*Software Engineering*

# Lecture 10

# Technical Debt (Part-2)

## Growth and competition in software market

- Main references of this topic: [1, 2]
- Software industry is a growing market
- High competition in new software-intensive products, services and systems
- To keep the end-users satisfied, software companies are required to improve their products and features constantly
- **Time to market: Lots of pressure to companies** to deliver high-quality products in a time frame which is faster than their competitors (
- To address these challenges, software companies are constantly searching for improvements to their software development processes to deliver high-quality products and features in lesser time with the same resources and time available [1,2]
- The highest quality product does not always necessarily acquire the most success

# Earlier release and fixing issues

- Main references of this topic: [1, 2]

- To ensure earlier release to market, the software company can make the strategic decision to release the product e.g. with lesser overall quality, functionality, or performance to make the product available to the end-users faster [1,2]

- Since the software development today has become more flexible, it is possible to release new versions and patches on a daily basis [4].

- It is also possible for software companies to fix technical issues easier after the initial release, which has made earlier release cycles a popular strategy in software development [5].

# Earlier release and fixing issues

- Main references of this topic: [1, 2, 3, 4,5]
- The benefit of fast software development and earlier release done with workarounds and shortcuts can be seen in the time-to-market [1-5]
- creating shortcuts can speed up the software development and ensure that the product and features are released under the planned schedule, or even earlier [1-5]
-

# Earlier release and Technical debt

- Main references of this topic: [1, 2, 3, 4,5]
- Earlier releases gives non-optimal solution to market
- Non-optimal solution is a quick implementation that is easier to take in use compared to optimal solution that takes longer time to implement [1-5].
- 
- However, the drawback of leaving shortcuts and workarounds to software can be omitted quality [1-5]
- shortcut is a goal-driven adaptation and improvisation that aims to minimize negative consequences like anomalies or structural changes [1-5]
- Increase in the number of non-optimal solutions in the source code, can create complexity, **which can be seen as a *technical debt*** in the software that has to be paid for eventually [1-5].
-

## Technical debt Types

- Main references of this topic: [1, 2, 3, 4,5]
- **Naive technical debt: Debt was generated due to** team member or business immaturity or process deficiencies that lead to sloppy design, poor engineering practices, and a lack of testing

# Technical debt Types

- Main references of this topic: [1, 2, 3, 4,5]
- **Unavoidable technical debt:** design and implementation decisions were made, had problem and might need to change. This decision affected some areas and require changes in delivered product, these are unavoidable technical debt.

- **Strategic technical debt:** an organization might deliberately make a strategic decision to take shortcuts during product development to achieve an important short-term goal, such as getting a time-sensitive product into the marketplace

## Technical debt Types

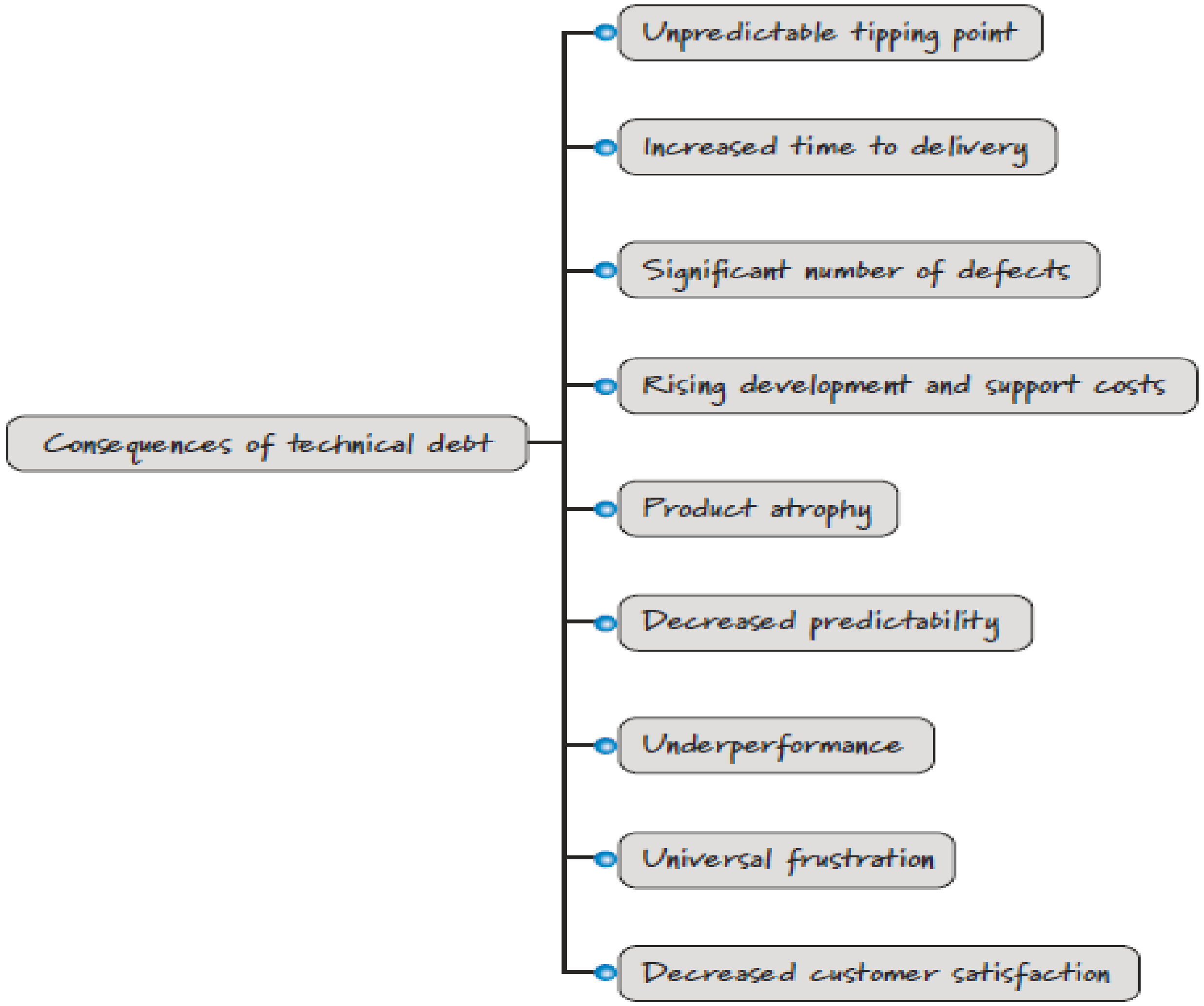
- Main references of this topic: [1, 2, 3, 4,5]
- Code Debt: Poorly written code that can be found as a problem in the source code,
- Design Debt : Violations and shortcuts in the principles of good design that under- focus on qualities such as maintainability and adaptability.
- 
- Architecture Debt:  Sub-optimal solutions and bad decisions in software architecture, which compromise internal quality aspects, such as maintainability, performance, robustness, modularity etc. In addition, architecture debt can be a solution that becomes sub-optimal when technologies and patterns come superseded. Architectural type of debt requires extensive development activities to fix.
- 
- Environmental  Debt  : Debt that happens in the environment of the application, which can include development, hardware, infrastructure, and supporting- related challenges; for example, manual processes that could be automated to increase productivity, or postponement of upgrades to infrastructures and components.
- 
- Knowledge Distribution Debt: The knowledge of software with long development history and millions of lines of code developed by original team members can suddenly change if the team is changed. This has a sudden effect, because the original knowledge is not transferred to new team members.
- 
- Documentation Debt : Missing, inadequate, insufficient, incomplete, or outdated documentation found in any aspect of software development.
-

# Technical debt Types

- Main references of this topic: [1, 2, 3, 4,5]
- 
- Test Debt : Shortcuts in testing such as lack of unit tests, integration tests, acceptance tests, written test scripts, code coverage, or the number of test cases being completed.
- 
- Requirements Debt The distance between optimal requirements specification and actual system implementation, for example, requirements that are only partially implemented.
- 
- Build Debt : Flaws in a software system built process, which makes it overly complex, time consuming, unnecessarily slow, and difficult.
- 
- Infrastructure Debt : Sub-optimal configuration of development-related processes, activities, technologies, supporting tools etc. Delays upgrades and infrastructure fixes, which have an effect on team productivity.
- 
- People Debt : Challenges with people in the software organization, which can have an effect on productivity cause delays in e.g. development activities.
- Usability Debt : Inappropriate usability decisions and standards that need to be changed in the future, for example, inconsistence in the navigational aspects

- Main references of this topic: [1, 2, 3, 4,5]

## Consequence of Technical debt

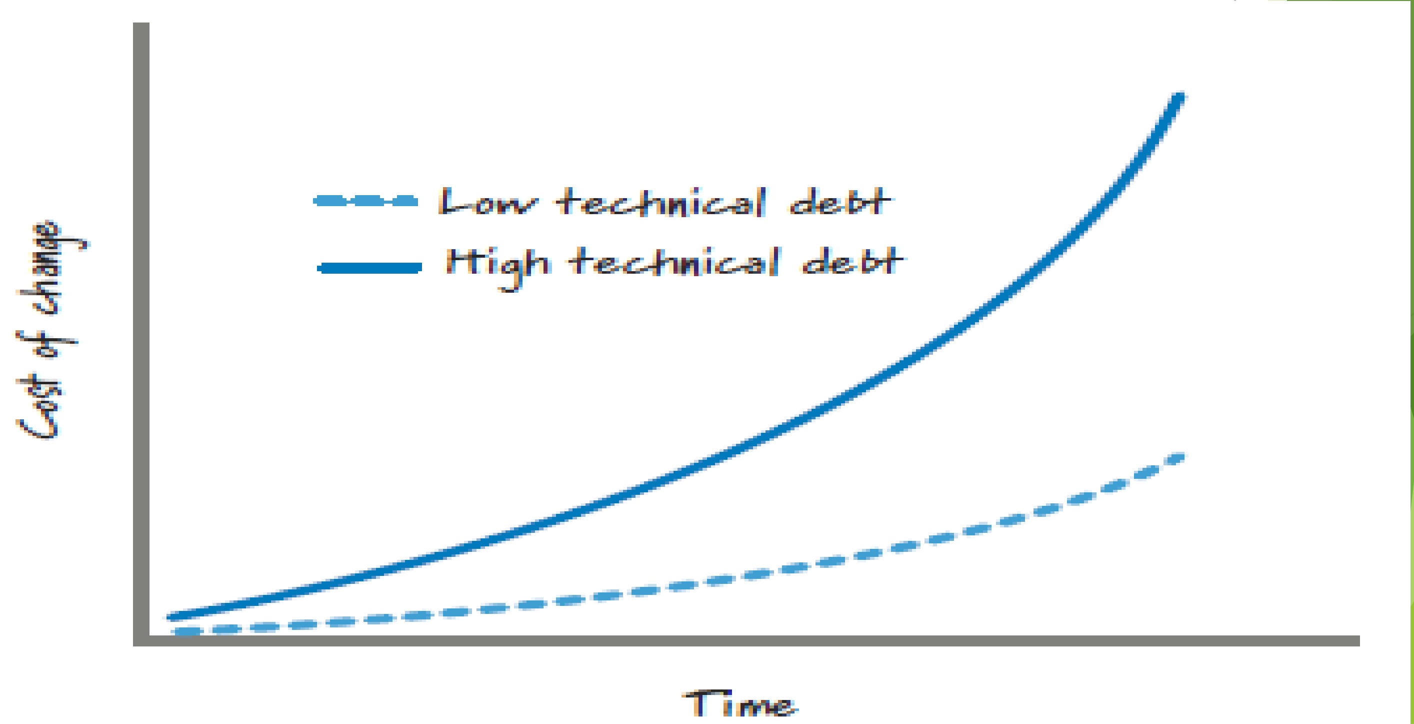

Consequences of technical debt
Unpredictable tipping point
Increased time to delivery
Significant number of defects
Rising development and support costs
Product atrophy
Decreased predictability
Underperformance
Universal frustration
Decreased customer satisfaction

## Consequence of Technical debt

- Main references of this topic: [1, 2, 3, 4,5]
- **Unpredictable Tipping Point:**
  - Each bit of technical debt, when added to the pool of existing technical debt, might do significantly more harm than the size of that new debt might imply.
  - At some point, technical debt achieves a sort of “critical mass,” where the product reaches a tipping point and becomes unmanageable or chaotic.
- **Increased Time to Delivery:** The greater the debt today, the slower the velocity tomorrow
- **Significant Number of Defects:**
  - Products with significant technical debt become more complex,
  - making it harder to do things correctly.
  - The compounding defects can cause critical product failures to happen with alarming frequency

## Consequence of Technical debt

- Main references of this topic: [1, 2, 3, 4,5]
- frequency
- **Rising Development and Support Costs**
  - What used to be simple and cheap to do is now complicated and expensive.
  - In the presence of increasing levels of technical debt, even small changes become very expensive

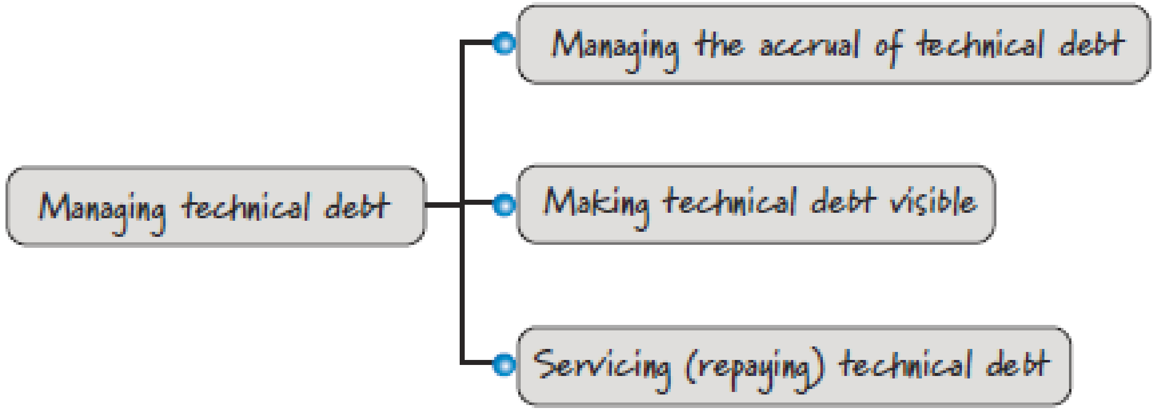


## Consequence of Technical debt

- Main references of this topic: [1, 2, 3, 4,5]
- **Decreased Predictability:**
  - estimates become bad estimates even for the most experienced team members.
  - There is simply too much uncertainty surrounding how long something might take when dealing with a debt-ridden product
  - making any sort of prediction is nearly impossible
- **Universal Frustration**
  - Technical debt doesn't suck the joy out of just technical people; it has the same effect on business people.
  - How long do we want to keep making business commitments that can't be met?
  - And what about our poor customers, who are trying to run their businesses on top of our debt-ridden product?
- **Decreased Customer Satisfaction**
  - So the extent of the damage caused by technical debt is not just isolated to the development team
  - or even to the development organization as a whole.
  - Even worse, the consequences of technical debt can substantially affect our customers and their perception of us

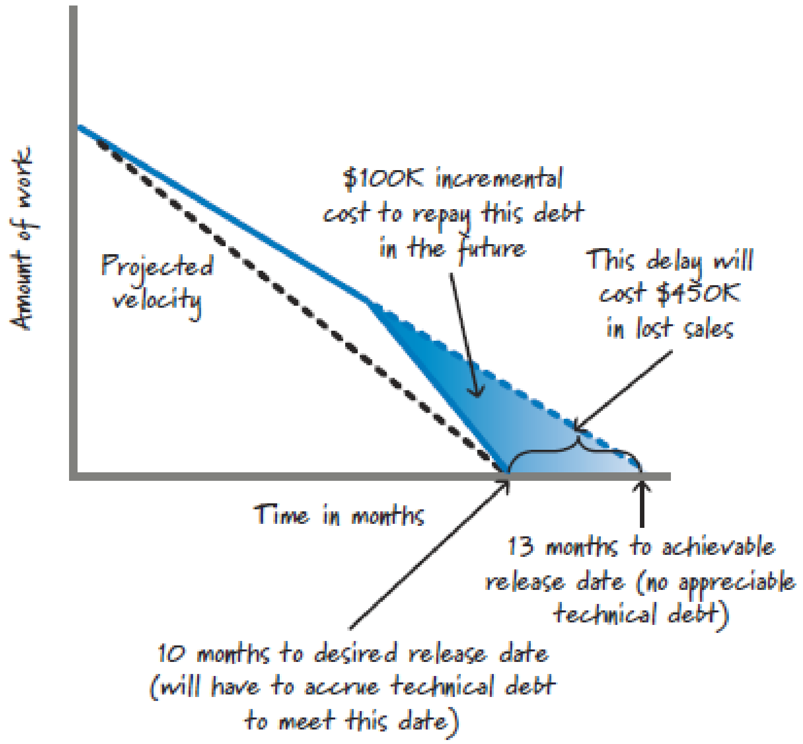
## Managing Technical debt

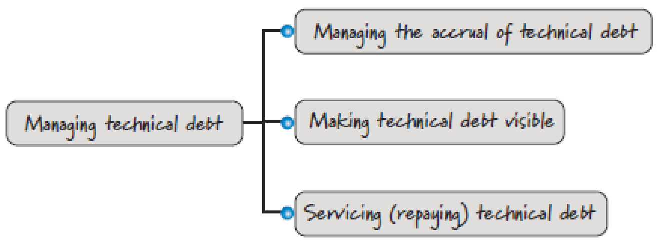
- Main references of this topic: [1, 2, 3, 4,5]
- Technical debt, like financial debt, has to be managed
- Technical debt management requires a balanced technical and business discussion that must involve technical and business people
- There are three principal technical debt management activities


Managing technical debt
Managing the accrual of technical debt
Making technical debt visible
Servicing (repaying) technical debt

# Managing Technical debt

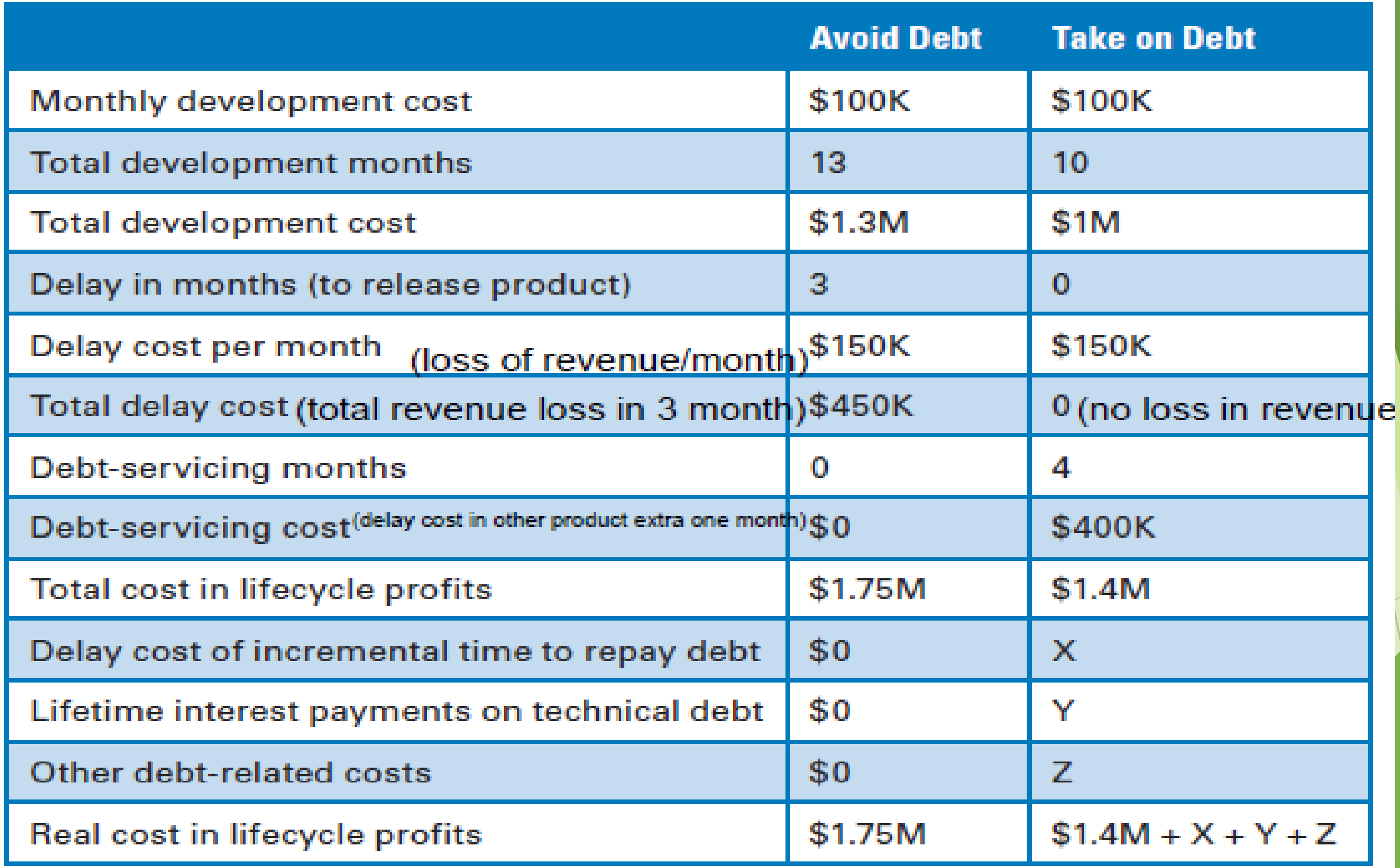
- Main references of this topic: [1, 2, 3, 4,5]
- **Managing the Accrual of Technical Debt**
  - **Use Good Technical Practices (**simple design, test-driven development, **continuous integration**, automated testing,**)**
  - **Properly Understand Technical Debt Economics**




Managing technical debt
Managing the accrual of technical debt
Making technical debt visible
Servicing (repaying) technical debt

## Managing Technical debt

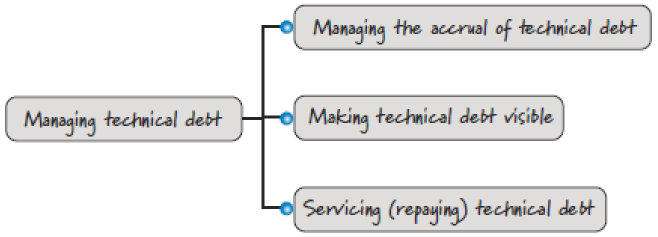
- Main references of this topic: [1, 2, 3, 4,5]
- **Properly Understand Technical Debt Economics**



| | Avoid Debt | Take on Debt |
|---|---|---|
| Monthly development cost | $100K | $100K |
| Total development months | 13 | 10 |
| Total development cost | $1.3M | $1M |
| Delay in months (to release product) | 3 | 0 |
| Delay cost per month (loss of revenue/month) | $150K | $150K |
| Total delay cost (total revenue loss in 3 month) | $450K | 0 (no loss in revenue |
| Debt-servicing months | 0 | 4 |
| Debt-servicing cost (delay cost in other product extra one month) | $0 | $400K |
| Total cost in lifecycle profits | $1.75M | $1.4M |
| Delay cost of incremental time to repay debt | $0 | X |
| Lifetime interest payments on technical debt | $0 | Y |
| Other debt-related costs | $0 | Z |
| Real cost in lifecycle profits | $1.75M | $1.4M + X + Y + Z |

## Managing Technical debt

- Main references of this topic: [1, 2, 3, 4,5]
- **Managing the Accrual of Technical Debt**
  - **Use Good Technical Practices (**simple design, test-driven development, **continuous integration**, automated testing,)
- **Make Technical Debt Visible**
  - it enables the development team and the business people to have a necessary conversation using a shared context.
- **Servicing the Technical Debt**
  - service or repay the debt
  - Happened-after delivery with technical debt
  - Every sprint, consider designating some amount of known technical debt as targeted technical debt to be serviced during the sprint.


Managing technical debt
Managing the accrual of technical debt
Making technical debt visible
Servicing (repaying) technical debt

# References

1. Jesse Yli-Huumo, THE ROLE OF TECHNICAL DEBT IN SOFTWARE, DEVELOPMENT, PhD Thesis, Lappeenranta University of Technology, Lappeenranta, Finland, 2017
2. Essential Scrum—Kenneth S. Rubin
3. The agile age-Managing projects effectively using agile scrum---Brian vanderjack
4. Neely, S., Stolt, S., 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy), in: Agile Conference (AGILE), 2013. Presented at the Agile Conference (AGILE), 2013, pp. 121–128. doi:10.1109/AGILE.2013.17
5. Greer, D., Ruhe, G., 2004. Software release planning: an evolutionary and iterative approach. Information and Software Technology 46, 243–253. doi:10.1016/j.infsof.2003.07.002