Technical Debt

approved by TC 24-April-2019

Definition

One Gartner analyst defines Technical Debt as "**the deviation of an application from non-functional requirements**". Wikipedia's definition states "Technical debt (also known as design debt or code debt) is a concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution now instead of using a better approach that would take longer."

Both definitions provide valuable elements that help define the types of issues that we would refer to as Technical Debt. These types of issues may have many causes, and while the particular cause may be helpful in reflecting on potential process change or even providing insight into how to address them, the important thing is to identify the debt that exists.

These items are referred to as "debt" because correcting them has a cost and will will need to be dealt with at some point.

This article discusses Tech Debt and described three types of Technical Debt which we think are useful:

1. **Deliberate tech debt**

   Often, engineers will know there's the right way to do something, and the quick way to do something, says Dag. In many cases, the quick way *is* the right way (to avoid over-engineering), but at times the team will intentionally do something the "wrong" way because they need to quickly deliver product to the market
2. **Accidental/outdated design tech debt**

   When designing software systems, Dag's team tries to balance thinking ahead and future-proofing their designs with simplicity and quick delivery. This is a tricky balance, he cautions, and nobody gets it right every time. As systems evolve and requirements change, you might come to the realize that your design is flawed, or that new functionality has become difficult and slow to implement. A good original design will often be easier to refactor incrementally, but sometimes you may have to bite the bullet and do a more significant refactor.
3. **Bit rot tech debt**

   Bit rot tech debt happens over time. A component or system slowly devolves into unnecessary complexity through lots of incremental changes, often exacerbated when worked upon by several people who might not fully understand the original design. Symptoms are, among others, copy-paste and cargo-cult programming.

Since Tech Debt is often tied to Non-Functional Requirements (NFRs), consider these categories of NFRs:

- Reliability
- Security
- Compatibility
- Maintainability (Analyzability, Changeability, Testability)
- Usability
- Portability
- Efficiency
- Functionality
- Performance
- Stability

It's easy to understand how scoring lowly on these items would be a bad thing - and constitute "debt" that needed to be dealt with. However, a lot of these are hard to measure and quantify. For example, what are our requirements for maintainability? Or Efficiency? Or Stability? Clearly, there is a point in which we'd say things are unacceptable; if it took 3 months to fix a bug, the application is not maintainable enough. If it took 17 high powered servers to host 1 user, the application isn't efficient enough. However, when does Maintainability or Efficiency become good enough - i.e., good enough that we don't have to do any more work in the short term to improve it? Our experience is that most NFRs are assessed based on gut feel and position relative to extremes, not specific metrics.

We recommend this definition for Technical Debt:

Technical debt is the implied cost of work caused by the need to revisit an existing solution due to known issues with the current approach.

Tech Council's role

The definition above uses the loose and non-specific terms like "need to revisit" and "known issues", mostly because we feel it will be impossible to list all circumstances that might cause a Tech Debt discussion. Instead, we recommend relying on the judgment of the people who are working on FOLIO to identify when things are deviating too much from what we **feel** are where we want to be.

Given that identification and assessment of Technical Debt can be subjective, the community needs to anchor its opinion. Given its role in the project, the Tech Council is the best suited to drive the Tech Debt discussions and to highlight to the community when we may be veering off our desired path relative to any NFR. As with other technical related issues, anyone in the project can bring an item to the Tech Council for consideration.

It's the Tech Council's role to assess Tech Debt and recommend action; it's not **solely** its role to identify Tech Debt - the rest of the community should raise issues relative to Tech Debt.

Not all Debt needs to be paid immediately

Another essential part of a conversation about Tech Debt has to do with payment. It's entirely reasonable to carry debt that you know about given certain circumstances. For example, if you know you are going to replace a module or application in a year, naturally you'd not be as concerned with any debt related to that module or application. Likewise, the cost to fix something may outweigh the perceived benefit from having it fixed... so you might want to wait until you get a bigger bang for your buck when addressing the issue(s).

All this to say that addressing the question about when to address Tech Debt is also a subjective one.

Process

The Tech Council will periodically (at least annually) survey the FOLIO codebase and technical environment (including existing requests and/or JIRA issues) and make recommendations to the Product Council on timetables to remediate items.

Current list

See this page to view the current list of Tech Debt items we are tracking.