Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-1, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

Quality code: Eliminating Technical Debt

Sanket Nirale[1], Vasudev Kawati[2] & Smitha G R[3]
[1,2]M.Tech(Information Technology), RVCE Bangalore-560059
[3]Assistant Professor ISE Department, RVCE Bangalore-560059

Abstract— *Technical debt is a system debt or code debt, it is used to referring to any system design, architecture, development within the code base, it is an short term solution to any particular job, which is applied before the complete or proper solution to any job, as per saying it is not a long term solution for any particular job. It is a kind of solution which is forwarded through the non-experts for the completion or the delivery of the product, but it is caught by the experts which may compromise the quality of the product.*

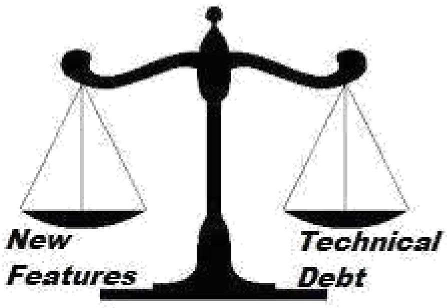
I. Introduction

Organizations must often decide whether to release a product earlier to accelerate the time-to-market factor or to release the best, smartest solution that has no open or pending issues but might take longer to release to market. They need to strike a balance between these two choices and design a plan for a quality release with the best time-to-market factor. In doing so, organizations may tolerate living with some of the known issues that might not be a priority at the time but have to be addressed in the long run. Known issues that remain in the backlog in the form of unfinished work are generally called technical debt. In simpler words, technical debt is the difference between what was initially envisioned and what was actually delivered.

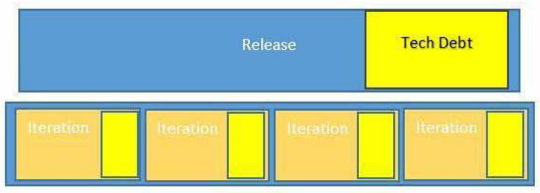
Technical debt depends heavily on whether it is associated with a new start-up project or with legacy products. Mature products have a lot of accumulated technical debt. Not all of this debt was deliberately added; it could be the result of simply not having anything better at the time the product was originally built. Typically, this accumulated technical debt for legacy products forces organizations to spend considerable time in reducing it rather than focusing on new feature development. This indirectly reduces the appetite to work on those new features that are crucial to keeping the business abreast of the latest market requirements and trends and also to remain competitive in the market.

Managing technical debt is critical to organizations with large code bases to make their products successful; they must handle a balancing act of keeping existing customers happy and attracting new customers. They must convince prospective customers to buy their products by either coming up with new features or enhancing existing features. If not properly planned and managed, technical debt can become a major blocker that can affect the organization's reputation in the long term and can have a huge impact on the existing and new customer bases. Technical debt plays a vital role in the ability to attract new customers, as they will always want to buy a product that is free of it.

Although different organizations adopt different strategies for dealing with technical debt, experience shows that the most successful way of tackling it is to budget for it in the release and in the iteration planning, as with any other budget.



Quality metrics (e.g., trouble reports) collected regularly from customers can be an important factor when deciding how much of the total capacity must be reserved for technical debt reduction. Having a specific technical debt "bucket" in the product backlog, and addressing, it will certainly add value to customers in the long run.



improvements. That way, the strategy can be effectively prioritized for giving the highest value to the organization today and in the future. A variety of items, such as enhancements to existing functionality or changes to the underlying architecture, can surface throughout the project's life cycle, which may seem important from the developer's perspective. Product management must therefore constantly strike a balance while prioritizing these items, to determine whether it's necessary to pull an item into the backlog or whether it can wait for some time.

It is difficult, if not impossible, to remove all technical debt, especially in organizations that have a high volume of legacy code. Given today's highly competitive market, and to ensure the organization's long-term success and survival, product managers must have a clear road map that includes a sound plan for simultaneous investments in both innovation for new features and improvements to existing products.

II.IMPLEMENTATION

One effective way to control technical debt is to prevent further accumulation. This is accomplished by creating buckets by estimating the average rate of debt discovered in the previous release cycle. This way, if the incoming rate of technical debt remains the same in the current cycle, the reserved bandwidth would ensure that the technical debt does not increase further.

In addition to focusing on reducing accumulated debt, organizations might encourage teams to develop a process in which a review and an update to the Definition of Done is done regularly to prevent the accumulation of debt in the first place.

After having decided to reduce or eliminate technical debt over time, organizations face another major challenge: prioritizing it. Any technical debt reduction must be justified. Is it worth the effort and does it really benefit the customer, or does it increase business prospects for the organization? A good strategy from product management is essential in evaluating true debt versus structural Technical debt can bog down any organization that attempts to be agile. If too much of the IT budget is spent on maintenance and not on innovation and development, productivity will decline sharply. The example used in this post is Telefonica in Spain; the company freed around 14 billion Euros and 18% of the total IT budget after they had removed thousands of legacy systems weighed down by technical debt. This demonstrates the monetary value of reducing technical debt within an organization. However, technical debt is not only detrimental in the sense that it constrains an organization’s budget, but also in that it is an impediment for developers because they are building on a less than solid foundation.

Developer productivity reacts negatively to technical debt as demonstrated by the following formula:

Developer Productivity = 1? [Size (technical debt)? size (system)]

This formula may not reveal strongly enough the negative effects of mounting technical debt on productivity; because as system size and technical debt grow productivity is likely to go below zero. Furthermore, any attempts to improve the situation can often result in more problems than in any improvement. Therefore, when technical debt consequences reach such heights what can be done to resolve the situation? There are three steps proposed in this post to remedy technical debt build up.

1. Keep track of it

Before engaging in managing technical debt, there must be a keen awareness of how tech debt affects the process outcomes at an organization. For example, is most of the IT budget spent on infrastructure and maintenance? Is there a large amount of innovation that needs to be pushed out, but only a small amount of innovation is able to be pushed out through a funnel? These are the type of metrics that are needed to solve the problem. There are several services that can provide these metrics. One mentioned in this post is CAST Software. CAST implements static source code analysis in order to produce an amount of total technical debt in the system under study. This enables the organization to compare and trend technical debt across applications.

2. Avoid it

If there is a vested interest in monitoring technical debt, there will also be a desire to avoid tacking on more debt onto the system. In order to avoid adding more debt, some key agile practices can be utilized, such as: continuous integration, automated unit testing, refactoring, complete feature testing, and test driven development.

3. Reduce

After identifying and preventing more technical debt from being added, the process to begin technical debt reduction can begin. The scenario drawn out in this post, in order to demonstrate some methods for tech debt reduction, consists of four sprints prior to release. Given this scenario there are three strategies that can be used to reduce technical debt.
The first strategy is to fix all bugs in the first sprint before building any new functionality. The benefits of this method are that it promises a stable foundation to build upon, bugs that are fixed sooner

rather than later are easier to fix, and after all bugs are dealt with there will be less risk present for the rest of the schedule.

The second strategy is to enact ‘hardening sprints’ in which a team will gather all bugs that were discovered during production and previous sprints, and then attempt to restore the product to an acceptable state in order to release it as planned. This, however, is not the best option and is a possible signal that there are other issues at hand that caused the implementation of this strategy.

Lastly, the third option would be to do a little bit of technical debt reduction work in each sprint. This method allows for small amounts of technical debt to be removed while also being able to provide new functionality within each sprint. The reason for why continuous delivery of functionality is important is that if any team refrains from delivering functionality for some time it will soon be the case that what they are producing is no longer relevant. It was found that integrating a prioritized list of technical debt items parallel to the development backlog and then focusing a percent of sprint effort or every nth sprint to technical debt reduction is the best strategy for technical debt management.

A clarification is made at the end of this post – that technical debt remediation is not inclusive to solely bug fixing. Bug fixing is an issue that must be taken care of as soon as possible in order to continue on to new things, while debt reduction is focused on reducing risk and not simply resolving such problems.

III. QUALITY ANALYSIS

How do we feel about the quality of our code?
Everyone rates it on a scale of 1-5, where 5 means “It’s great, I’m proud of it!” and 1 means “Total crap”. Compare. If you see mostly 4s and 5s, and nothing under 3, then never mind the rest of this article. If you see great variation, like some 5s and some 1s, then you need to explore this. Are the different ratings for different parts of the code? If so, why is the quality so different? Are the different ratings for the same code? If so, what do the different individuals actually mean by quality?

Most likely, however, you will see a bunch of 2s or worse, and very few 4s and 5s. The term for this is Technical Debt, although for the purpose of this article I’ll simply call it Crappy Code.

Do we want to have it this way?

If not, what do we want the code quality to be? Most developers want a quality level of 4 or 5.

Yes, the scale is arbitrary and subjective but it’s still useful. If opinions vary strongly then you need to have a discussion about what you mean by quality, and where you want to be as a team. It’s hard to fix the problem if you can’t agree on where you want to be as a team.

What is the cause of this problem?

The first step in solving technical debt, is to admit and accept this fact. “but wait, we inherited a bunch of crappy legacy code. We did NOT write it!”

How to use Technical Debt the Right and Wrong way?

After looking at the above definition of technical debt you may be thinking that this type of debt is an inherently bad thing, but that is not the whole truth. At times the cost of taking on technical debt is less than the cost of releasing functionality at a later date but delivering higher quality code. Under these circumstances it is helpful to look at how the financial world treats debt as a useful tool. If you mortgage a house after having studied how to pay it back, this is seen as a prudent use of debt and a useful investment. Saving up to pay the entire amount of a house upfront is not a feasible reality for most people. However, if you are continually using your credit card with abandon this is not a good financial practice. The same premise is applicable in software development.

IV.CONCLUSION

Using a technical debt in many cases may possibly arise different kind of problem are need to face by technician, so eliminating technical debt stuffs in code produce good quality project by using simpler code and natural language that everyone can understand.

V.REFERENCES

[1] Frances lash “How to avoid the brittle code of technical debt” April 26, 2016 available in “www. ontechnicaldebt.com”
[2] Ramesh Lakkaraju “Reducing Technical Debt” 28 August 2015 available in “https://www.scrum alliance.org”
[3] Matthias Marschall on “Technical Debt” Sep 30, 2015 available in “http://www.agileweboperations. com”
[4] Bastien Gallay on “How would you go about reducing technical debt in an existing software?” May 8, 2013 availble on ”https://www.quora.com”