offnotes.org

JAN 3, 2022 • 9 MIN READ • NEWSLETTER

The Economy of Tech Debt

All you need to know about tech debt along with an hands-on approach to manage it effectively in your development cycle.

Most articles about tech debt are pretty much theory only with lots of interesting philosophy. I'd like to approach this topic in two steps:

- The *first part* of this article is all about theory: what's a tech debt, what's not, what's the root causes and impacts on product and team.
- In the *second part* I'm going to give you a **hands-on approach** on how to manage it so you can improve your product time-to-market.

What's Tech Debt

The Economy of Tech Debt

instead of using a better approach that would take longer.

It was introduced by Ward Cunningham with the following words:

> ***“Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with refactoring. The danger occurs when the debt is not repaid.”***

So, in fact:

- Is not something necessarily *wrong*, *nor it is inherently right*
- Is *potentially* dangerous when left unpaid (it can stack up quickly and easily)

This means we can accept some degree of tech debt during the development and be sure to constantly keep it in check.

What's Not Tech Debt

It is often much easier to put all the mess under the tech debt umbrella but there are some stuff **which are not eligible as tech debt**, like:

- **Legacy Code**
  As long it's tested/properly written, *legacy code is just a boring code* you should have not fun to rewrite once again.
- **Trending Techie**
  Don't be such a *baby,* if it's not the FoTM it's *great* anyway.
- **Multiple Design Patterns**
  As much you can appreciate a common design pattern inside a product,

The Economy of Tech Debt

- **Code Guidelines**
  Same of above, feel free people having *some uniqueness*

Measure It

Before you consider any action you need to measure it: a good metric to evaluate tech debt and plan any action is the **amount of unplanned work**.

> **Not all of the technical debt is worth repaying**
> **Usually when the time spent refactoring exceed the benefits it's probably a better idea to just ignore the debt *(note: you should still keep the track of it, where it lays and why you have decided to not address)*.**

Unplanned work is any task due to bugs, service interruptions, or flawed software designs.
Statistically has been shown that **unplanned work should represent from 5% to 15% of the work of an organization**.
When it goes higher it should be *accurately monitored* in order to avoid further problems.

Want to read more?
Subscribe for free to read the entire article
👇👇

Root Causes

Unplanned work, and therefore tech debt, have 3 different root causes:

everything is bound to collapse. However, time pressure lead inevitably to corner-cutting, which means tech debt. Beware artificial ship dates because they leads to hidden cost.

- **Team Changes**
  Teams are living creatures, they change over the time *(and that's a good news, indeed)*. When new people comes into the team, they typically have steep learning curve to learn what has come before. Learning is far from being perfect.
- **Pivots**
  Engineering teams hates that, but pivots are a fundamental part of business reality. The alternative to pivoting is flogging a dead horse. However pivots lead to stretching existing implementations to fit new objectives; eventually things may break.

Continuous accrual of technical debt leads to a vicious cycle: large technical debt reduces the productivity and morale of the team; at the same time low productivity attracts management push for more features and postponement of technical debt issues, which in turn further increases the technical debt.

Business Impacts

the role tech debt plays in a healthy and successful product.

> **Mounting tech debt means spread of unmanaged complexity, resulting in less business value being unlocked.**

Why?
Simpler tasks require unnecessary cognitive load which means more *[shitty]* work to get done and *obviously*longer delivery times. Moreover, result is anything but good and the situation *is going to get worse once new debts are going to be added.*

Impacts are on tech side:

- **Performance/Stability**
  The responsiveness and the overall stability of the application
- **Changeability**
  Likelihood of introducing new defects while chaning existing feature or adding new ones
- **Security**
  The ability of your application to prevent unauthorized intrusions

... but also *for team:*

- **Morale**
  The mess of tech debt slows developers down and sometimes even prevents them from doing their job. It can make difficult to be proud of the work you're doing as a lot of time is lost dealing with annoying problems *(moreover invisible to product stakeholders)*.
- **Transferability**
  The ease with which new developers can onboard into the product.

51% of engineers consider quitting their job due to tech debt (read more)

> **A less performing product means losing business to your**
>
> **competitors. Technical debt can become a silent killer of your company revenue.**

Technical debt can become a silent killer of your team productivity.

A Practical Approach

Learn how to Communicate, Manage & Minimize the Impacts of tech debt in your code.

The Economy of Tech Debt

Communicate Effectively

Discuss about tech debt to non tech people *maybe hard.*

> **The secret here is to speak the language of business: Persuade your stakeholders that time improving the product behind the scenes will increase productivity, sustainability, and scalability.**

- **Focus is always on cost and time instead of quality**
  Unmanaged tech debt increase the volatility in the marginal cost of features (aka. longer development times for new/existing features).
- **Collect Data**
  Measure measure measure: the number of bugs reported and accumulated per per week, emergencies that occur per week and so on. The point is to deliver your message along with measurable data. *Because data are money*.
- **Invest to save**
  Investing 10-15% of our time in code quality help to reduce developers turnover. *How much does it cost us to replace a leaving dev?*

Make the waste visible

The difference between financial debt and technical debt is visibility.
It's not always clear how much technical debt you have at any moment in ti
A tech debt follow 3 different stages:

- **Unknown**
  As you may imagine this is tech debt that you simply haven't identified and have no idead it even exists. Uncertainty makes this debt the **most dangerous typ**e: you can't consider it in future tasks without knowing of its existence. This is the reason of missed deadlines, blocks and unforeseen issues.

- **Dormant**
  You know about it but you don't plan to address it right now. You can spot this kind of debt while you are working on something, eventually in some old codebase. The best thing you can do is to **take a note** and face it later in the distant future. **It's perfectly healthy to have.**
- **Active**
  You know about it, it's defined and you are able to plan a schedule to fix it in the foresable future. A common example you may incur active tech debt is when introducing a new feature following a strict deadline; the result maybe less-than-ideal but you plan actively fix in a minor update after the release. **Also Active Debt is perfectly healthy to have.**

The pyramid of tech debt. The lower Unknown level is the most dangerous type.

You must consider your product as a living ecosystem.

As tech team you're like gardeners managing a green park along each season of the year. It's springtime: you enthusiastically spend a large sum at the garden

center and work your soil adding new plants *(features)* and architecting each garden *(organizing your infrastructure)*.

> **Discipline is a fundamental part of your work.**

You must be diligent, regularly tames and tends their urban oasis, ripping out weeds, and ultimately ending up with a beautiful park *(congrats, you are maniging tech debts!)*.
Failing to do regular maintenance, you eventually end up with a mess of dead plants everywhere, a dense forest and frustration.

> **The goal is to make the waste visible, then schedule a plan to fix them.**

Like working a garden, software needs time and resources to keep it sustainable and to avoid long-term costs.

> **The biggest symptom of tech debt in an average software is a specific issue or issues correlated to each other** (by technical dependency) **with a moderate/high occurrence-rate.**

Hands-On Approach

As any good plan it starts measuring the impact.

You can’t plan or prioritize anything without knowing its impact.

Critical tasks that are easy to fix will be done first, while very valuable fixes that are more difficult to do will be started but take longer to complete.

1. Track & Grade Your Debt

While you can't totally avoid it, your job is to monitor where is easy to found. Once you spot it take a deep breath and avoid to fix it immediately.

Annotate it in your backlog along with a meaningful description or a grade.

> **(*light*) to 5 (*heavy*).**

- **Severity**: defines how much the user is disturbed/prevented from using your product. Show-stoppers or tech debt inside core parts of any foundation layer should be addressed urgently (level 5). Users rely on it and, you are likely to want to continue to develop those features and certainly continuing to build on shaky foundations is far from an optimal solution.
- **Occurrence:** the amount of times any user encounter the issue. Be sure to correctly link the tech error with the issue (1-5 at least in top 5 of tracking occurrences)
- **Dependency**: How much the issue is linked to other dependency (aka. you should also touch other parts of the code).

Now put all issues in an ordered list with those total grades.
That's your priority.

2. Allocate time in each cycle

In my team we have reserved part of any sprint (2 weeks) for addressing tech debt. The amount of this time may vary depending on a lot of factor (how big the activity is, pressure to deliver other product related features...) but you should consider a dedicated slot (~20%) of each iteration. If you can't commit to a fixed amount of time each iteration use the end of a quarter as suitable time to do this.

3. Beware Artificial Deadlines

Deadlines can be real or artificial. Most of the times artificial deadlines are used

to set some boundaries for all all people involved. However the hidden cost of an artificial deadline is a tech debt: while you think you can address a debt just after the release, instead you may end using the time supporting early changes an

Minimize Impacts

I'm pretty sure you land in this paragraph hoping for a simple and fast answer.
At this time you should have learned you can’t avoid Tech Debt completely.
It’s an inevitable part of shipping a product.

But don't be discouraged, it doesn’t mean you can’t take some actions to minimize it:

- **Consistently refactoring the codebase**
  In our team we often jokes about continous refactoring but a disciplined approach to refactoring leads to a codebase that is low-maintenance, highly readable as well as highly functional, all while bringing down technical debt.
- **A common known enemy**
  Engineers waste more than seven hours per week on tech debt which is often invisible to stakeholders. Communicate the value of paying down technical debt to the product owner and stakeholders in your organization to make sure you have a common understanding.
- **Be aware of ninja devs**
  Find talent that can create a working solution is not enough: it is also necessary to find talent that diligently creates a solution that is as sustainable as it is functional. Be aware of the ninja devs in your team.
- **Encourage cross-collaboration**
  Everyone should have a robust understanding of how core stuff works in the project; pair programming and cross-reviews must be part of the workflow.
- **Take time**
  Expect to spend a chunk of time/money after each major feature release to address the Tech Debt accumulated during the push to get the feature over the finish line.

- **Don't write twice**
  MVP (Minimum Viable Product) are critical part of each fast-moving business. You can’t afford to write things twice: produced code should b

plan even for an MVP. This doesn't mean over-engineer; it means engineer once.

Key Takeaways

- Tech Debt is an evitable output of any development activity. Not all of the technical debt is worth repaying (see time/benefits - checkout this article)
- Consider investing 15-20% of each iteration for maintenance and improvements requested by your team (read more)
- Tracking & classification is fundamental, instill good habits in your team (read more).
- Speak to non technical stakeholders using business terms (check out this article for some examples)
- Tech Debt also impacts on team morale (see "State of Technical Debt 2021")
- Propose a Tech Wealth culture in your organization

SHARE TWEET