Functional Debt

The internet is awash with conversations on technical debt. In comparison, little is said about functional debt with regards to Salesforce CRM even though both exist – and both have an impact on return on investment (ROI).

Salesforce investments cost money, so ideally, you want a system that works to drive up ROI. But you will not be able to see ROI go up if the system is expensive to maintain, or not efficient and users don't like it.

In this paper you will learn what you need to know about functional debt, and how to identify it within your business.

We also have two other papers covering technical debt, and CloudROI's approach helping businesses achieve technical and functional optimization.

Functional Debt

Functional debt occurs when it costs users more clicks, screens, revisions, time, and more, to use or navigate a system than it would have in a completely optimized system.

This phenomenon is often the result of forcing processes into molds and templates that result in a clunky user interface and increased time to execute daily tasks, which do not optimize the end user's experience.

Oftentimes, the developer has chosen a heavy framework to support these molds and templates, when in actual sense the project should have been small and simple.

They end up with a product offering more functionality than needed. The development and maintenance of these extras in terms of functionality is functional debt.

Functional Debt is the direct result of previous business decisions that causes a degraded ability to address current or future business goals.

– Matt Eland, 2019.

As it is, functional debt has its origin in functional aspects of the software or product. It is debt with respect to the functionality of a product.

Let us put this into perspective with a few scenarios.

Scenario 1:

You have been growing the user base for your software product, say, a web app. The development team suggests adding two sets of features that will satisfy the clientele's business needs. You not only give the green light to implement the changes, but also let this form the basis of how your software product is going to work. With that, you have decided on your general strategy for handling customer needs.

A few years down the line, and a large potential customer approaches you expressing an interest in your product. There is a catch though: the customer wants you to implement a major change to how the software product works.

Implementing this change will mean you are altering some of the decisions you already made on how your product works. This also comes with the risk of upsetting your existing user base that is already happy with the current functionality as it is.

Whatever you decide to do has functional debt written all over it. Let me explain.

You could opt to rework the current logic into what the large potential customer wants. In this case, chances are that the current users will have issues with the new way in which the product works, this creates functional inefficiencies and disrupts users, i.e., functional debt.

Scenario 2:

In a startup software company, the development team is working on a new product. Through the efforts of the sales team, a potential customer gets wind of this ongoing product development. The big prospective customer demands a feature that was not in the original product vision.

Because the startup needs this sale, the management decides to incorporate the demanded feature. Like in the first scenario above, here's where the functional debt comes in:

i) This feature will not be present in future versions, or;
ii) The company will not be selling this feature to any other customer.

In either scenario, the development and maintenance of these additional features become functional debt.

The various faces of functional debt

Functional debt exists where a feature solves some problem today. The definition of the problem changes or evolves over time, yet the functionality of the feature fails to adapt.

In his article, *Every Feature Is Considered Harmful: Debts in Software Development*, fibery.io founder Michael Dubakov defines functional debt as a scenario where "the needs of users are changing, but the system forces them to follow old flows and doesn't cover new cases".

He explains that whenever you add a new feature to a software product, you introduce debt or create room for one in the future. I could not agree more.

There is a reason why the pioneers in extreme programming introduced the YAGNI principle. Simply, do not add any functionality until it is deemed necessary. This is by itself a caveat against inducing functional debt where it could be avoided by keeping the features to a minimum.

While building a system with more features could be functionally more efficient for users, this may turn out to be only advantageous in the short run.

In the long run, such a complex system may prove less flexible as the needs of users are changing. This may lead to a situation where the system forces users to follow old flows instead of evolving with the changing user needs. This creates a functional debt in the future.

Implementing new features on an existing product or creating a new software product with more features than the users actually need – creates additional development and maintenance work. This is also Functional Debt.

Functional debt also exists where the developer sacrifices quality for the timely delivery of a cloud computing product. The functionality is rippled with cutbacks. It is only partially implemented in an attempt to deliver an application within a limited time. The most common cases of this suffer from under-simplification of functionality to meet the platform or technical specs, without taking the end process into consideration. This lack of optimization is creating a bad user experience. This is where the functional debt comes in.

Because of the bad experience or extended times it takes them to execute simple daily tasks, people do not like the system and may gradually avoid using it. Picture this: you're a business spending hundreds or thousands of dollars on a system that the target users are not using because they don't like it. This is functional debt.

How do you identify functional debt?

The first step to dealing with functional debt is to detect its existence. Here, you want to ask yourself a few questions that will give you the right perspectives.

Is your IT staff running bulk data load processes regularly? If you answered positive, this could identify that processes are not adequately handling data that could be input by your business users.

Do your business users have workaround notes that they have to use just to get the system to function the way they needed? This could signify that your product is not optimized and users are finding it hard to navigate.

Do you have low use of your CRM system? Users will often shy away from a system that is hard for them to use. As a result, you will notice a decline in system usage – pointing to a possible functional debt.

Do users often track down your system administrators for basic questions on how to utilize the system? That would be a wake-up call that something is amiss.

Do you have standardized training materials or built-in prompts that assist users through the process and require that they input the correct data? Such materials are necessary for guiding your users through the system so they do not have to track down your tech team to get help.

Do you have validation rules in place to keep your data clean? I not, you could be staring at a functional debt that requires addressing.

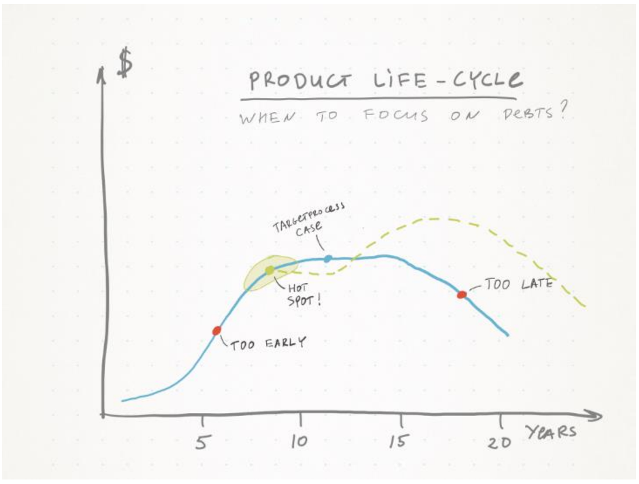
Are all users following the same process or is it difficult to get the team to align on things such as what data should be populated in the field? A streamlined system will have users following the same process in most part. It makes it intuitive for the individual to know what data to input where it is required.

How do you deal with functional debt?

Build every system with the user in mind. Aim to create an optimized system for users in terms of UI and data visualization.

When you are adding new features, do not settle for half measures. Test everything to ensure it is working to improve the user's experience. Also, you may consider adding only those features that are essential to the proper function of your product. The idea it to keep the system 'lightweight' so it is more manageable. Technically speaking, a more manageable system will deliver better functionality on the user-facing side.

Where functional debt already exists, you have got to dedicate part of your resources to repaying debt right from the time the product has shown proof of success in the market. Michael Dubakov suggests this to be after 4–8 years when the product is just headed into the "mature" phase.



Do not shy away from removing those unused or nonessential features to offload your system. How do you know when a feature is nonessential? This should be a no-brainer after you have had the product in the market for, say, a year or two. At this point, you should measure the value of various features based on how frequently they are used.

Every feature you remove will carry with it a corresponding debt reduction coefficient. So is every feature you leave out from the onset by building small systems with only the most important and required features.

In case you cannot keep your system(s) small, it will help to opt for a modular system. Modularization will allow you to repay debts more easily by rewriting modules – without affecting the entire system.

How can you start to minimize functional debt?

CloudROI takes a holistic approach to mitigating functional debt, aiming to find that optimized point between technical and functional debt.

By utilizing experts from both business and technical fields, CloudROI is able to architect a holistic system that considers all stakeholders of the system.

To minimize functional debt in your company, consider hosting a meeting or short meetings with your stakeholders and ask them for any workarounds that they use on a daily basis.

In these meetings, try to find out any red flags and troublesome processes that need addressing.

Where there is functional debt, business users may usually keep workaround notes that they have to use from time to time to get the system to function the way they needed.

While this is their way to get around the problem on a daily basis, it can form a strong foundation toward solving the problem. These sticky notes can contain lots of useful information you can use towards finding the ultimate solution.

So are the notes that your IT staff will have with them on how the system should run. These should form a perfect starting point to resolve your functional debt.

Bottom line

We focus on generating a return on investment by reducing technical and functional debt. The two – functional debt and technical debt – frequently occur together.

You will, therefore, need to get a clear understanding of both functional and technical debt to be able to minimize them and drive up your ROI – and achieve tangible growth.

We have provided a detailed rundown of all the information you need to know about technical debt, how to identify, and deal with it in our next paper titled *Technical Debt*. Enjoy.