with video game software, beta testers will acquire knowledge and skills of the game and be a step ahead of newcomers at launch; in some cases, beta testers are rewarded with special items or accolades when playing the final release version.

Staging and Deployment

Once thorough testing of a product has been completed to satisfaction, it is released to a specific client or to an open market: retail stores and/or online. During the prior phases, the software development team learns about which types of systems that their product will deploy to. When ready, the product will be package in a way that makes it deliverable, distributable and installable for end users. This will include packing the software into a standalone executable or into a compressed folder, storing the package on a media (cd, flash drive, HDD, web server, etc.), and making it accessible to others.

Depending on requirements set forth during the planning phase, the project team may or may not have to deploy the software

on a client's system(s). Most larger businesses and organizations have in-house I.T. teams to handle installation and deployment while many smaller business and organizations will self-install or hire someone to install the software, which may be the project team.

Monitoring

The monitoring phase of the software development cycle includes user support and future updates. Not all software comes with further support and is extraneous to the other phases. During monitoring, the development team will start back at the planning phase and work their way through the cycle again as necessary while working on any updates to the software. A list may be created during the initial development process containing what additional functionality may be included in future updates, possible expansions into additional computer systems and operating systems, where further review of code efficiency may be focused on, and yet to be solved bugs/errors. While software is on the market, feedback may be collected on how to make the software 'better', additional bugs may be discovered and reported to the development team, and more accessibility may be provided for consumers to access it to create a larger user base.

More often found with open source software, since the source code is open to improvement and scrutinization by the public, are 'nightly' builds, because code may be updated and improved upon at any time and by any person outside of the official development team. Nightly builds are unofficial releases that contain recent fixes to code to add features, functionality, or bug fixes and are considered less stable than official releases. Though nightly builds of software may be highly beneficial to some users, it undergoes less testing (in general) when compared against official 'stable' releases. Since

updates are applied regularly to nightly builds, code may break (become unusable) and bug/crash occurrences may be amplified. After several subsequent nightly builds have satisfied the development team enough to warrant an official release, the software will undergo a review, repackaging, and testing process again and be released as a 'stable' build.

Similar, but slightly different to nightly builds, are regular and less frequent updates or patches to software. Software engineers will regularly monitor and test software and apply or 'push' updates to software locally and/or remotely.

Documentation

Documentation of software is a very important component of a software product that is quite often neglected. Major software releases by large corporations commonly provide decent documentation; however, not all software is released by a corporation that has the funds and employees to create thorough documentation. There are several different types of documentation that ought to be created during project development and provided for use by developers, clients/users, and anyone else who may need to work with the software. Early on during project development are two types of documentation: Requirements and Design documentations. As software is developing and completed, three more types of documentation ought to be created and accessible: Technical, User, and Marketing documentation.

Requirements documentation is created during the planning and analysis phase. This documentation will gather requirements

from the client or created for the expected user base who may purchase the software as off-the-shelf software. This documentation will be a go-to guide for the development team to determine what will be included in the project they are working on. During the project life cycle this document is subject to change and is rather common; regular communication with clients is essential to ensuring the product is developed according to wants and needs. Design documentation will piggyback off the requirements gathering process to assist developers in designing how the software will look and operate. Excellent descriptors in design documentation will explain why components will be included in the product. While tackling what is included in software, what may also be included are one or more examples of how *not* to design a component or feature. Product development may be expedited when software engineers are focused on how to initially design a component in a specific and consistent way, rather than spending time trying to build it and others using varying methodologies and approaches to problem solving.

Technical documentation includes comments littered throughout the source code and compiled documents. While creating code components of software, software engineers can insert line comments to explain what the following code does or how it works. Commenting in code is notorious for being neglected but is considered good code etiquette and is very helpful to other programmers that will likely encounter the code in the future. By creating comments explaining how code works, other developers will spend less time trying to understand how it works when it may need to be updated or fixed. Sometimes code is poorly written, and comments are negligible or nonexistent causing entire components needing to be redone, because software engineers cannot understand it. When being thorough while creating code, many other people that rely on being able to read it will have an easier time

attempting to fix it; this helps promote the concept of "pay it forward". Other technical documents that are provided with software pertain to how software is to be serviced, maintained, and how it works. Technical documents are created for other software engineers, IT professionals, and users of the software.

User documentation may include some of what is found within technical documentations but is aimed at supporting users of the software. Most commonly, user documentation consists of how to use software and is fundamentally a user manual. Most likely, user documentation will be located in user 'help' sections within software, online under help sections and as web-based articles, and within printed material shipped with a product. As useful as user documentation may be, many product developers and their benefactors seek to include user documentation as a means to market their software and increase a software's userbase.

Marketing documentation is not as complete as other types of documentation due to its differing nature. This documentation is short and used as a method to advertise to and educate potential clients of what the software can do for them.

Software Licensing

As with any product in most every country, software may be protected under various types copyright and patent laws. The significance of copyrighting a product is to guarantee, to an extent, that software will be protected against unauthorized use and reproduction. In addition to filing claims for copyrights and patents, users may be presented with a EULA(End-User License Agreement) when purchasing or installing software containing a listing of the owner's rights and exemption from liabilities due to malicious use by the user. By doing so, creators of software are able to maximize revenue and prevent others from negatively impacting sales of their product(s) due to statutorily defined theft or misuse. These licensing agreements are often quite lengthy and filled with technical and legal jargon making them highly controversial when needing be agreed to by clients prior to use of the software.

Some software is distributed freely by individuals and companies yet is still protected under licensing laws. Comparable to EULAs, Creative Commons licenses enable creators to protect their products and themselves from liability from unwanted and illicit uses of their software. Some CC licenses permit anyone to modify, repackage, and share software but with conditions: attribution, code must remain open-source, and keeping the software non-commercial or not-for-profit are but a few possible clauses that may be included with a licensing agreement.

More information about Creative Commons licenses may be found at:

https://creativecommons.org/licenses/