Technical Debt

CEN 4010 Intro to Software Engineering

Professor Alex Roque

What is Technical Debt

- Ward Cunningham (1992) described it…

> *"Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite…The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt. Entire engineering organizations can be brought to a stand-still under the debt load of an unconsolidated implementation…"*

What is Technical Debt

- In reality, it’s any to **shortcuts** teams **purposely take** and also to the **many bad things** that plague software, such as:
  - **Unfit (bad) design** – once okay but no longer due to business & technology changes
  - **Defects** – known problems not yet removed
  - **Insufficient test coverage** – more testing is needed but we don’t do it

What is Technical Debt

- In reality, it’s any to **shortcuts** teams **purposely take** and also to the **many bad things** that plague software, such as:
  - **Excessive manual testing** – rather than using automated testing
  - **Poor integration & release management** – not done in an economical way
  - **Lack of platform experience** – limited or loss of technical skills (COBOL, new tech.)

Technical Debt

- **Technical debt** metaphor resonates with **business people** who are familiar with **financial debt**; technical debt **payment** comes in form of additional development work

- So imagine, that you can a principal and interest.
  - If you pay the interest only (code a band aid, just to move forward) you wont really remove the problem.

- Essential, you need to invest the time in a proper code refactor (pay of the principal) to get rid of the technical debt.

Image sourced from
www.shutterstock.com

Technical Debt – Three Types

1. **Naïve** (aka…reckless, unintentional, mess)
   - **Causes** include
     - Immaturity of team members or the business
     - Development **Process** deficiencies that lead to
       - Sloppy design
       - Poor engineering practices
       - Lack of testing
   - **Cures** include
     - Proper training
     - Good understanding of how to apply technical practices
     - Sound business decision making

Technical Debt – Three Types

2. **Unavoidable**
    - These are usually **unpredictable** and **unpreventable**
        - Design work culminates in user-valuable features
        - Cannot **perfectly** predict up front how design & features need to evolve
        - End result is that change is needed thus technical debt
    - **Examples**
        - Home-grown software created based on current business needs
        - Purchased, Embedded, or API accessible software also based on current business needs

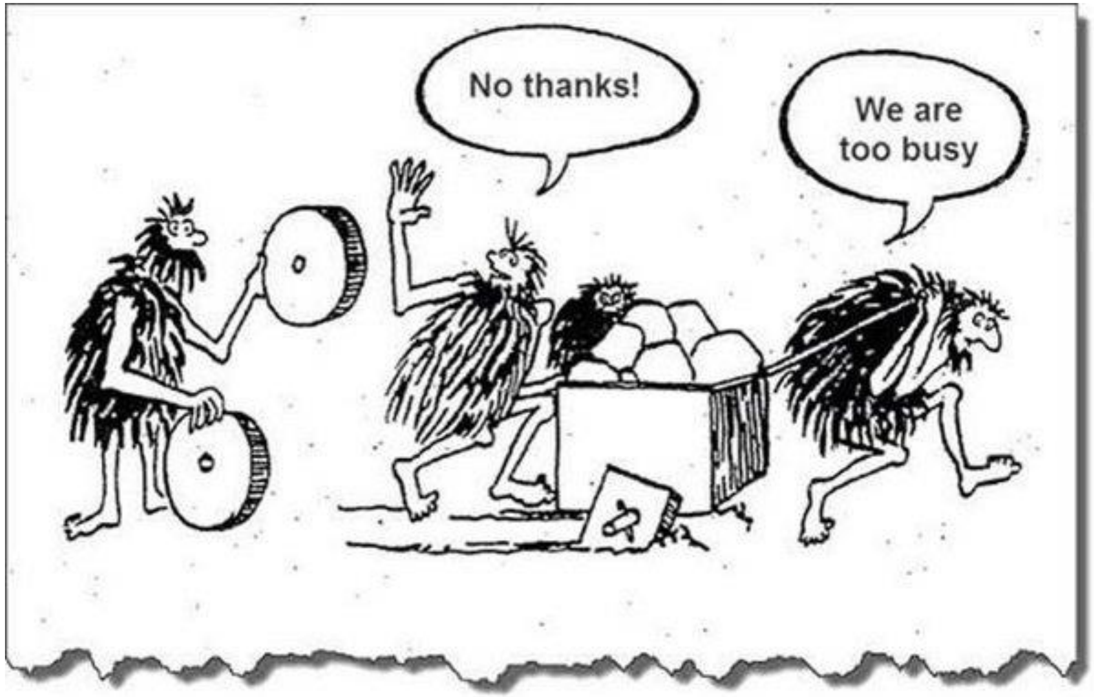
Technical Debt – Three Types

3. **Strategic**

   - Tool used to better leverage the **economics** of important, often time-sensitive, decisions
   - **Examples**
     - **Purposely make a strategic decision** to take shortcuts during product development to **achieve an important short-term goal**, such as being first to market, etc.
     - Capital-strapped organization taking shortcuts to deploy a product in order to **generate revenue**

But what happens if we don't pay off Technical Debt?

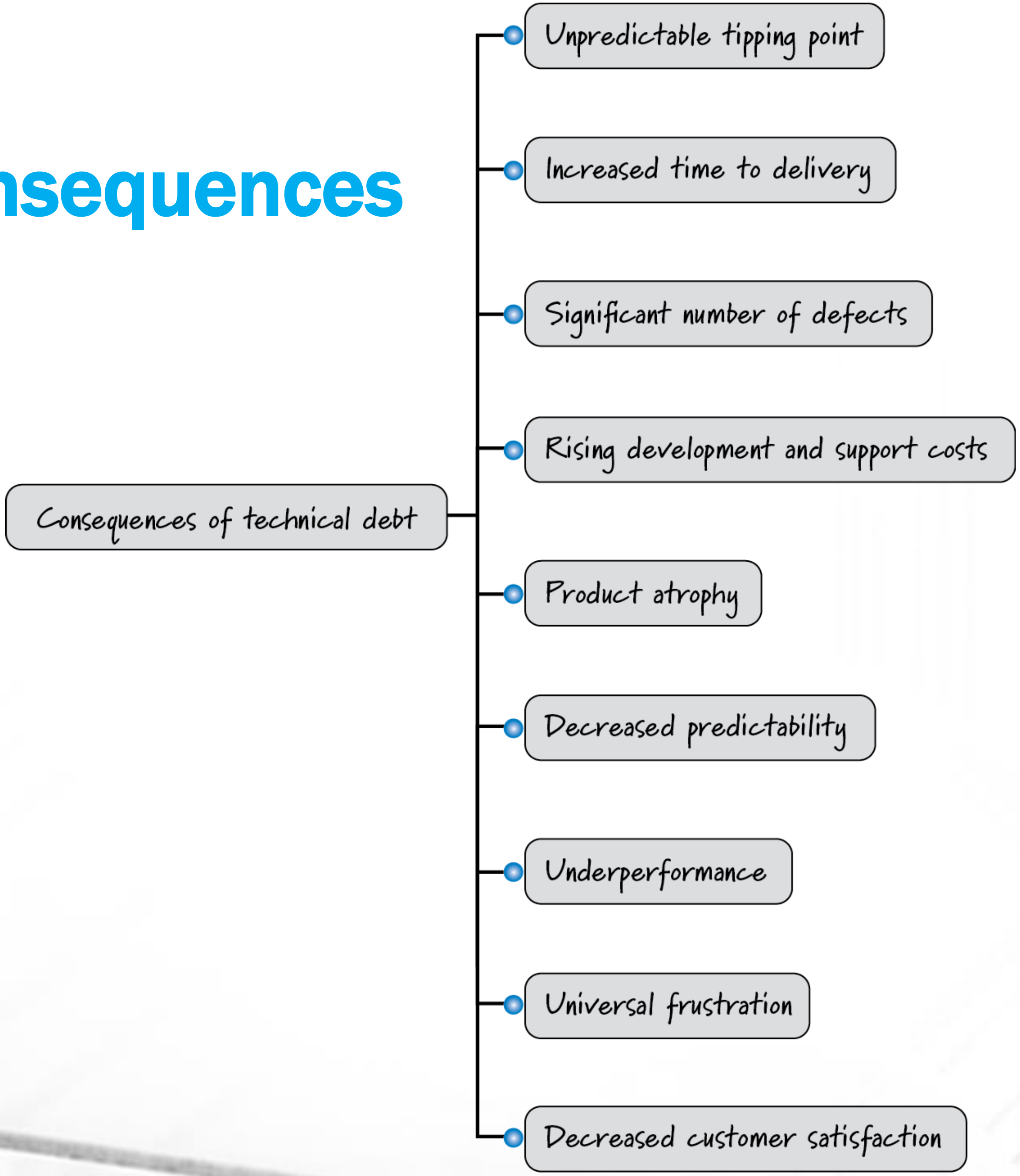
- Not allocating the time to pay off technical debt may end up in catastrophic results (system shutdown, hard to code new feature for)
- It can also blind us from innovating and finding a better, more efficient way to move forward.
- In general, as technical debt rises, so do the severity of consequences



Technical Debt Consequences

- Think about the consequences of carrying technical debt…..
- What do you think they are?
- How do you think it impacts the organization?

Consequences



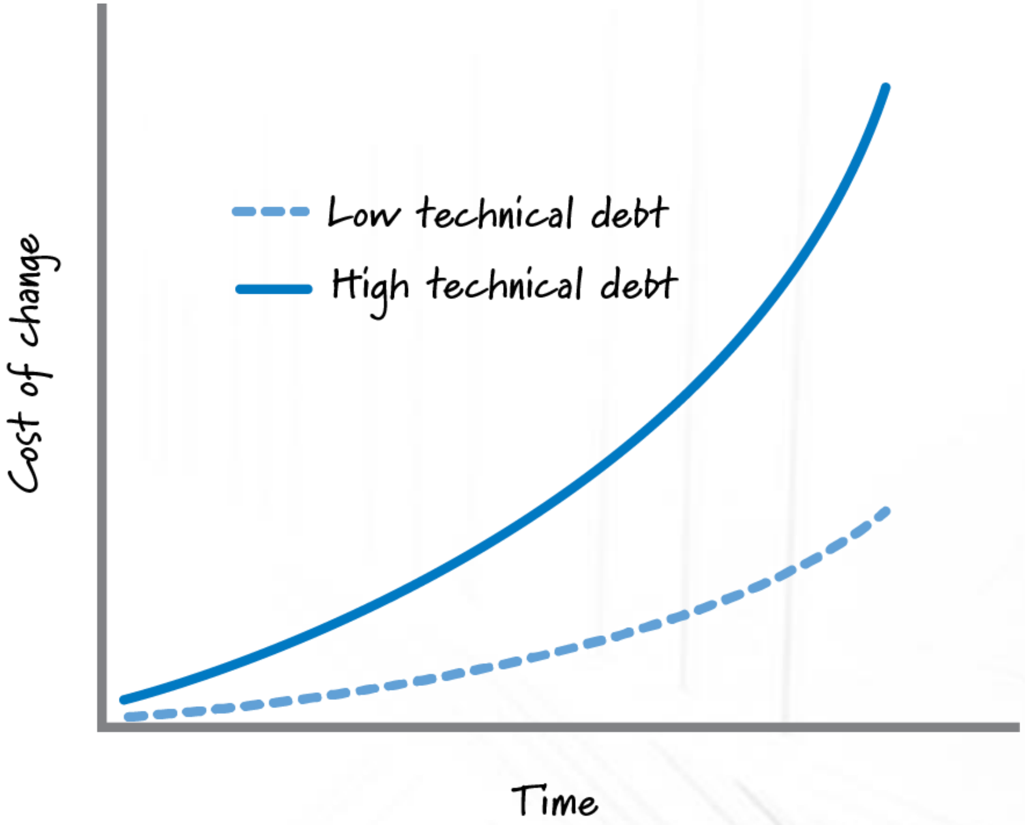
Consequences of technical debt

- 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

Consequences

- **Unpredictable Tipping Point:** Slowly piling on technical debt can lead to a tipping point where the product becomes unmanageable.

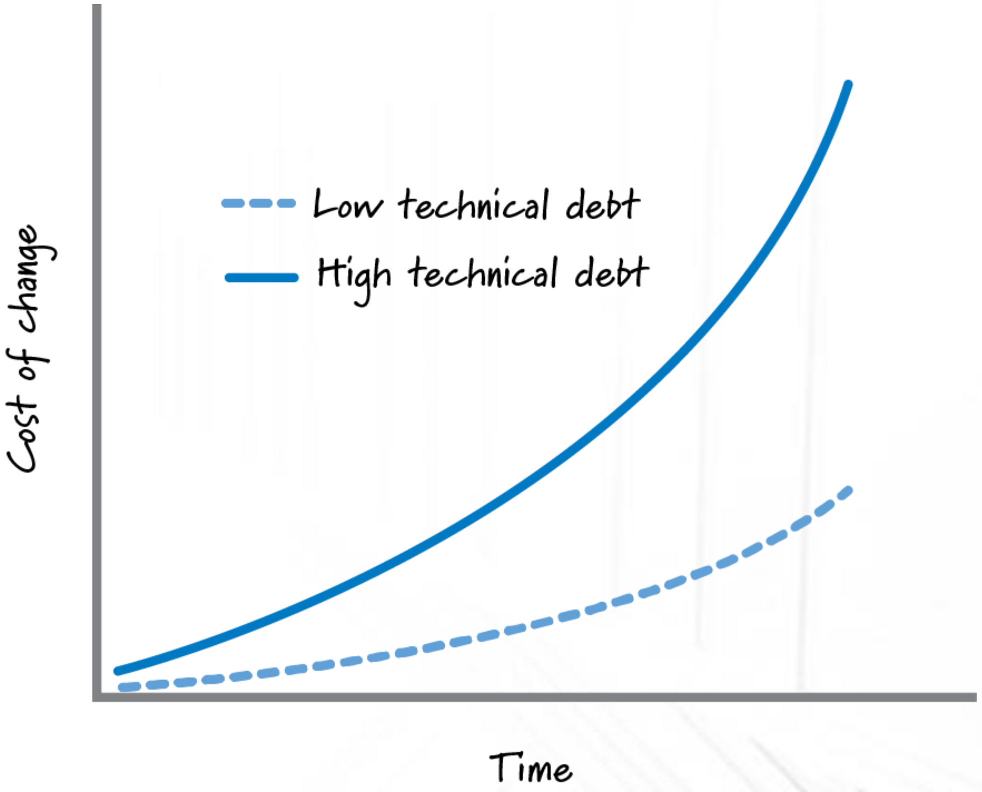
- **Increased Time to Delivery:** Usually, piling on technical debt means that as we repay back the time, we have less time to deliver new features and product fixes to customers.



Consequences

- **Significant Number of Defects:** Products with higher technical debt makes them more complex and can lead to more defects being inadvertently produced.

- **Rising Development and Support Costs:** With increasing technical debt, even small changes become expensive. Therefore the costs associated with development and support of the product start increasing.



Consequences

- **Product Atrophy:** Since we stop adding new features, we settle, and lack of new features becomes the norm.
- **Decreased Predictability:** Too much uncertainty due to the debt, predictability decreases.
- **Underperformance:** Reduced expectation, people get used to underperformance.

Consequences

- **Universal Frustration:** Trust in development team erodes and frustration takes its place.
- **Decreased Customer Satisfaction:** Since there are not new features and there is a lack of stability in the product, customers suffer.

Pay it off right here, right now?

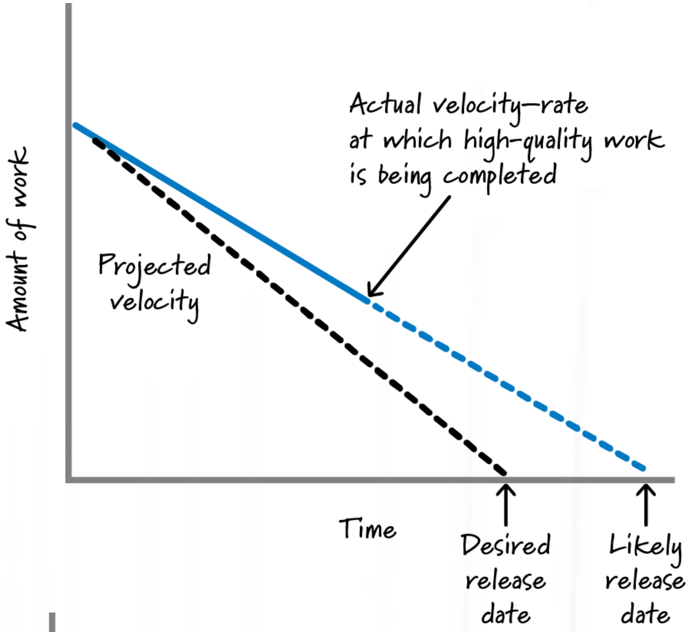
- Sometimes a developer will find a small piece of technical debt during a code review…should they fix it if they have the time?
- Normally, YES you would want to if the time was available but, always make sure:
  - Is there a way to test the refactored code after a change?
  - Is there a way to properly perform integration testing on the modified code?
  - Is there a way to properly perform regression testing on the product?
- If the answer to any of these questions is a “No” then you will probably cause more problems by paying of the technical debt now than in a planned manner.

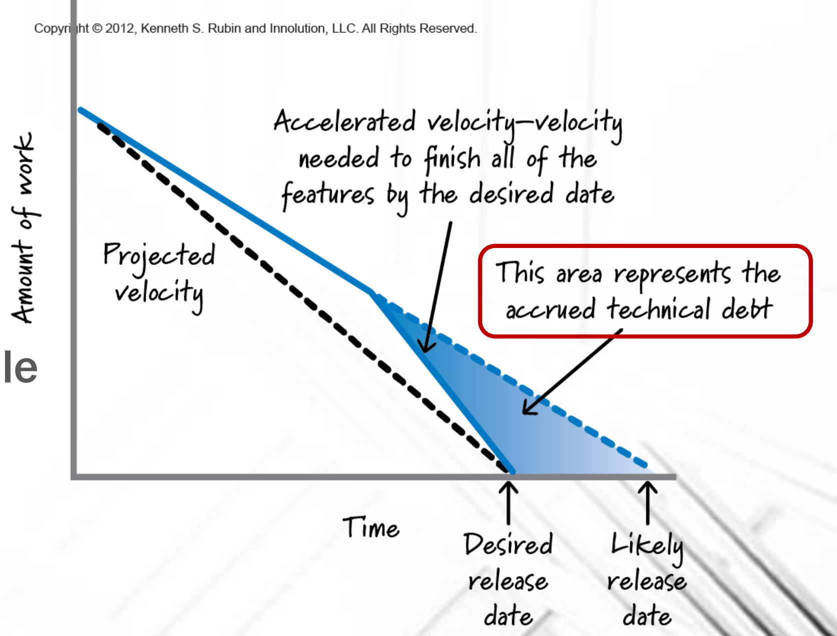
Causes of Technical Debt

- Three main forms of Technical Debt are…
  - Naïve, Unavoidable, Strategic

- Pressure to Meet Deadline
  - Naïve & Strategic driven by business pressure

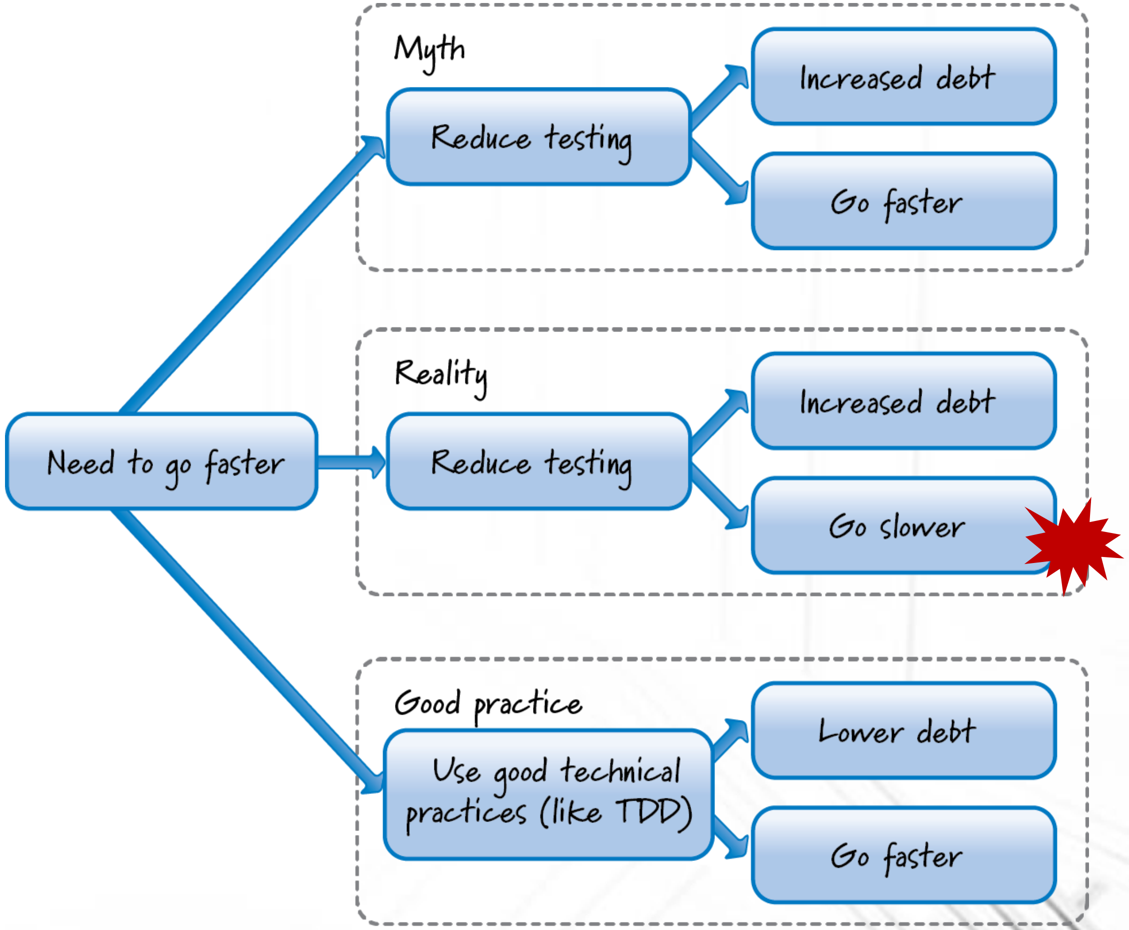
- Attempting to Falsely Accelerate Velocity
  - Cut scope or add more time?
  - Cutting corners to work faster to meet unreasonable fixed scope and date (tech. debt)





Causes of Technical Debt

- **Myth:** Less Testing Can Accelerate Velocity
  - Thinking is that we can progress faster when it will cause a slow down as we find defects much later in the process



Need to go faster

Myth

Reduce testing

Increased debt

Go faster

Reality

Reduce testing

Increased debt

Go slower

Good practice

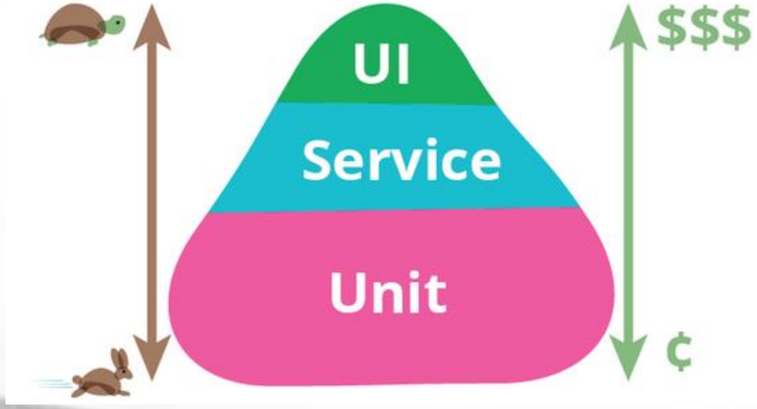
Use good technical practices (like TDD)

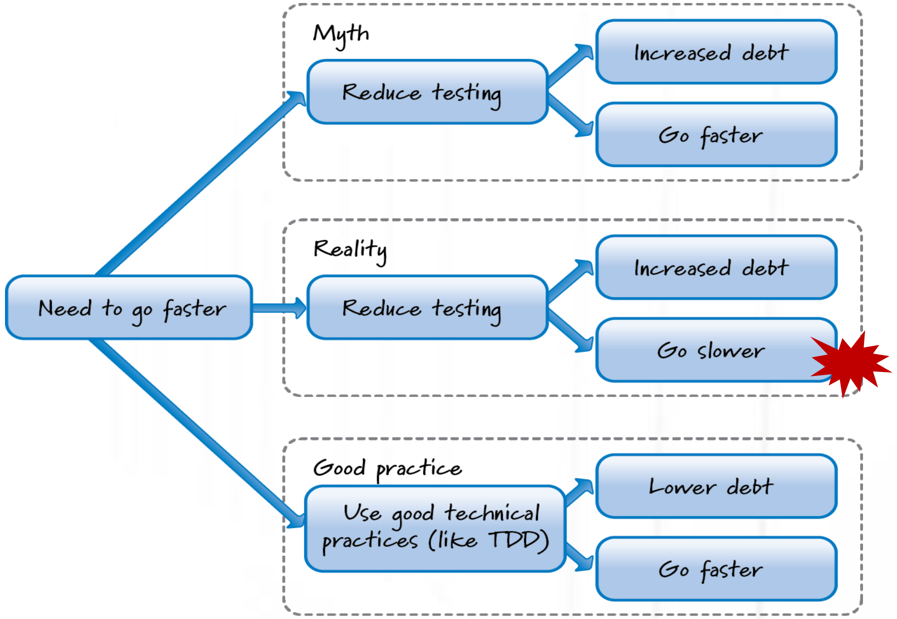
Lower debt

Go faster

Causes of Technical Debt

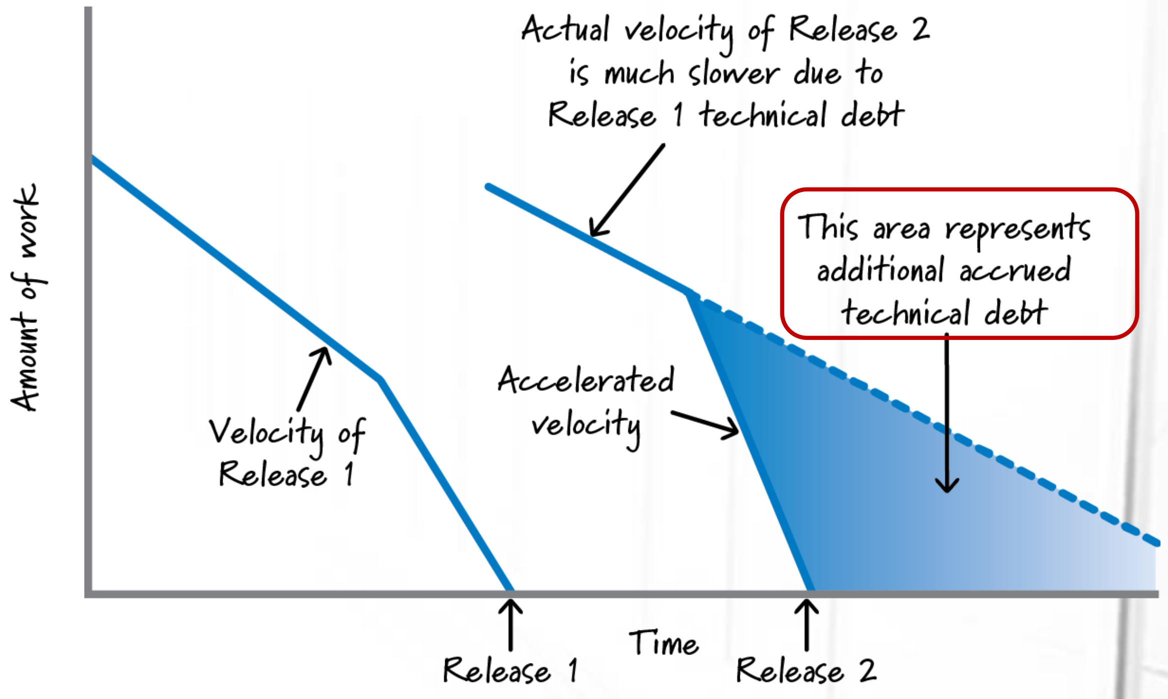
- **Myth:** Less Testing Can Accelerate Velocity
  - TDD = Test Driven Development: The Developer writes unit tests before writing the code that will make the test past.
  - Test Pyramid https://martinfowler.com/bliki/TestPyramid.html





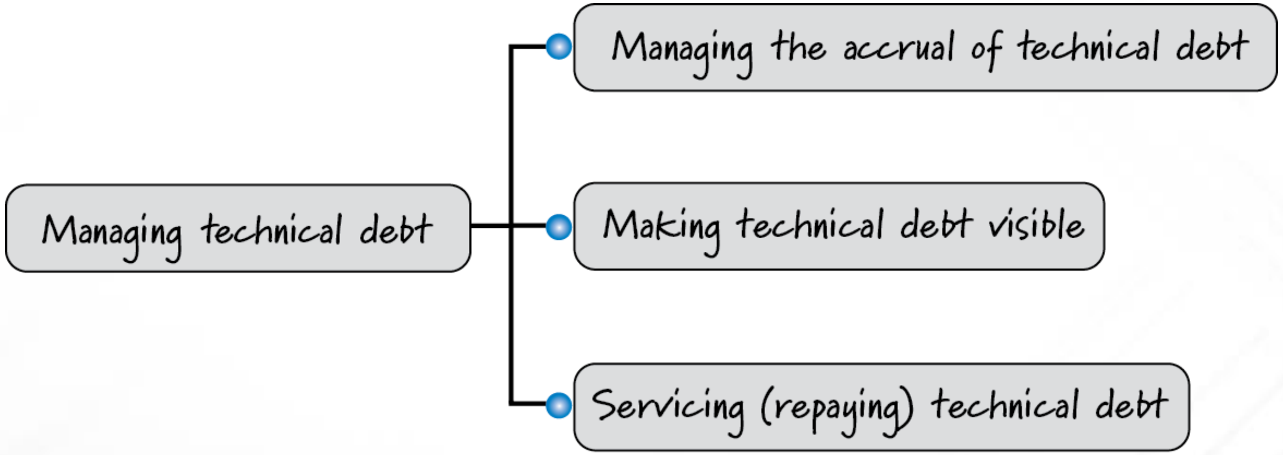
Causes of Technical Debt

- **Debt Builds on Debt**
  - Future Debt builds on top of Existing Debt causing economic harmful consequences



Managing Technical Debt

- High Technical Debt leads to **bad choices**:
  - Do nothing, problem gets worse
  - Make ever-larger investments to reduce the debt which consumes valuable resources
  - Declare technical bankruptcy, retire the technical debt, and replace the debt-ridden product with a new product full of its own costs and risk associated with its creation



Managing technical debt

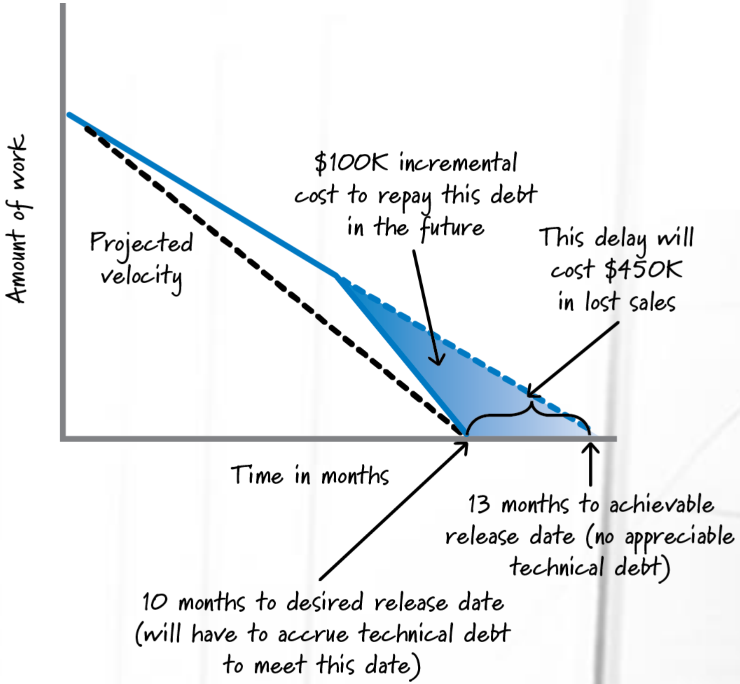
Managing the accrual of technical debt

Making technical debt visible

Servicing (repaying) technical debt

Managing Technical Debt

1. **Managing the Accrual of Technical Debt**
   - Can only accrue so much of it, like borrowing money!
   - Stop adding **Naïve** technical debt
   - Use good Technical Practices (Scrum Framework does not formally define)
     - Simple design,
     - test-driven development,
     - continuous integration,
     - automated testing,
     - refactoring, etc.
   - Use Strong Definition of Done
     - Issues not addressed by definition of done come back and "bite" us later
   - Properly Understand Technical Debt Economics
     - Sadly, many organizations simply do not understand implications of technical debt



Managing Technical Debt

- Another good way to try avoid Naïve technical debt is encouraging code reviews between developers.
- Code reviews help include quality and having a fresh pair of eyes can bring any potential future technical debt.
- Also, when possible defining proven architectural best practices for teams to follow can reduce naïve technical debt.
- Push for business stakeholders to see the value of code reviews and best practices!

Managing Technical Debt

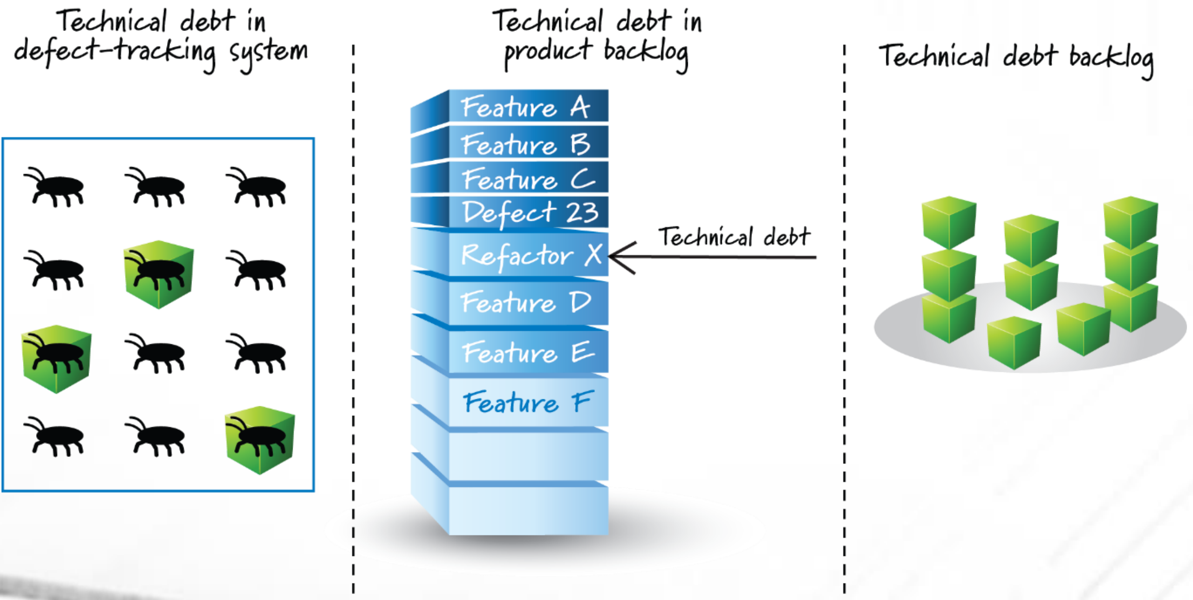
- Be an advocate for change. Make sure your leadership understands the danges in technical debt!

DID YOU FINISH THE SOFTWARE YET?
NO, I'M STILL PAYING OFF THE TECHNICAL DEBT FROM THE LAST PROGRAMMER YOU RUSHED.
I DON'T KNOW WHAT THAT MEANS.
WELL, THAT EXPLAINS A LOT.
Dilbert.com @ScottAdamsSays
1-3-17 © 2017 Scott Adams, Inc./Dist. by Andrews McMeel

Managing Technical Debt

2. **Making Technical Debt Visible**

- Using Technical Debt metaphor allows development team to have conversation with business people
- Make it visible at the Business Level (Balance Sheet) (Show it at the user story or % of velocity)
- Make it visible at the Technical Level



Copyright © 2012, Kenneth S. Rubin and Innolution, LLC. All Rights Reserved.

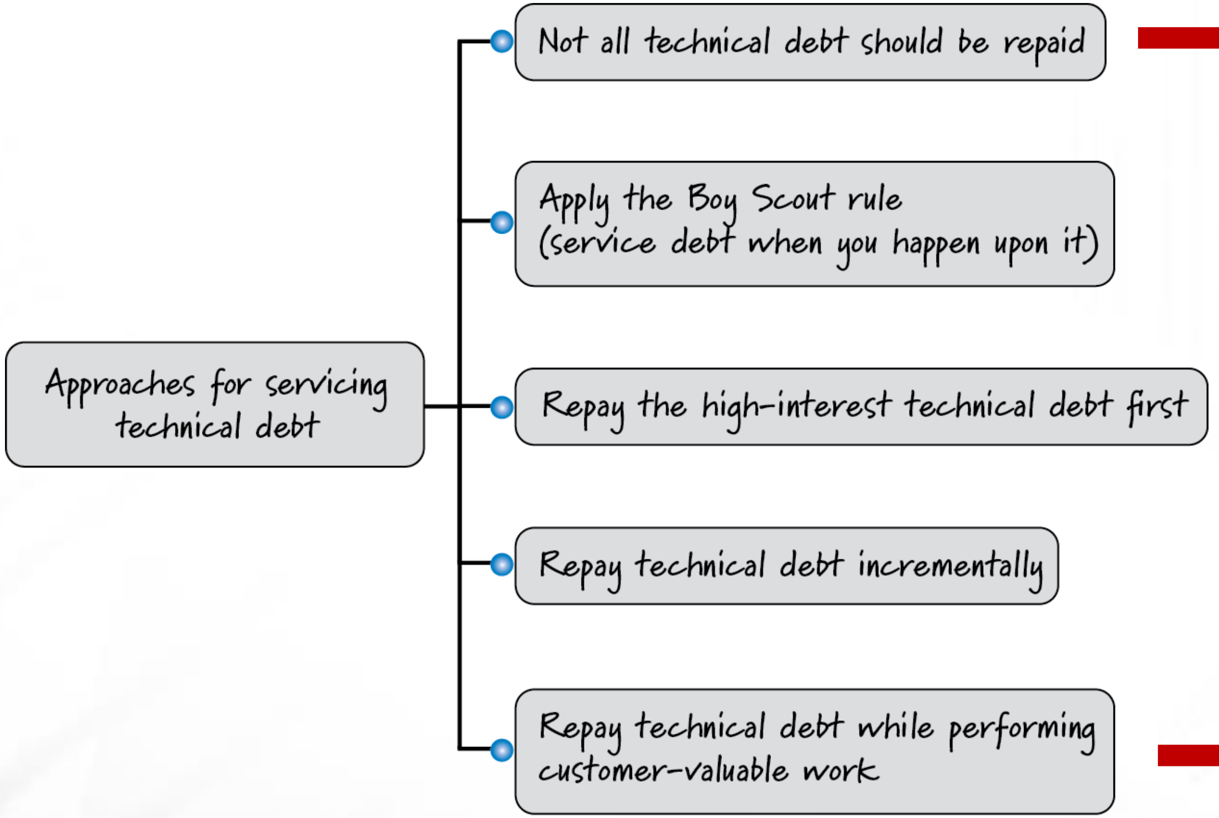
Managing Technical Debt

3. **Servicing (repaying) the Technical Debt**

- Some status categories
  - **Happened-upon** technical debt – unaware that it existed until it was exposed (e.g., a work-around had to be created)
  - **Known** technical debt – known to the development team
  - **Targeted** technical debt – known and targeted for servicing (repaying) by the development team
- **One approach to servicing the debt**
  - Determine **IF** it should be serviced
  - During work discover **Happened-upon** debt, clean it up, up to a reasonable threshold, then classify the remainder as **Known** debt
  - Designate some amount of time in each sprint to service **Known** debt making it **targeted**

Managing Technical Debt

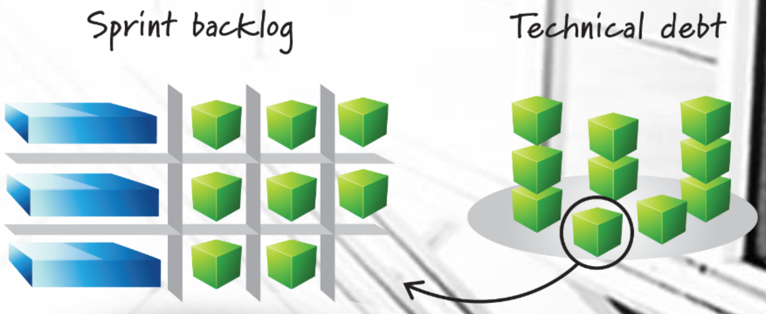
- 3. Servicing (repaying) the Technical Debt



Approaches for servicing technical debt

- Not all technical debt should be repaid
- Apply the Boy Scout rule (service debt when you happen upon it)
- Repay the high-interest technical debt first
- Repay technical debt incrementally
- Repay technical debt while performing customer-valuable work

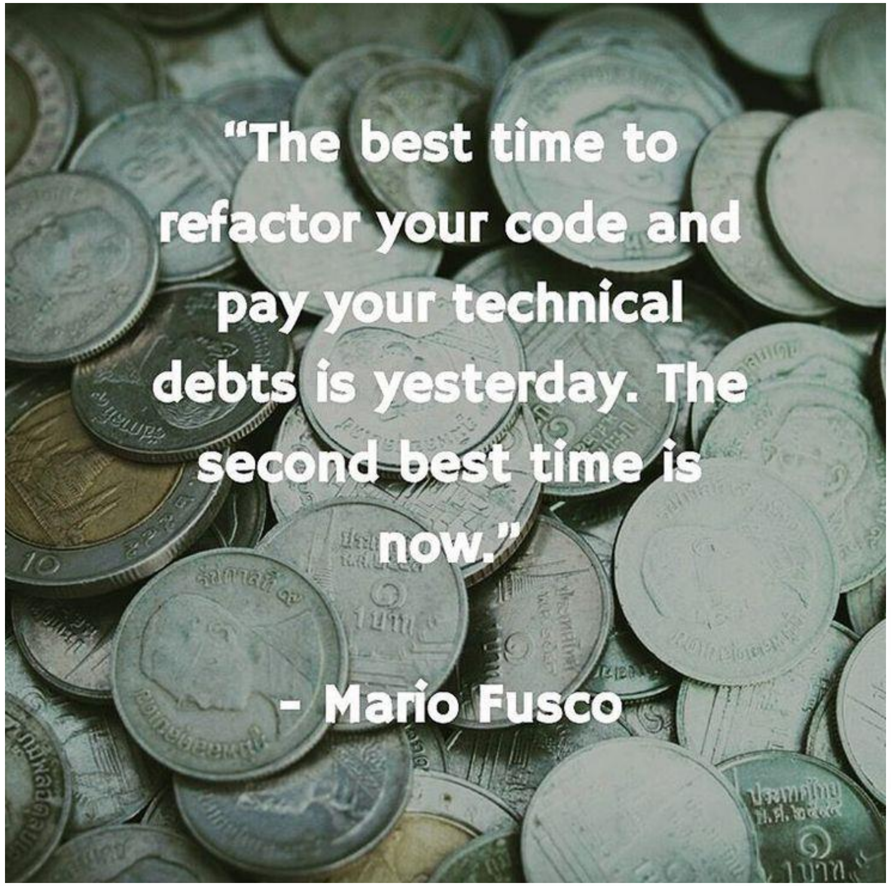
- Product Nearing End of Life
- Throwaway Prototype
- Product Built for a Short Life



Customers will notice!

- This is more frequent. Why?
- Customers are more technical savvy! Competitors are always trying to lure customers away.
- Legacy Products that are not refactored or replaced always at risk. Customers will say that:
  - The architecture is not flexible enough to meeting business needs
  - The UX is outdated
  - The data model is not flexible enough to support new ideas
  - The performance is rendering the product to be almost unusable

In short...make a plan!



"The best time to refactor your code and pay your technical debts is yesterday. The second best time is now."

- Mario Fusco