Cairo University
Faculty of Computers and Artificial Intelligence
Software Engineering Program
Software Evolution and Maintenance

Assignment 02:
Software Maintenance and Technical Debt

Khaled Ibrahim Shawki
k.shawki@stud.fci-cu.edu.eg

1.1 What is technical dept from a maintenance perspective?

Maintenance is not just about fixing bugs and implementing regulation changes, although that is where most of the effort is spent.
Under the software maintenance umbrella, according to the old but still important Lientz and Swanson study, we have four maintenance categories:

- **corrective** – changes in software maintenance are those that fix bugs, flaws, and defects in the software. It often comes in the form of quick, small updates on a semi-regular basis.
- **adaptive** – changes that focus on the infrastructure of the software. They're made in response to new operating systems, new hardware, and new platforms to keep the program compatible.
- **perfective** – maintenance that addresses the functionality and usability of the software. Perfective maintenance involves changing existing product functionality by refining, deleting, or adding new features.
- **preventative** – maintenance that refers to software changes carried out to futureproof your product. So, software maintenance changes are preventive when they prepare for any potential changes ahead.

In the old review, over 75% of the resources allocated were spent on perfective maintenance (customer enhancements, documentation improvements, and optimizations) and adaptive maintenance (changes in hardware, software, and data).

Technical debt
We can look at technical debt as a mix of multiple categories of debt:

- **code debt** – affecting mostly maintainability, readability of the code;
- **architectural debt** – impacting the architecture of the solution;
- **knowledge debt** – affecting the state of documentation of the system (technical as well as functional documentation);
- **test debt** – affecting the quality of the applications;
- **technology debt** – affecting the supportability and security of the system.

After having assessed dozens of projects each year, either through the architectural assessment or the IT due diligence service that Yonder provides, I find that most software solutions have significant problems with technical debt. The specific areas include code, architectural, test and technology debt way beyond a level that I am (or the project developers are) comfortable with.

1.2 Explain how technical dept relates to evolution rules defined by Lehman?

In software engineering, the laws of software evolution refer to a series of laws that Lehman and Belady formulated starting in 1974 with respect to software evolution.

The laws describe a balance between forces driving new developments on one hand, and forces that slow down progress on the other hand. Over the past decades the laws have been revised and extended several times.

Lehman Laws:

1. ***(1974) "Continuing Change"*** — an E-type system must be continually adapted or it becomes progressively less satisfactory.
2. ***(1974) "Increasing Complexity"*** — as an E-type system evolves, its complexity increases unless work is done to maintain or reduce it.
3. ***(1974) "Self Regulation"*** — E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
4. ***(1978) "Conservation of Organisational Stability (invariant work rate)"*** — the average effective global activity rate in an evolving E-type system is invariant over the product's lifetime.
5. ***(1978) "Conservation of Familiarity"*** — as an E-type system evolves, all associated with it, developers, sales personnel and users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.
6. ***(1991) "Continuing Growth"*** — the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime.
7. ***(1996) "Declining Quality"*** — the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.
8. (***1996) "Feedback System"*** (first stated 1974, formalised as law 1996) — E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

What is technical debt?

The term "technical debt" was coined by Ward Cunningham, a programmer who is also known for developing the first wiki. Technical debt is a metaphor that equates software development to the world of financial services.

Imagine that you have a project in front of you and you can develop it in two potential ways. One is quick and easy but will require modification in the future. The other comes with a better design but will take more time to implement.

In software development, releasing code in the former quick and easy approach is like incurring debt - it comes with the obligation of interest while technical debt comes in the form of extra work and money spent on it in the future. Taking the time to refactor is equivalent to paying down the principal. While this takes time in the short run, it also decreases future interest payments.

how technical dept relates to evolution rules defined by Lehman?

Lehman's laws set rules to avoid reaching this catastrophe, by setting strict laws and strictly following them when building systems from the beginning, taking into account building systems in a scientific way that can be added and modified and using the latest methods and theories to live as long as possible, and to benefit people and companies.

Preferences:

1. Software maintenance and technical debt [tss-yonder.com] [link].
2. Technical debt [Wikipedia].