COMP33812:
Software Evolution
Coursework 1

Alex-Radu Malan
9770386

Laws of Software Engineering

Introduction

In the world of computer science, successful products are defined by quality and the ability of a certain team to orchestrate and plan things so that the final user will benefit from it and the organisation as well in terms of profit. An efficient and reliable software product is the outcome of the software engineering.[10] The software part of the notion refers to a collection of executable code created by programmers, but also by the libraries and the documentation related to it. Engineering is the road that a team has to go through, a representation of all the planning, requirements, design, updates that affect the software through the time. After a software product is launched, in order to maintain on the market and return a profit to the organisation, it has to grow, to be developed and gain new functionality for the end user, otherwise, the competition will make it implode. Software evolution is the notion used in the process of developing software through software engineering methods and techniques. The evolution of a software product is increasing in difficulty based on the age of that software. The big problem in the past was the cost of the maintenance and also the cost of the continuing development of a software which leads us to Manny Lehman; a computer scientist that developed a set of laws between 1974 and 1996 which focus on the balance between the new developments driving forces and the forces that are slowing down the overall progress[3].

Lehman's Laws

A software development environment can be defined as being the total resources needed by a team in order to build and deploy software-intensive systems.

First of all, there was and there also is in present a problem amongst the software engineer teams: estimation. One of the hardest parts in creating a software product is estimation since every task, every test, everything that should be part of a plan with a product as the goal will have to be estimated in a matter of time, cost, etc[5]. The software development environment was having this problem when Lehman was in the process of creating these laws. Beside estimation there were other problems such as the structure of a software, how should a team be divided into member labels, and how the engineering and the software challenges can be solved. In the 1960-1970 people would lack software engineering practices or even software engineering which lead to so-called "Software Crisis"[9]. In 1968 the "Software Engineering" term and idea was created in order to find a solution to the crisis. There was a conference which started to create and apply practices for software management and production. Next decades after this conference the software started to be modular so that huge projects would be broken into small pieces/chunks. In the 1980's the idea of "objects" was implemented which helped developers to create GUI and interacting objects, also known as "Object-Oriented Programming". Although after these practices, there was another issue that persisted and leads to wasteful software. Developers had to create programs that would require a small amount of memory and limited processing power which took all these inventions backwards since the quality of the software was weak.

When Lehman started to think about creating a template/rules for the software evolution he firstly divided the software into three categories: S-type, P-type, E-type[4]. The S-type also refers to the static-type which is defined by the specifications and solutions of a software product that are well defined and understood but also unchangeable. The P-type refers to the practical systems which have specifications well defined but the solution is not well known. The last category of software is the e-type also known as an embedded type which represents the real-world example of software that has huge potential in a matter of changes and evolution with influences from real-world variables like taxes, laws, etc. All the laws that Lehman created are also applied to the last category, the embedded type.

The reason for this category stick is because the most important part in everything is the real-world part, the practice part, since if we would stick to the first and second categories that would mean that we were sticking to theory. In theory, you can have big upfront requirements but in practice, there is always something that occurs during the process.

Software Development Environment

In the present software development environment there is a big difference that occurs compared to the past. In 1989 when the Internet was born, people started to gain access to documents and information way easier than before. The paradigm that was trending back then was the idea of selling only the binary to the customer and the source code to remain hidden since revealing the source code or any other documents related to the software product would represent a competitive disadvantage[7]. This paradigm leads us to the open source software that was a reaction for the tired programmers that had to license software and only get executables instead of developer-friendly source code. This movement was a reason for the huge increase in software engineering productivity. All the tools, frameworks and languages developed were made open source by their creators so that those who use and develop them do not need to pay for it. The most important part of this is the fact that people are keen to learn and create so that is why there are so many large-scale systems and software that is being developed by multiple individuals who do not even work in the same place or do not know each other at all[8].

Present Applied Rules

In the present software development environment, the Lehman's Laws have been incorporated into a new idea or concept called 'Agile'. The idea is to create and continue developing the software without having a big plan from the start and also without knowing the solutions, basically excluding the first two categories of software started by Lehman(s-type and p-type). The first law of continuing change is referred in the Agile Manifesto as using small iterations and user stories for continued implementation of functionalities to a software product. As we use the first law in practice, it completes the second law which states that it increases in complexity. It is vital to break a software into small chunks instead of having a huge task. The third, fourth and fifth laws are applied by just having iterations and agile practices which would identify issues and address them to the whole team and the customer as well. The sixth law which refers to the feedback is one of the Agile's most important values. The last two laws are also implemented by the Agile principles combined with the values and practices which can lead to a better software evolution.

Conclusion

In conclusion, the software environment has changed a lot over the years with closed to open source systems, practices and concepts that were created, software optimisation and team structure. Lehman's laws played an important part in the software development and its practices development by giving an idea and a point of reference about how a software can evolve with the end goal of achieving success.

Bibliography

1. Architecting.co.uk. (2018). *Development Environment Definition*. [online] Available at: http://www.architecting.co.uk/papers/Development%20Environment%20Definition%202.2.pdf

2. Cc.uah.es. (2018). *The evolution of the laws of software evolution. A discussion based on a systematic literature review.* [online] Available at: http://www.cc.uah.es/drg/jif/2013HerraizRRG_CSUR.pdf

3. En.wikipedia.org. (2018). *Lehman's laws of software evolution*. [online] Available at: https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

4. Karchworld Identity. (2018). *Lehman's Laws of Software Evolution and the Staged-Model*. [online] Available at: https://blogs.msdn.microsoft.com/karchworld_identity/2011/04/01/lehmans-laws-of-software-evolution-and-the-staged-model/_

5. Lehman, Meir M. (1980). "Programs, Life Cycles, and Laws of Software Evolution". *Proc.* IEEE 68 (9): 1060–1076. [http://dx.doi.org/10.1109/PROC.1980.11805]

6. Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle". *Journal of Systems and* Software 1: 213–221. [http://dx.doi.org/10.1016/0164-1212(79)90022-0]

7. Paulson, J.W., et al (2004). "An empirical study of open-source and closed-source software products". IEEE Trans. Software Engineering 30(4): 246-256. [http://dx.doi.org/10.1109/TSE.2004.1274044]

8. Scacchi, W., et.al. (2006). "Understanding free/open source software development processes". Software Processes: Improvement and Practice 11(2): 95-105. [http://dx.doi.org/10.1002/spip.255]

9. School (2018). *A Brief History of Software Engineering | Viking Code School*. [online] Vikingcodeschool.com. Available at: http://www.vikingcodeschool.com/software-engineering-basics/a-brief-history-of-software-engineering

10. TutorialPoint (2018). *Software Engineering Overview*. [online] Available at: https://www.tutorialspoint.com/software_engineering/software_engineering_overview.html