Filip Falinski

A report on

**TECHNICAL DEBT**

**What is Technical Debt?**

The Term “Technical debt” originated from Ward Cunningham who first used the term when describing why recourses need to be budgeted for refactoring to stakeholders. Technical debt is very similar to financial debt in that to the person borrowing the funds it is more important to have the item now rather than save up for it themselves and purchase it at a later stage. In software Development, technical debit ( also know as “tech debt” or “code debt”) is when a company needs to deliver a product fast in order to meet the deadline. This approach is neither bad or good, it solely depends on how it is dealt with, Some companies take tech debt intentionally and strategically meaning that they are aware of the consequences and are prepared to deal with them this is referred to as “true Tech Debt”. While other companies might be pressured by deadlines and competition and take the debt in order to go the easy way rather than taking the long way with releasing poor code in order to achieve gains. Now technical debt does not necessarily mean that the code or the project will be a mess, like mentioned before if it is approached with the right plan it can be risky but beneficial to the company.

**Minimizing Technical Debt.**

Companies using agile development cannot avoid technical debt but the can minimize it so that it does not hinder them in the future. The first step to minimize tech debt is to identify the source of the debt and its cause. This needs to be acknowledged by the team and they need to adjust to it. When trying to minimize the debt it is not always possible to fix all the issues that cause the debt, meaning that the highest risks must be prioritized first. When the high risks have been identified, the second part comes into play which is to execute the risks. Once all the issues have been prioritized they need to be executed and adjusted to meet the business. Another method that can help reduce debt is to use automated testing as it can identify code defects in early stages. After the issues have been identified and solved the team needs to plan out for the future aspects of the development and how they can adjust so that similar problems don’t occur and they can work efficiently without worrying about more technical debt.

**Technical Debt Management.**

Management of technical debt is the work that builds when quicker and short term solutions are implemented instead of optimal solutions. Like mentioned before technical debt isn’t always bad it can really beneficial to a company when they want to get ahead of other companies and send their applications to the public market but this is only when it is managed correctly. It might not be the prettiest solution in terms of code and other software terms but it can help generate revenue which than will help to sort out a schedule and address and rework the issue the were present on launch. One of the major way to manage this debt is to talk about it during every meeting with the developing team, it is more than likely that they are aware of the debt and can provide their own input on it which will help manage it. This will help in the long term as it will help the manager know when to take on more debt or when to slow down on it to sort out current issues.

Being organised is very important in managing tech debt, similar to talking about debt at every meeting it is also crucial to talk about what happens if the technical debt is not addressed and the problems it will cause in the future. That is why creating a schedule in the developing cycle that has area that are dedicated to testing, bug fixes and running maintenance and dealing with the technical debt insures that the team is dealing with the faults and weaknesses while maintaining to the work schedule. Giving the task of tracking the technical debt to the development team is also a good practice, making the team list the debt to the backlog makes sure that they know that it is as important to deal with as it is to implement new features.

The last way to manage any tech debt is to plan and create reasonable sprints. Asking too much of the team is the easiest way to build up code debt, so by communicating with the team and planning out sprints that will produce efficient amount of work is a good way to avoid more debt. Sometimes it will be inevitable to build it up but if the rules listed above are followed that debt can be dealt with along the way.

**What Causes Technical Debt?**

It is almost inevitable to not encounter tech debt, simply because Technical debt occurs when the design and implementation collide with the business aspect of the project and deadlines. This means that there are many causes which can lead to tech debt, it can happen when certain development work needs to be cut short by the developing team such as writing clean code or writing proper documentation. Some of the most common issues that develop into technical debt are, ‘Business pressure’ this is when a company is under pressure from stakeholders with meeting their deadlines or meeting their contract agreements which leads the company to make decisions such as producing poor code, reduce testing and skip other aspects of the development which will lead to the company loosing productivity. Another cause is ‘Incompetence’ this is when the developer has no idea how to write proper code and proceeds to take the easy way out and rush production with poor and messy code. The list of what causes code debt is constant which is why it is inevitable to avoid it. Identifying the issue in the early stages of development is the best solution as it can be dealt with but more the project mature it becomes more difficult to manually identify and manage the technical debt.

**Way’s Technical Debt Can Impact Your Business**

Technical debt can become a very big problem if it is allowed to get out of control. If this happens it creates problems for the developing teams as they spend most of their time paying interest for past projects instead of working on new functions or implementing updates. The debt must be paid of eventually and this will be at the cost of new products or new updates which slows down progress because instead of making new things the team will be fixing code so that it can be readable and easy to work with in the future.

Speaking of code, another effect of tech debt is poorly designed code. If a developer takes the easy and quick route to release a product than it is more likely that the code will have a messy structure. This extends to poor design, poor design might mean a couple of things. One is the use of ne style for a block of code and then not following that same behaviour throughout the project. Anther is not using descriptive names thus making it hard to know what a method does just by reading the name of it and instead having to go through the code to understand it. These a just two examples of poor design, now doing these things might get the project complete in time but it will be difficult for new or other programmers to work on the product, which at the end will increase the technical debt as it will take more time for programmers who have not worked on the code before to understand it so they can either fix bugs or implement new features.

Releasing a non-perfect product can be a either a good thing or a bad thing as it can give the company a boost in income or do the complete opposite and put the company in a very difficult situation. Producing a rushed product just to meet deadlines can bring consequences. These might not appear on release day but will eventually pop-up, including bugs and crashes. These issues will have to be fixed by the development team which will again increase the technical debt. By taking the quick route and writing dirty code the team will have a much harder time to find these errors and fix them which of course takes time, an error take could have taken two weeks to fix if the code was clean might take double the time to sort out due to messy code. This again will increase the tech debt which again slows down the productivity of the company.

Tech debt will drain the companies productivity. It will lead to slower and slower output, It will not only slow the team but it will slow the entire production cycle. Code debt will also impact the testing team. Test will have to be sped up in order to meet deadlines. If this happens than there will be tests that will be skipped or missed by the team and in doing so the debt will be increased as these test will have to be tested. The team will have to go back and find which tests have been missed and run them further increasing the debt and slowing down the project.

**Measuring Technical Debt.**

Making sure that technical debt it measured is a vital part of any project, this allows the team and manager of the project to see if they are in the green and are working efficiently or are starting to fall off and rack up debt. There are a number of ways that technical debt can be measured one good practice is to make note of bugs. Making sure that the engineers are keeping a tally of bugs they encounter makes it easy to see what bugs are fixed and which aren’t, by doing so it shows if work is being done in a efficient way or if the technical debt is increasing.

Complex code like mentioned before can be a sign of technical debt. Code written should be self-explanatory, meaning that if there are multiple comments next to the code explaining what the code does is a sign of complex code. There will be situations that code will need to be complex but other than that it should be kept simple, there are a few techniques to avoid this such as Cyclomatic complexity, Class coupling and depth of inheritance. Making use of these techniques will lower the chance of increasing the debt by making code complex.

There is a term which is ‘Cycle time’. This is a metric that is used to measure the time that passes from each commit to deployment. Doing this makes it easier to measure the technical debt as it shows how long it takes the engineers to implement a new function or fix an bug or error. If a bug or error is taking a long time to fix by the team than it might lead to a bigger problem which will increase technical debt.

A different way to measure tech debt is by using the code churn metric. This metric counts how many time code has been either deleted, replaced or rewritten in a line of code. Code churning is sometimes inevitable but when the project is deployed and any bugs are fixed this should start to reduce. If it churn is still happening than there is some sort of issue. On the subject of lines of code another metric is Code Coverage. This measures how much code is being executed during testing, this shows if the code is clean or messy. The more unused lines of code means that it is poorly written and more work needs to be done.

A personal way for making it easy to measure technical debt is by making use of Code Ownership. By having too many engineers work on a single task is an easy way to make a mess of things, this does not mean to have one person work on a project as that can lead to big problems such as over stressing the person or if the person does not show up to work leading to the project not meeting the deadline. It is a good practice to have a list of the people assigned to each task and what project they are working on.

The Main Technique to measure technical debt is by using the Technical Debt Ratio(TDR). This was specifically created to measure tech debt, the formula is pretty straight forward, it is ( Remediation Cost ÷ Development Cost ) x 100 = TDR.

Where remediation cost is calculated as a function of the code quality and development cost is divided by the resources, when putting the two together it should roughly show how much resources will be needed for a project. The ideal TDR is about five percent.