Technical Debt

You have probably come across the term technical debt online or some other place. But are you aware of its implications on your business? Do you know you can minimize both technical debt and functional debt and drive up your return on investment (ROI) without hurting your bottom-line?

In our first paper in this three-part series, we provide a detailed rundown of the various aspects of functional debt, what causes it, and how you can handle it. In this, our second paper, we will look at technical debt with regards to Salesforce CRM.

Technical Debt

Technical debt is now popular and gets tossed around quite often in reference to anything that is customized versus configured. But what is it exactly?

The metaphor refers to “immature, incomplete, or inadequate artifacts in the software development lifecycle that cause higher costs and lower quality in the long run. These artifacts remaining in a system affect subsequent development and maintenance activities, and so can be seen as a type of debt that the system developers owe the system.” – Carolyn Seaman and Yuepu Guo, 2011.

You incur technical debt when you customize something with software code that could have been solved with point and click configuration. This customization takes more resources to maintain and troubleshoot later, making it future "debt."

Technical debt quantifies the implicit cost of additional software maintenance that crops up in the future due to the technical decisions chosen now to ship things faster.

Developers sometimes take shortcuts in implementing software codes by leaving some parts of their code pending a future refactor or code review. Because you are leaving out some otherwise standard procedures in implementing a new system or maintaining an existing one, you are able to save money and time in the process.

You buy yourself time and meet some much-needed deadline. You manage to deliver the required software capabilities faster or achieve faster time-to-market and timely software release in case of a new product. Eventually, you have got a working prototype ready to show for a round of investment – hurray.

Even better, the system is likely to be optimized right off the bat on the user-facing side, with simplified deployments.

All these are laudable outcomes. But you are creating the system outside of the parameters that your software service provider has designed it to work. You achieve optimized results in the short term, at the cost of long-term value.

In essence, you are engaging in less-than-optimal coding practices to achieve these immediate-term benefits. This approach to software development creates a kind of debt that you are supposed to repay by addressing the gaps in your software code as soon as possible.

If left unaddressed, these shortcuts tend to yield unanticipated rework costs that offset the gains of rapid delivery – this situation is what is now commonly referred to as technical debt.

Think of technical debt as the tradeoff between the immediate benefits of rapid product delivery and long-term value – because incurring technical debt may often speed up software development (in the short run).

This short-term benefit is achieved at the cost of extra work in the future.

By incurring technical debt, you assume that:

1. The debt can be repaid, and;
2. The short-term gains you are making now will, in turn, bring sustainable returns for the long haul.

Technical debt will also include those functions that are built expeditiously with little comprehensive understanding of a business' specific requirements. Over time, the business is bound to bolt on a few tweaks here and there to make the function(s) 'work.'

Failure to make these tweaks may lead to a situation where a business is paying exorbitant fees to maintain a system that is not working for its intended purposes.

Overall, the technical debt metaphor embodies the relationship between the short-term benefits of delaying particular software maintenance tasks or doing them hastily and less properly, and the long-term cost of such delays.

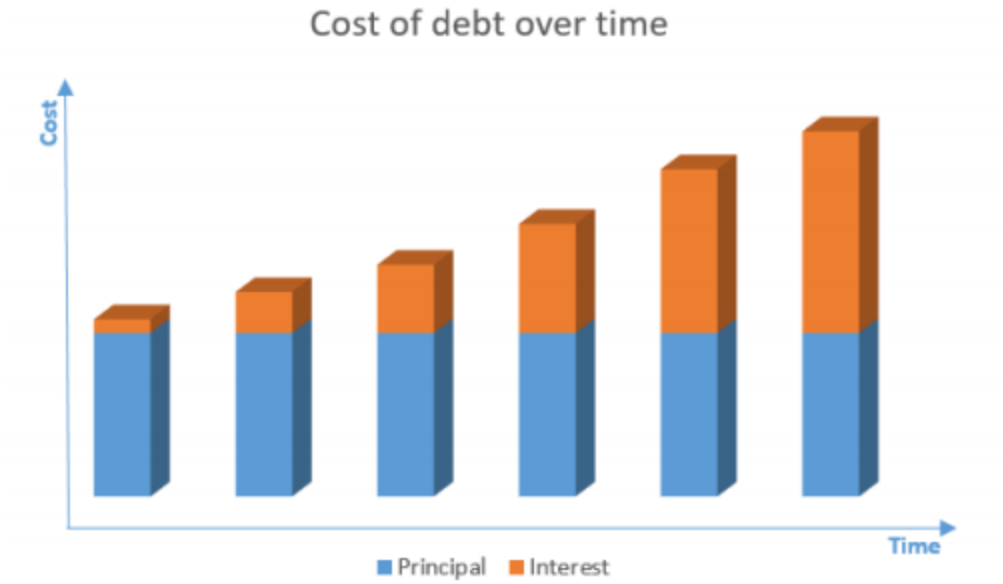
Origin of technical debt

Celebrated American software developer Ward Cunningham introduced the metaphor 'Technical debt' in 1992 when he said:

"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite..."

Technical debt is by itself not inherently bad. There are instances where it becomes necessary. The problem only starts to rear its ugly head when this debt is sustained for extended periods of time.

Just like financial debt, the longer you stay with the debt, the more expensive it gets. This is because, just like in the case of an actual loan, technical debt has interests.



All that maintenance work required in the future to make up for the technical workarounds chosen now is termed as interest. This extra work must be done. Otherwise, the interest will accrue over time and make it hard to implement changes in the future.

"…The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt." – Ward Cunningham.

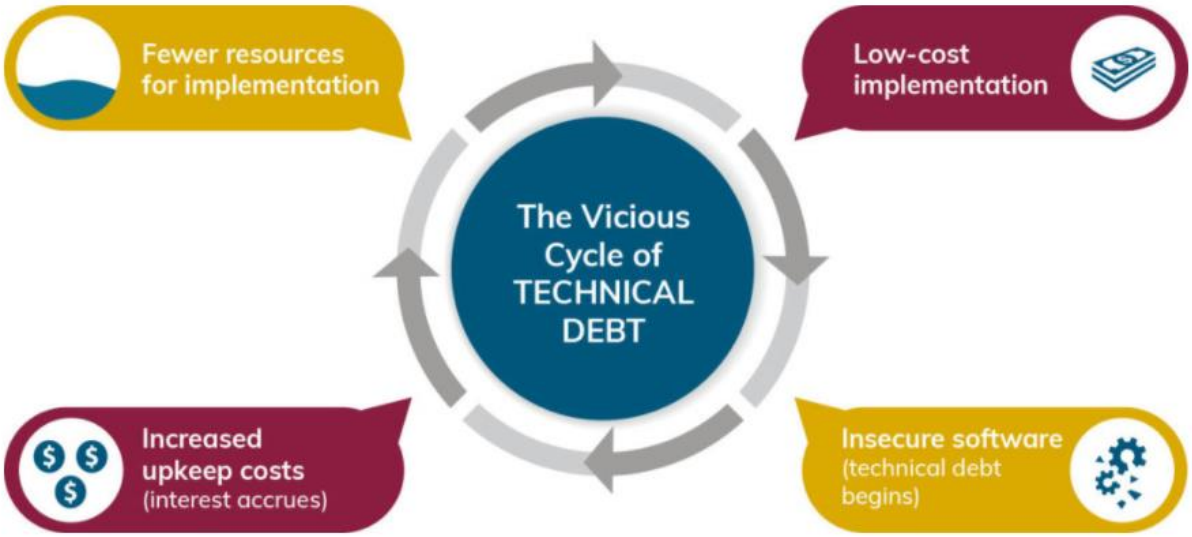
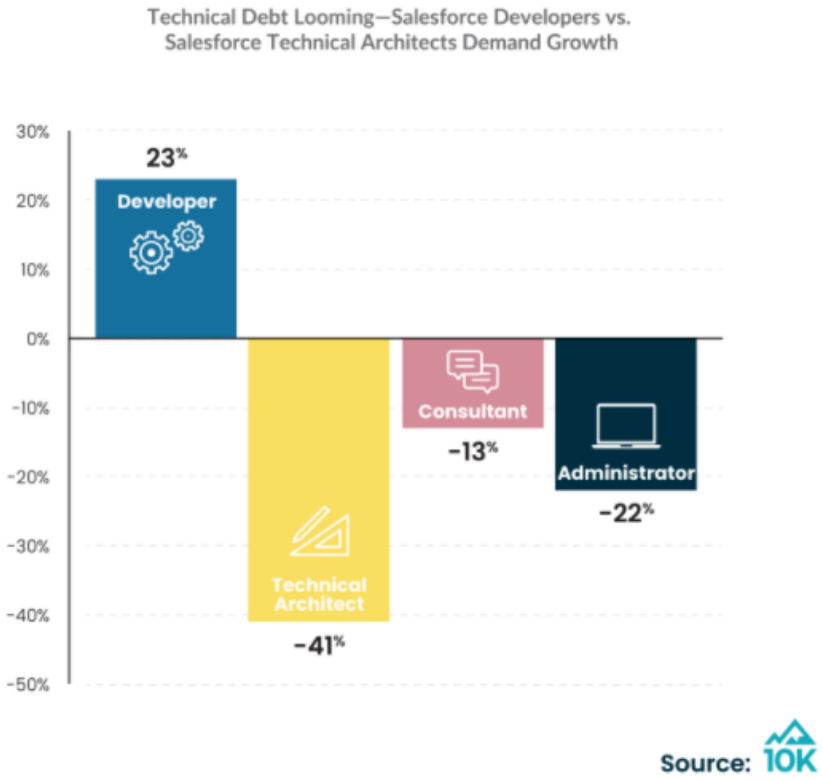


Image [source]

A looming technical debt quandary

In this [2020 Salesforce Talent Ecosystem Report], the YoY global demand growth for technical architects, consultants, and administrators took a nosedive.



This only suggests that several organizations may be headed into substantial technical debt by increasingly embracing quick-win add-ons and customizations at the expense of long-term, strategic focus.

Over time, this trend can result in an accumulation of insurmountable technical debt. Salesforce architects are a crucial cog in any company's Salesforce vision engine. Just like in the case of houses, the exclusion of an architect is likely to result in a Salesforce solution riddled with issues that must eventually be addressed at some point in the future. Needless to say, time and money are wasted in the process.

Identifying technical debt in your company

User feedback typically provides the most revealing indicator of technical debt in a company. Every software you develop is intended to serve the user. If user experience becomes convoluted or clunky, it may indicate the presence of old and unpaid technical debts.

Was the system built with a smaller user base in mind? Chances are that your system is now serving a larger group than before.

Do processes or systems fail when attempting to handle large amounts of data or records? Or, does support take up the majority of your IT staff time?

If your answer is positive for either or both of the questions then you may be looking at a technical debt situation.

The debt can come in many forms such as undocumented code, lack of training materials, workarounds that are tribal knowledge, systems that are running without any owner, systems that have not been updated within the past year, or one central source for all IT knowledge such as a system administrator.

Whatever the case, you have to act on technical debt once it becomes apparent. Otherwise it would only get worse.

What to do about technical debt

In their case study, *Managing technical debt*, Zadia Codabux (University of Saskatchewan) and Byron Williams (Mississippi State University) suggest assigning dedicated teams for technical debt reduction and allowing other teams around 20 percent of the time per sprint as a way to reduce debt.

Because technical debt includes customizations previously made where standard functionality was unavailable, introducing dedicated teams to rewrite everything can go a long way in fixing the debt.

Borrowing from Ward Cunningham, the inventor of the technical debt metaphor himself, we can pay down the principal and fix the technical debt by refactoring the now ineffective design into something better. This is one of the surest paths to greater productivity on the Salesforce solution involved. The idea is to gain by implementing a one-off reduction in future interest payments.

While you ideally must repay a technical debt once is becomes apparent in a future time, it would be better to plan its repayment right at the time you are creating it. This applies to those technical debts that have been created as a conscious decision to ship things faster now, or to achieve some other short-term benefit.

For instance, you realize that you'll need to leave out some functionality to minimize your IT spend, be sure to note this as a compromise. Then indicate when you will work on that pending functionality, paying down this debt.

This is what I call sealing off the pitfalls from the get-go. Address the technical debt factor from the initial stages of your IT project.

How CloudROI became masters of minimizing technical debt

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

By utilizing experts from both business and technical fields, CloudROI can architect a holistic system that considers all stakeholders.

What you can do to minimize technical debt

Here, the best place to start is review support ticket volumes; look at where tickets are coming from, and how long they take to resolve.

The idea is to assess what the business is flagging as the issue and determine if it really is an issue or not. That step should form a suitable starting point toward solving the actual technical problem. Even more importantly, this review will help you avoid wasting valuable resources in solving nonissues.

Bottom line

Salesforce investments cost money, so you want a system that works to drive up ROI. But you will not be able to see ROI go up if technical debt is weighing you down. Check out our next, and final, paper for an introduction to minimizing technical and functional debt; and see your ROI go through the roof.