LEHMAN'S LAWS

OR THE LAWS OF SOFTWARE EVOLUTION

Carolina Barrios González - UO275672
Luis Miguel Alonso Ferreiro - UO270139
Jesús Alonso García - UO271723

Carolina Barrios González - UO275672
Luis Miguel Alonso Ferreiro - UO270139
Jesús Alonso García - UO271723

CONTENTS

Carolina Barrios González - UO275672
Luis Miguel Alonso Ferreiro - UO270139
Jesús Alonso García - UO271723

INTRODUCTION

Lehman's laws, also known as the laws of software evolution, are a set of rules formulated by Manny Lehman, a German computing scientist, and László Bélády in the seventies. These rules were created in order to comprehend the changes that software experiences along its existence.

In 1969, Lehman started an empirical study while working on IBM with the purpose of improving the programming effectiveness of the company. Even though it did not have much impact within IBM, this study sparked a new field of research: software evolution.

The first time the laws saw the public light was at a lecture at the Imperial College of London in 1974, where Lehman proposed the first 3 rules. From that point on, the laws were revisited, edited, and updated until 1996, where the last two were added, creating the full set of 8 laws we know nowadays.

To understand these rules, we should first take a look at what kind of systems they apply to. Lehman divided software systems in three main categories:

- *S-programs*, which are written according to an exact specification of what that program can do.
- *P-programs*, which are written to implement certain procedures that completely determine what the program can do.
- *E-programs*, which are written to perform some real-world activity.

Lehman's laws only apply to this last type. This is because E-programs are heavily linked to the environment and model, which change constantly and thus create a need for adapting software.

THE LAWS

1: CONTINUING CHANGE

A program used in a real-world environment must continuously re-adapt to the environmental changes, otherwise it would become deprecated. When the modified system is re-introduced, it promotes environmental changes, so maintenance is unavoidable.

2: INCREASING COMPLEXITY

As an evolving program changes its structure, it becomes more complex, so it is necessary to spend resources to preserve and simplify the structure. This maintenance does not add new functionality, so those are additional costs to the changes due to the environment.

Carolina Barrios González - UO275672
Luis Miguel Alonso Ferreiro - UO270139
Jesús Alonso García - UO271723

3: LARGE PROGRAM EVOLUTION

Program evolution is a self-regulated process, established at an early stage in the development process, so system attributes remain invariant for each system release. This means that it is necessary to make small changes, so it reduces the extend of structural degradation.

4: ORGANIZATIONAL ESTABILITY

The rate of development is constant and independent from development resources. This is because a change of resources has imperceptible effects on the long-term evolution of the system. Which means that large development teams are often unproductive.

5: CONSERVATION OF FAMILIARITY

During the life cycle of a system the changes that happen every release tend to stay constant or decrease. This means that the teams should bring new features and content in a controlled and fixed manner, or they will risk losing the understanding of the system. This is because every new release is bound to come with some faults and limiting the number of changes allowed is a way to keep track of those errors.

6: CONTINUING GROWTH

New content and functionality should be brought forth with every new release, to keep the users satisfied with the application. This closely ties in with the first law most of the times a new release means creating new code.

7: DECLINING QUALITY

Software will decline in quality unless its design is maintained and adapted to new constraints and functionalities. This implies that the changes required to satisfy this new changes and features make the system more complex and bring the quality down, this only means that more effort is required to satisfy those requirements while not giving up quality of software in exchange.

8: FEEDBACK SYSTEM

The development and maintenance process of software is a multi-loop with multiple agents and more than one level of feedback, that means that as a system ages it tends to be harder to change due to its complexity. This law recognized the user's input to provide new life for future releases.

Carolina Barrios González - UO275672
Luis Miguel Alonso Ferreiro - UO270139
Jesús Alonso García - UO271723

CONCLUSION

Lehman's laws were a huge breakthrough back in the seventies and, as we said, they opened the door to a new whole branch of studies. But looking at them after almost 30 years without changes, we really think its time for an update.

Software systems and the way they are being developed nowadays have changed almost completely since the 90s, and while most of the aforementioned rules still have a great impact on how software should be made today, we feel like they should be revisited and edited more often so they stay relevant in the industry.

On the other hand, we also believe that these should not be treated as laws, but more as best practices that teams should follow in order to achieve a higher level of code quality. Many projects have been successful without taking these rules into account, but we think that the progress would have been smoother if they did.