Elías Llera García-Riaño
Sara Maria Ramírez Pérez

TECHNICAL DEBT

Technical debt in sense of architecture and design construct.

Book: Managing Technical Debt: Reducing Friction in Software Development

1. TECHNICAL DEBT REIFIES AN ABSTRACT CONCEPT

Technical debt is just a concept to promote the dialogue between business people and technical people within a software development project. It helps developers understand the importance of shorter time to market and the business people the relevance of early technical decisions. To achieve it, we need to consider impact over time of the decisions, evaluate their lifecycle costs and introduce mechanisms to express and estimate their impact. That is, giving concrete values to an abstract concept, which helps to make the economic consequences more real and tangible.

2. IF YOU DO NOT INCUR ANY FORM OF INTEREST, THEN YOU PROBABLY DO NOT HAVE ACTUAL TECHNICAL DEBT

Classifying an issue as technical debt depends on the business and how the context changes. There could be a part of the system that we know we can improve, but it is not important enough for us now and thus doesn't need maintenance. This means that decisions have an "interest rate". Perhaps there is no debt at a given moment, but the consequences of the decisions will appear, there is no debt until you see the consequences. That is what we call potential debt. And so, we are always incurring some interest. It is very important to understand interests and how they change along the time to manage the technical debt.

3. ALL SYSTEMS HAVE TECHNICAL DEBT

There is always a trade-off in every single system, so every single system has some amount of technical debt. It could be well managed, tending to zero (the ideal) or non-well managed when struggling with some very difficult quality or development issues. The important thing is to know the technical debt. Every software has an architecture, the difference is whether you are aware of it versus when it happens to you. The same happens with technical debt.

4. TECHNCIAL DEBT MUST TRACE TO THE SYSTEM

It is essential to associate technical debt to explicit technical debt items (specific parts of the system: code, design, test cases…). There exists a difference between the sources of friction (related to processes, people…), which causes the technical debt, and the debt itself. We need to focus on the debt itself and be able to point in the system the specific sections that need rework at a given moment. That is what we call traceability, every technical debt item should be traced to specific parts of the system.

It's very easy to suffer from the Kitchen Sink Syndrome by thinking everything going bad in the system is a technical debt.

"Get specific, get concrete and trace it to the system, not to the abstract level"

5. TECHNICAL DEBT IS NOT SYNONYMOUS WITH BAD QUALITY

There are bad qualities that are indeed a kind of technical debt, but we have tools to identify and manage those problems, such as static code analysers, quality assurance practices, defect management, quality conformance…

We even have decisions, related to technologies of the system, intentionally taken just to gain some value. This also create technical debt but provide more value than costs. What we need to do is to keep track of those decisions to keep consequences under control.

6. ARCHITETURE TECHNICAL DEBT HAS THE HIGHEST COST OF OWNERSHIP

Why architecture? Because architectural decisions, and so, architectural technical debt, have impact in almost each part of the system, as those technical debt items are extremely connected in a huge net of dependencies. Obviously, architectural decisions are quite harder to change as the system grows. So, cost in architecture accumulate as the system became harder to maintain.

7. ALL CODE MATTERS!

At first, we might think that the main source of technical debt is the code that will go in the next release version. This may be true in some scenarios, but it is important to consider that there are many more potential sources of technical debt. The code we write in tests can cause technical debt, and so does the scripts written for deployment. The same happens with configuration files or the module responsible of calling an external API. Even the code that is not put up for release is causing technical debt – leaving it out has consequences.

8. TECHNICAL DEBT HAS NO ABSOLUTE MEASURE – NEITHER FOR PRINCIPAL NOR INTEREST

A financial debt has usually its interest rate defined from the beginning. However, that is not the case with technical debt. When you take a decision, you cannot know the full extent of its consequences. That's why most attempts to make comparisons and give measures of technical debt will fail, the value of technical debt is given by a certain point in time and based on different evolution scenarios, so it will rapidly change. However, these measures and comparisons do help us to identify the most sensible parts of our project.

9. TECHNICAL DEBT DEPENDS ON THE FUTURE EVOLUTION OF THE SYSTEM

As we stablished previously, every time we take a decision, we are establishing technical debt. The value of that debt depends on the changes that will need to be made in order to "pay" that technical debt. However, this will not be done when the decision is taken, but in the future development or maintenance of the project. And when the time to make changes arrives, new decisions arise, and then the circle starts over. And that's why managing technical debt is not a one-time activity, but an ongoing process.