
# Technical Debt

17-313 Fall 2025

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

Sources:

- Managing Technical Debt. Ipek Ozkaya. CMU SEI

# Administrivia

- P4 is out

# Learning Goals

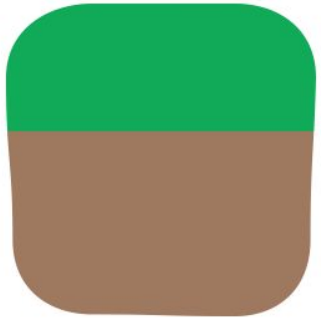
- Understand the concept of technical debt
- Reflect on personal experiences of technical debt
- Explain the importance of technical debt management
- Learn techniques for managing technical debt

# Technical Debt

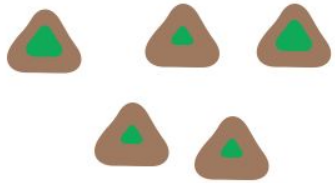
# Technical debt

*Any software system has a certain amount of **essential** complexity required to do its job...*

*... but most systems contain **cruft** that makes it harder to understand.*



*Cruft causes changes to take **more effort***

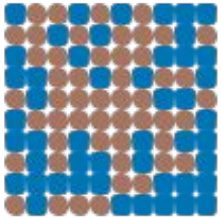


*The technical debt metaphor treats the cruft as a debt, whose interest payments are the extra effort these changes require.*

*https://martinfowler.com/bliki/TechnicalDebt.html*

# Internal quality makes it easier to add features

*If we compare one system with a lot of cruft...*



*the cruft means new features take longer to build*

*this extra time and effort is the cost of the cruft, paid with each new feature*

*...to an equivalent one without*

*free of cruft, features can be added more quickly*

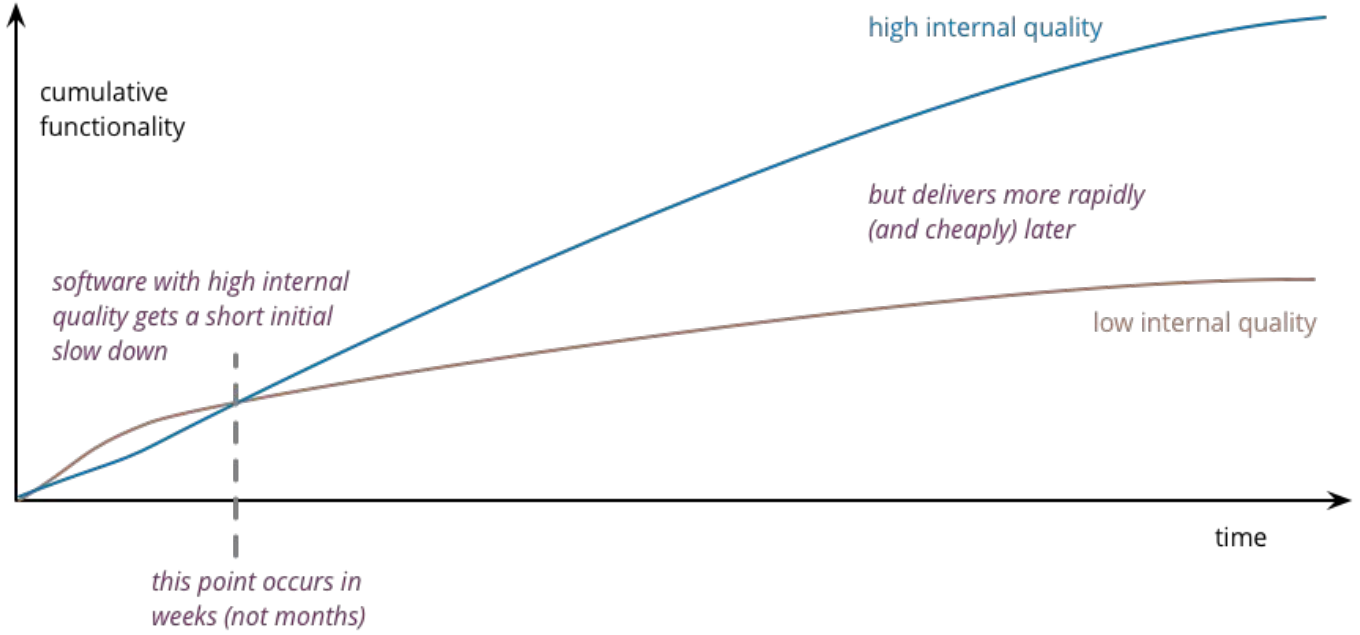
# Technical Debt != Bad Internal Quality

*"In software-intensive systems, technical debt consists of* ***design or implementation constructs*** *that are expedient in the* ***short term*** *but that set up a technical context that can make a* ***future change more costly or impossible.*** *"*

*"Technical debt is a contingent liability whose impact is l****imited to internal system qualities*** *– primarily, but not only,* ***maintainability*** *and* ***evolvability****."*

*Managing Technical Debt: Reducing Friction in Software Development.* Philippe Kruchten, Robert Nord, Ipek Ozkaya

# High internal quality is an investment



# What actions cause technical debt?

Tightly-coupled components

Poorly-specified requirements

Business pressure

Lack of process

Lack of documentation

Lack of automated testing

Lack of knowledge

Lack of ownership

Delayed refactoring

Multiple, long-lived development branches

...

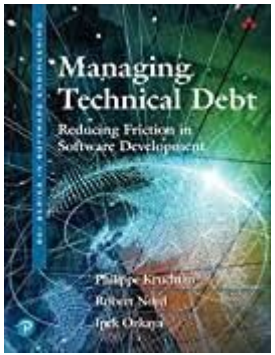
# Bitrot: Even if your software doesn't change, it will break over time

# EVERYONE CREATES TECHNICAL DEBT

# Bad: Too much technical debt

- Bad code can be demoralizing
- Conversations with the client become awkward
- Team infighting
- Turnover and attrition
- Development speed
- …

# How to manage technical debt?




*Managing Technical Debt: Reducing Friction in Software Development.*
Philippe Kruchten, Robert Nord, Ipek Ozkaya

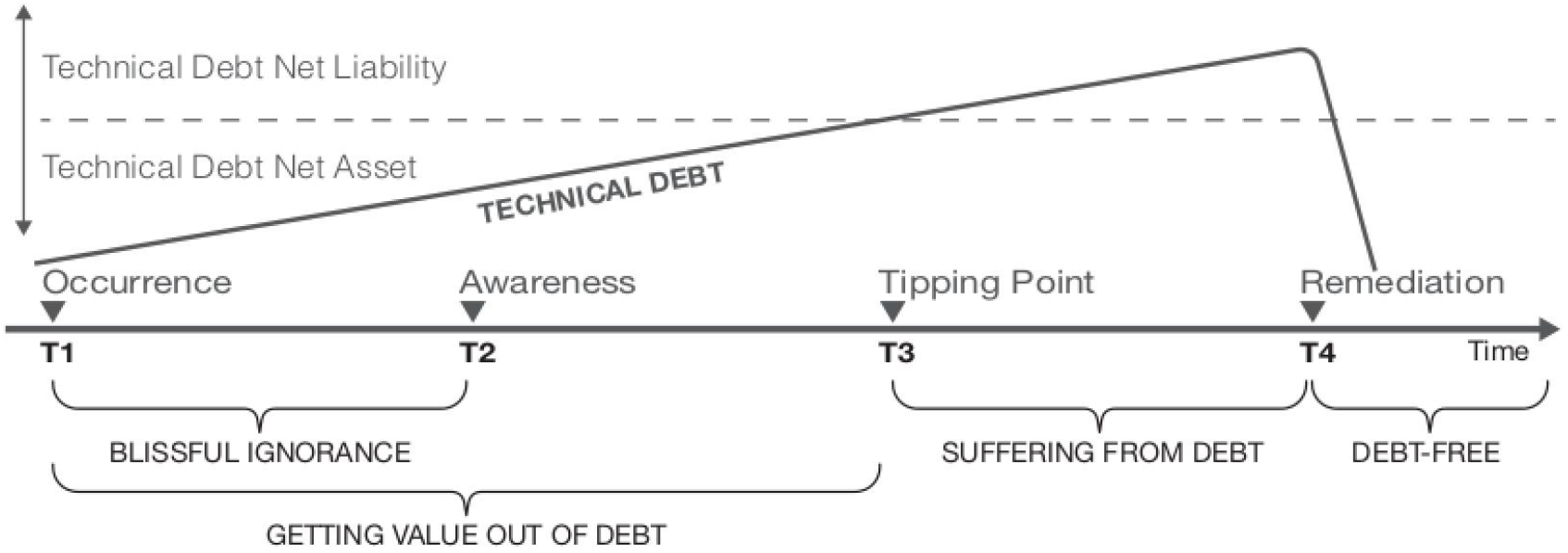
# Principles of Technical Debt Management

1. Technical debt is a useful rhetorical concept for dialogue.
2. If you do not incur any form of interest, then you probably do not have actual technical debt.
3. All systems have technical debt.
4. Technical debt must trace to the system.

# Principles of Technical Debt Management

5. Technical debt is not synonymous with bad quality.
6. Architecture technical debt has the highest cost of ownership.
7. All code matters!
8. Technical debt has no absolute measure.
9. Technical debt depends on the future evolution of the system.

# When should we reduce technical debt?


Technical Debt Net Liability
Technical Debt Net Asset
TECHNICAL DEBT
Occurrence
Awareness
Tipping Point
Remediation
T1
T2
T3
T4
Time
BLISSFUL IGNORANCE
SUFFERING FROM DEBT
DEBT-FREE
GETTING VALUE OUT OF DEBT

# Managing technical debt

Organizations needs to address the following challenges continuously:

1. Recognizing technical debt
2. Making technical debt visible
3. Deciding when and how to resolve debt
4. Living with technical debt

# Not all technical debt is the same

| | Reckless | Prudent |
|---|---|---|
| **Deliberate** | *"We don't have time for design"* | *"We must ship now and deal with consequences (later)"* |
| **Inadvertent** | *"What's layering?"* | *"Now we know how we should have done it"* |

https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

# Group Activity

Describe two plausible examples of technical debt in the midterm scenario (MMate).

1. Deliberate, prudent
2. Reckless, inadvertent

Discuss the reason for incurring debt (e.g., value added?) and the debt payback strategy

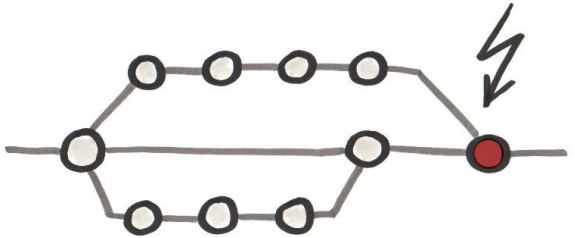
# How can we avoid (inadvertent) technical debt?

# Common Anti-Patterns

- Not having a QA process! Or no-one follows it

# Common Anti-Patterns

- Not having a QA process! Or no-one follows it
- Bad version control practices
  - Everyone commits to the main branch
  - Long-lived feature branches
  - Huge PRs

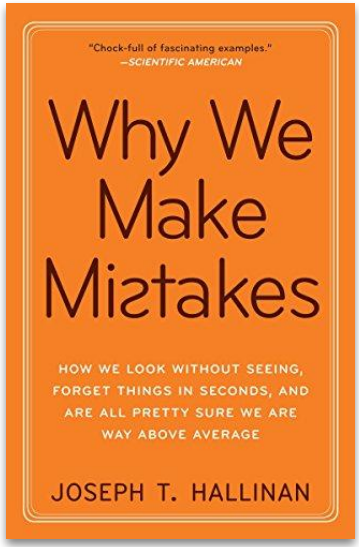


# Common Anti-Patterns

- Not having a QA process! Or no-one follows it
- Bad version control practices
- Slow and encumbering QA processes
  - changes take forever to get merged
  - time could be better spent on new features

# Common Anti-Patterns

- Not having a QA process! Or no-one follows it
- Bad version control practices
- Slow and encumbering QA processes
- Reliance on repetitive manual labor
  - focused on superficial problems rather than structural ones
  - results may vary (e.g., manual testing)
  - mistakes will happen!



# Case Study: Knight Capital

## Knightmare: A DevOps Cautionary Tale

D7 DevOps April 17, 2014 6 Minutes

I was speaking at a conference last year on the topics of DevOps, Configuration as Code, and Continuous Delivery and used the following story to demonstrate the importance making deployments fully automated and repeatable as part of a DevOps/Continuous Delivery initiative. Since that conference I have been asked by several people to share the story through my blog. This story is true – this really happened. This is my telling of the story based on what I have read (I was not involved in this).


In layman's terms, Knight Capital Group realized a $460 million loss in 45-minutes. Remember, Knight only has $365 million in cash and equivalents. **In 45-minutes Knight went from being the largest trader in US equities and a major market maker in the NYSE and NASDAQ to bankrupt.**