What is Technical Debt and How Should You Manage it?

Published by FeldsparTech Solutions

A debt needs to be managed diligently. It is a burden whether on an individual or a company. Leave it unattended and it can grow to crippling levels. Technical Debt just like the Financial Debt can adversely impact businesses. But what does the term 'Technical Debt' imply?

It is important that all organizations and especially IT leaders know about Technical Debt and its impact.

Table of Contents

Intended Audience

This white paper is intended for Business and IT leaders and managers. It provides an understanding of Technical Debt and the various ways it gets created in an organization. It also details approaches to managing Technical Debt and sheds light on new technologies available to keep the cost of managing it to a minimum.

The objective is to provide necessary information to leaders to be aware of Technical Debt. This in turn would help them make informed decisions within their respective organizations.

What is Technical Debt?

A Software Development Project that meets its deadline is a rarity. Dealing with software is dealing with the abstract and the intangible. There is no straight line to tread. Assumptions can go wrong and good intentions do not matter much. Such projects invariably face situations that threaten to derail the projects from meeting their goal. Compromises and alternatives that help keep moving and inch towards the finish line are, therefore, willingly grabbed!

Easy options or shortcuts that we are sometimes forced to take while delivering code, directly impacts the quality and performance of software. A poorly designed system or badly written code results in suboptimal software solutions. **Technical Debt** is a measure of such inefficiencies in the software. It has to be repaid by 'spending' more effort year after year just like the interest payment on financial debt.

Companies have their own reasons for the haste and compromises they have to make with the software:

1. Pressure of Timely Delivery
2. Complex and Varied programming platforms
3. Lack of Experts
4. Undefined Processes or lack of adherence
5. Budgetary constraints

But isn't this the norm? It is no secret that projects mostly take more time than estimated. Good resources are always difficult to find. Budgets seem less than what one would like to have.

Aren't these part of doing projects and running a business?

Let's examine Technical Debt in a little more detail and try to understand its impact.

More About Technical Debt

Technical Debt can also be defined as ‘rework’ that needs to be done on a software system to keep it effective. This is the extra effort that has to be put on the system. This effort was never planned or desired, it does not provide any additional value. It is a cost that has to be incurred due to informed or uninformed decisions made in the past.

Examples of Technical Debt

1. **Incomplete components:** Components that need rework, without which you cannot progress and implement other changes.

2. **Workaround:** Extra effort required to implement something new in a legacy setup.

3. **Lack of documentation:** Incomplete documentation that makes it difficult for new team members to understand the system and implement new changes or fix existing bugs. The additional documentation to be done to ensure proper understanding for new team members is Technical debt in this case.

4. **Refactoring:** Duplicate, unorganized, suboptimal and difficult-to-maintain code needing attention and effort. This occurs when a team keeps writing code for all new requirements without proper analysis of the architecture, design and already existing components.

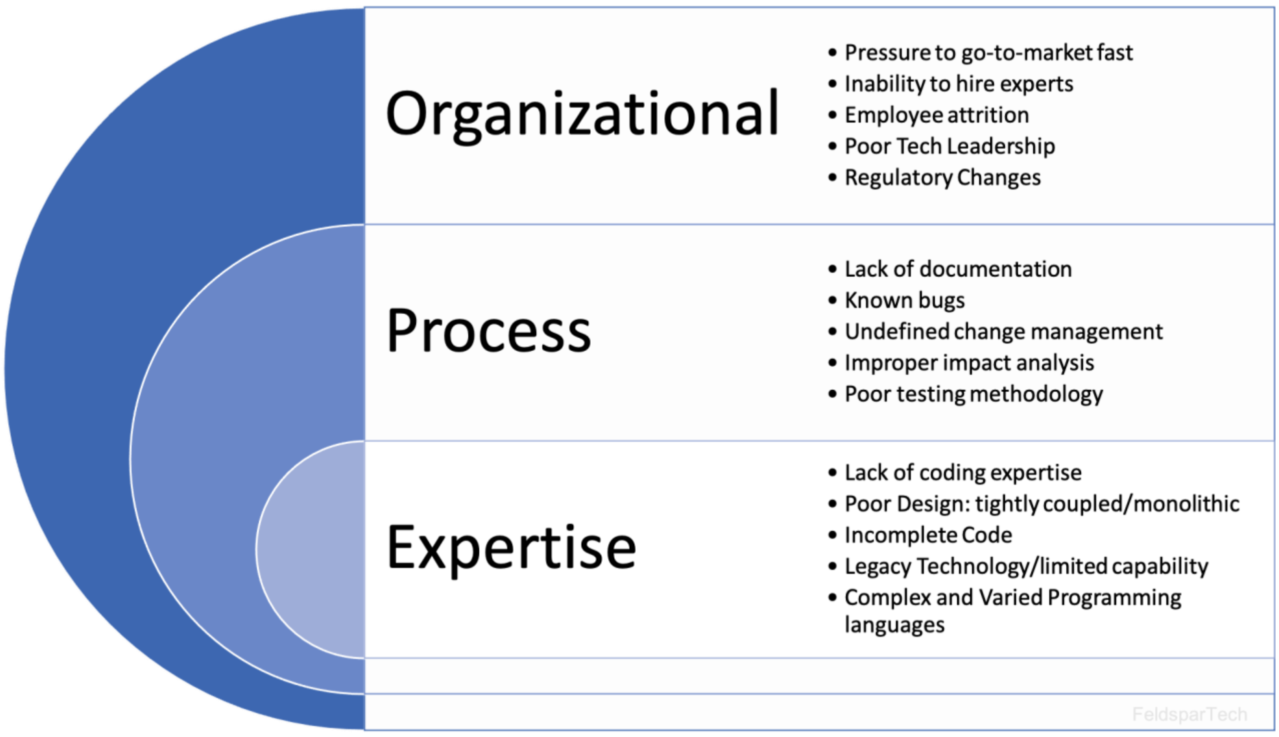
It is important to note that **Technical Debt** is different from **Maintenance Effort** and **Effort for New Work** like **R&D and Innovation**.

Maintenance effort is required to incorporate changes due to changes in business processes or regulatory requirements. It could involve new integrations or upgrades the system to scale up performance. Addition of features and functionality are usually take up as part of maintenance. **R&D effort** is for new offerings and unique features that the business may want. Most organizations plan and budget for Maintenance and R&D spends but not for Technical Debt.

Technical Debt is an undesired spend hence organizations tend to ignore it during planning.

Why Technical Debt

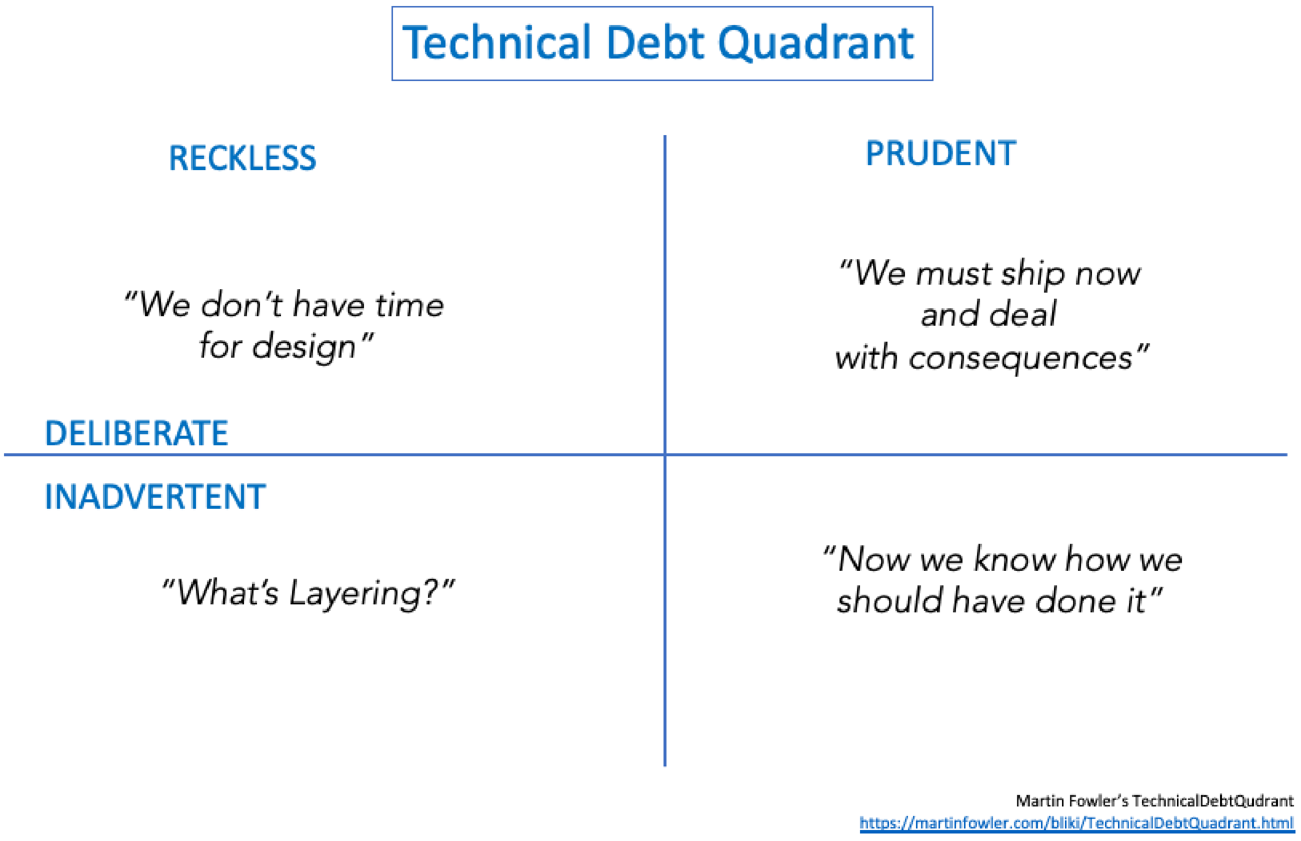
Let’s look at the factors contributing to technical debt in a little more detail. They can be categorized into three main buckets:


Organizational
• Pressure to go-to-market fast
• Inability to hire experts
• Employee attrition
• Poor Tech Leadership
• Regulatory Changes
Process
• Lack of documentation
• Known bugs
• Undefined change management
• Improper impact analysis
• Poor testing methodology
Expertise
• Lack of coding expertise
• Poor Design: tightly coupled/monolithic
• Incomplete Code
• Legacy Technology/limited capability
• Complex and Varied Programming languages
FeldsparTech

Technical Debt can result due to reckless behaviour of teams or could be due to a well thought-out decision. It can result due to decisions where despite knowing the right thing the team chooses not to do it due to existing constraints.

There also are many instances where Technical Debt is inadvertent when a team makes the best possible effort, still makes mistakes and learns about a better approach only post facto.

Martin Fowler lists the reasons for Technical Debt in his famous [Technical Debt Quadrant](https://martinfowler.com/bliki/TechnicalDebtQuadrant.html). According to him Technical Debt can be due to reckless/prudent and deliberate/inadvertent behaviour.


Technical Debt Quadrant
RECKLESS
PRUDENT
"We don't have time for design"
"We must ship now and deal with consequences"
DELIBERATE
INADVERTENT
"What's Layering?"
"Now we know how we should have done it"
Martin Fowler's TechnicalDebtQudrant
https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Consequences of Technical Debt

Technical debt can have serious impact on an organization if left unattended:

1. The Technical Debt for most organizations increases with time. One simple reason is that as systems get older, there is more effort to extract value out of them. All the pounding of system changes and band-aid solutions weigh heavily on the software solutions over time.

2. Team morale takes a blow if the system is inefficient. A lot more effort needs to be spent on achieving very little output. Productivity of the team declines resulting in demotivation across IT organization.

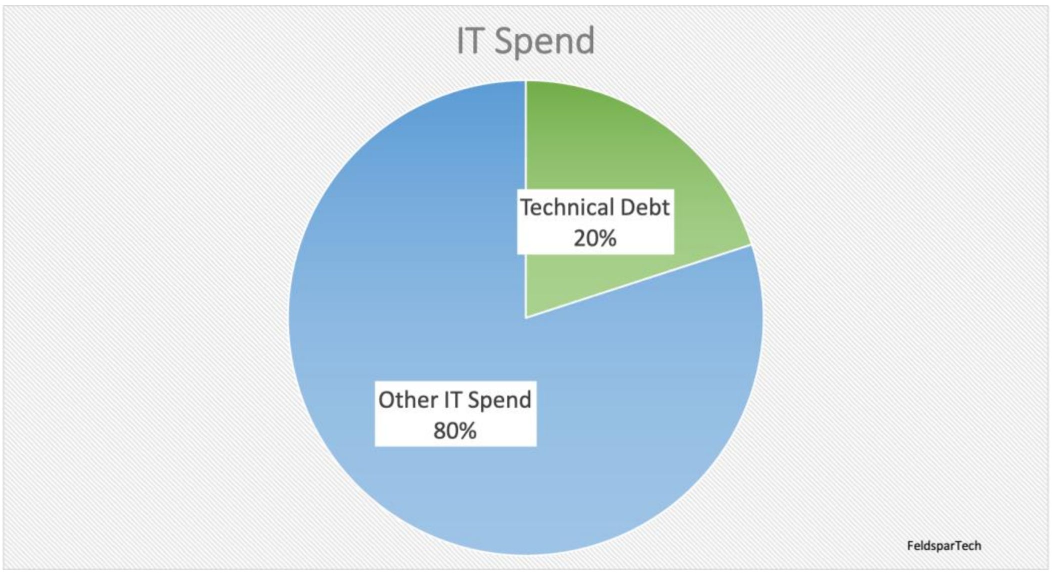
3. The costs/consequences of unhandled Technical debt get worse and become unmanageable over time.

Cost of Managing Technical Debt

How much money is being spent today to manage Technical Debt?

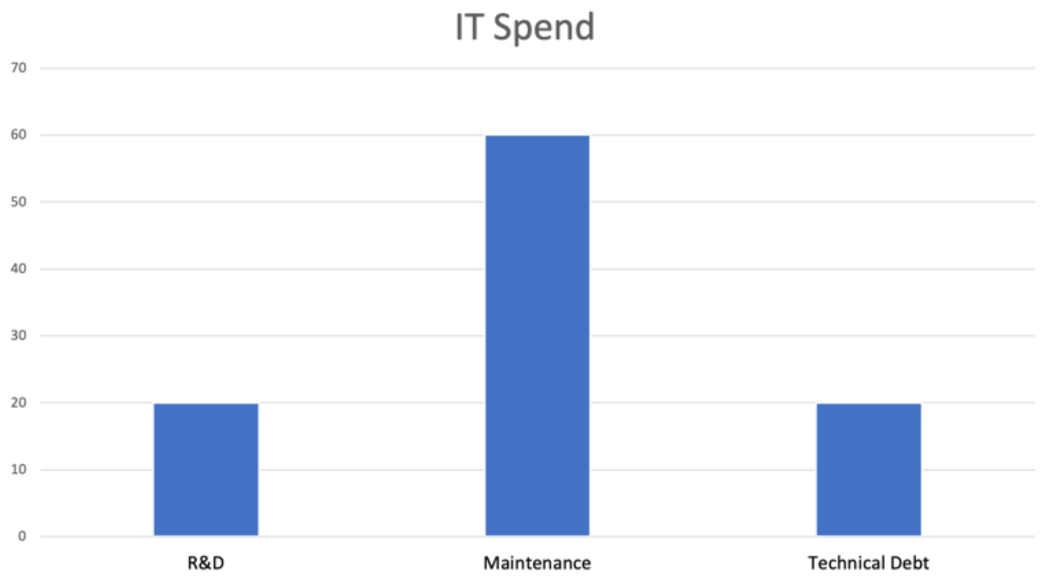
Annual Spend:

Organizations tend to spend 10-20% of their IT budgets on paying Technical Debt. This spend is something that could have been avoided or minimized. While it seems low, the scary part is that it comes straight out of the budget for new solutions, R&D and innovation.



How does it compare with spend on R&D/Innovation?:

In many organizations R&D and Technical Debt spend end up being almost equal, about 20% of the overall IT spend.



Is Technical Debt for Large Organizations Only?

Large organizations may have bigger debt due to complex systems and multiple changes. Because there are more players in big organizations the likelihood of Technical Debt is more in big organizations. Also churn of developers is another reason that contributes to more debt for bigger organizations.

Smaller organizations will have comparatively smaller debt. An important point to note is manageability. Even a comparatively smaller Technical Debt can be very difficult to handle for a small organization.

IT systems can be rendered unchangeable or unmanageable in both the cases if not attended to properly.

The same psychology is at play at both the places. There is interest and enthusiasm in building new systems. When systems are live, the goal is invariably to make-do with what exists. Seldom do refactoring projects result out of strategy. Organizations are compelled to take them when no options are available.

Can You Avoid Technical Debt?

Technical Debt will creep in advertently or inadvertently even with the best of the teams. If you run a business, you are bound to incur Technical Debt. So the attempt should not be to totally get rid of it but to minimize it and keep it within manageable limits. It should be treated like the cost of doing business.

Technical Debt cannot be avoided.

Minimizing Technical Debt requires planning and buy-in from the company leadership. Here are *best practices* to reducing Technical Debt:

1. *Do not overdo customization in packaged software:*

   A packaged software should be configured and not customized by writing more code. Upgrades to the packaged software can render custom functionalities incompatible. Any addition to functionalities and features should be done through the package software and not on top of it.
2. *Check attrition/turnover:*

   Any member leaving the team creates a knowledge deficit in the system. New members take time to learn and may not achieve the level of competence of the original team.
3. *Design for Decoupling:*

   Services and components that are tightly connected are difficult to change. Changing one can adversely impact the others. Decoupled or loosely coupled components allow changes without impacting the smooth functioning of the system.
4. *Design nimble systems that evolve and adapt to changing business needs*:

   A system should be able to undergo infinite changes and upgrades without impacting its performance and stability. With time a system should become more and more stable irrespective of the changes it undergoes. It should be easy to retire components and add new ones as per changing business needs.
5. *Limit people dependency:*

Put processes in place so that people do not become the bottleneck. The system should be so easy to understand and operate so that there is no need for an expert to run the show.

6. *Limit programming languages and tools used in your organization:*

   Different languages and tools will require different expertise. Ensuring availability of experts can become daunting.

7. *Standardize the way design and architecting is done*:

   Incorporate your best design and implementation practices in templates. Any new implementation should start with the templates. This will help achieve standardization and uniformity across your organization.

8. *Modernize legacy systems:*

   Systems that have been in use for long seem indispensable as they end up being the core of business. But the cost of running them can be huge. Such systems should be modernized phase wise if possible. Old systems that are past their shelf life should be retired and replaced with new modern implementations.

9. *Budget for Refactoring:*

   Systems that have been implemented recently should also be looked at for performance and efficiency improvements. It is a good practice to revisit and clean up the code base for poorly written or duplicate code.

Planning well to mitigate high Technical Debt is a good approach.

How to manage Technical Debt

The first step to managing Technical Debt is to accept that it exists in your organization and needs to be repaid every year. As long as code is written, there will be inefficiencies that creep in however diligent the IT team might be. Technical Debt is akin to the cost of doing business.

Once the existence of Technical Debt is accepted, the next step is to ensure the awareness of it. The leadership needs to understand it and track it just like the financial debt. Every organization should know its Technical Debt. It should make sincere attempts to know the sources contributing to the Technical Debt. The leadership should measure and track Technical Debt for determining the company's performance.

An organization should actively plan for 'repaying' Technical Debt every year.

Avoiding High Technical Debt

Use approaches that introduce lower Technical Debt.

1. Use pre-tested, prebuilt components as much as possible
2. Design loosely coupled systems using microservices
3. Use modern techniques for design and architecture
4. Build a nimble, changeable system for changes are the only constant.
5. Simple intuitive code and flows that are easy to handover to new team members
6. Adopt newer, faster and easy to use technology and tools
7. Allow for time and effort for the team to do proper work
8. Emphasize on design upfront
9. Try to build a culture of avoiding short-cuts for quick gains

Planning to Manage Technical Debt

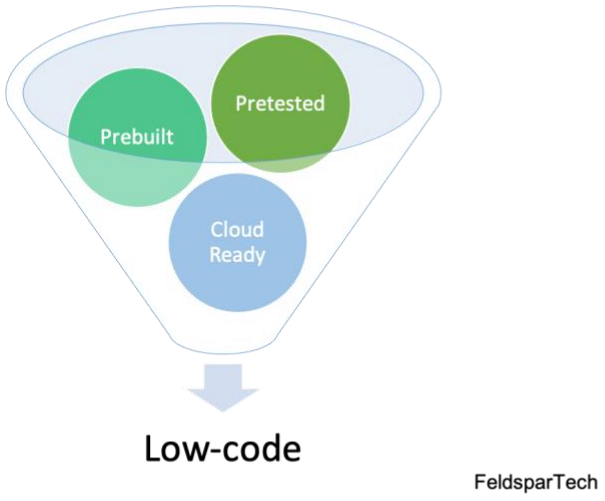
1. Organizations should plan for Technical Debt in similar ways to financial debt as both have the dubious potential to derail the business of a company

2. The leadership/management should plan to Define, Identify, Measure and Address Technical Debt
3. Every year an adequate budget needs to be allocated to attend to the 'Interest' of Technical Debt i.e. to perform projects for refactoring, updates or changes

A New Approach to Minimizing Technical Debt

We now know about Technical Debt and its consequences. You may wonder if there is a guaranteed way to keep the Technical Debt in check. Is there a methodology or a solution that encapsulates all the best practices and approaches mentioned in the previous sections?

There is help in the form of Low-code Technology that promises to help build solutions ***fast* and *right***. Low-code Technology offers platform for application development using pre-built components and services. There are templates are implemented using the best practices of design and architecture. These are pre-tested components that you can put to use to build your own software solution.

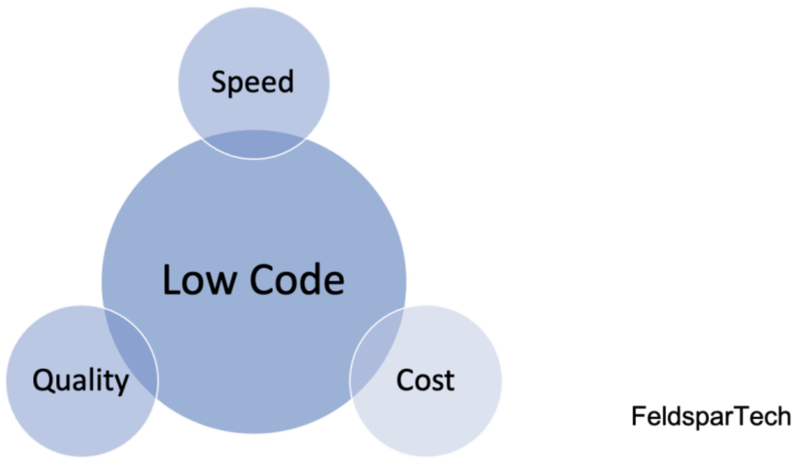


Due to pre-built components the pressure of time is almost negligible with Low-code. Changes are easy to make so there is no need for a short-cut most of the times.

Below is a list of benefits of Low-code Technology:

1. Building solutions is fast and simple with pre-built, pre-tested components.
2. Making changes is not time consuming. You can write and rewrite the application any number of times without taking too much time.
3. Testing and validations are integrated and are performed automatically as you develop the application.
4. Helps you standardize and implement best practices through guardrails in the platform.

5. Reduces dependency on coders. Reduced coding hence reduced defect injection.
6. Microservices-based decoupled systems
7. Encapsulation of standard requirements - like Security, encryption, SSO, Access control. So that you need to focus on the core functionality.
8. Low-code solutions are able to harness improvements in the underlying programming languages & libraries without any cost to the product owner or maintainer.



Please note that even if you use Low-code, you will still accrue Technical Debt. But it will be at a reduced scale and in manageable proportions. The interesting part is that Low-code will reduce the time and effort required to attend even to Technical Debt thus encouraging you to ***'fix in time'*** rather than postpone and ignore the accumulating debt.

The demand for software will only increase. Business insistence on faster go-to-market will always create pressure on the coding teams. The best way is to limit debt at the start and then plan to 'repay' it from time to time. Adopting techniques and technologies that help you achieve this will make your IT organization efficient.

If you would like to discuss more on this topic please reach us at ***info@feldspartech.com***

References

https://martinfowler.com/bliki/TechnicalDebt.html
https://en.wikipedia.org/wiki/Technical_debt
https://www.pluralsight.com/blog/software-development/erasing-tech-debt
https://www.cio.com/article/3410878/what-cios-need-to-know-about-low-code-software-development.html
https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/tech-debt-reclaiming-tech-equity