Technical Debt

17-313 Fall 2024
Foundations of Software Engineering
https://cmu-17313q.github.io
Eduardo Feo Flushing

Sources:
- Managing Technical Debt. Ipek Ozkaya. CMU SEI

Administrivia

- P3A due tonight
- You can collect your cheat sheets
- There was a post-midterm activity with bonus points

Midterm Feedback

- Avoid generic answers
- Risk: What is a mitigation plan?
- Architecture:
  - What can go wrong if we don't plan it?
- Design Docs:
  - Definition of success: Why would you use microservices in the first place?
- Code Reviews:
  - Checklist is for reviewers not for reviewee
  - You shouldn't include checks that can be automated (why not?)
- Continuous Integration
  - What is it? What advantages does it bring to the process?

Midterm Wrapper

Model Answers & Clearer Expectations:

Instead of direct answers, we will provide model solutions that illustrate how to approach questions effectively. This will help you understand the level of detail and structure expected without compromising the need for critical thinking.

Midterm Wrapper

Exam Clarity & Time Management:

Some questions may feel under-specified, and this is intentional. It encourages you to apply concepts critically, as real-world problems often require interpretation. Specifying too much would limit this learning opportunity, so the goal is to develop your problem-solving skills. ***If we specify exactly what we expect****, we are essentially* ***giving away the answer****, which* ***limits your ability to think*** *through and apply the concepts independently.*

Midterm Wrapper

Tailored Recitations & More Review Sessions:

*While we can't offer more than one review session, we will schedule an optional review session with CAs for those who want extra practice. Additionally, **you are encouraged** to attend **office hours to ask questions about both theory and application**.*

Midterm Wrapper

Course Material & Application:

*You are encouraged to **take notes** during lectures, as it helps improve understanding and retention. Consider handwriting notes to engage more actively. To strengthen the connection between theory and application, we will add more theory questions to the homework, giving you more practice in applying concepts in different scenarios*

Just so you know, taking notes effectively is challenging **(probably impossible)** if you are multitasking on your laptop.

Tips for success

- Frequently review material
  - E.g., active recall and spaced repetition
- Come to office hours to clarify concepts
- Take notes in class
- Engage in class discussions
  - E.g., Ask one question in class each week
  - If you are consistently **eating chocolate** every Sunday, Tuesday, and Thursday, between 4pm - 5:15pm, you are doing fine
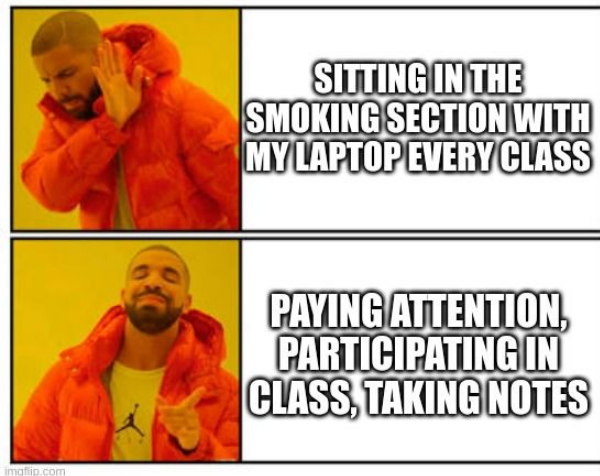
Learning Goals

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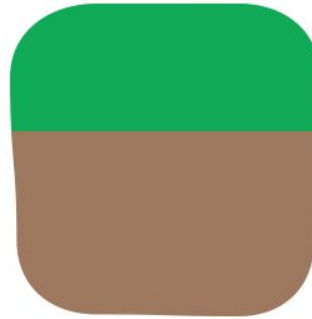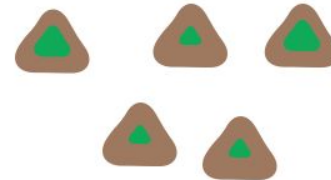- Learn techniques for managing technical debt

S3D

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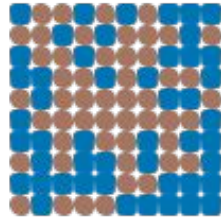
Cruft causes changes to take **more effort**

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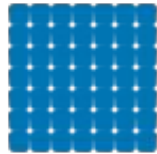
If we compare one system with *a lot of cruft...*

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
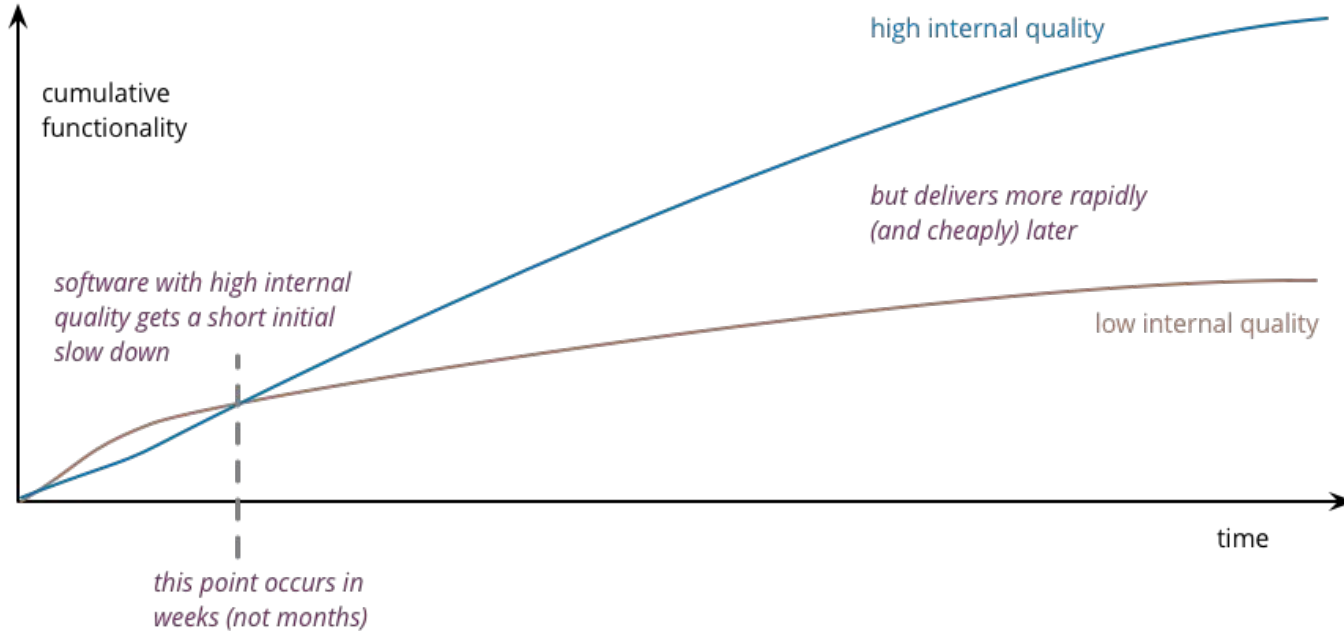
S3D

Carnegie Mellon University

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

S3D

Carnegie
Mellon
University

High internal quality is an investment

cumulative functionality

high internal quality

but delivers more rapidly (and cheaply) later

software with high internal quality gets a short initial slow down

low internal quality

this point occurs in weeks (not months)

time

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

...

S3D

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

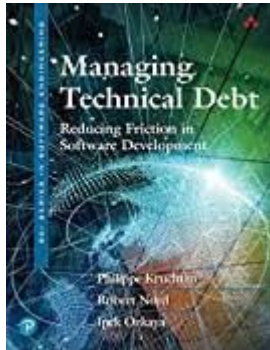# EVERYONE

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.
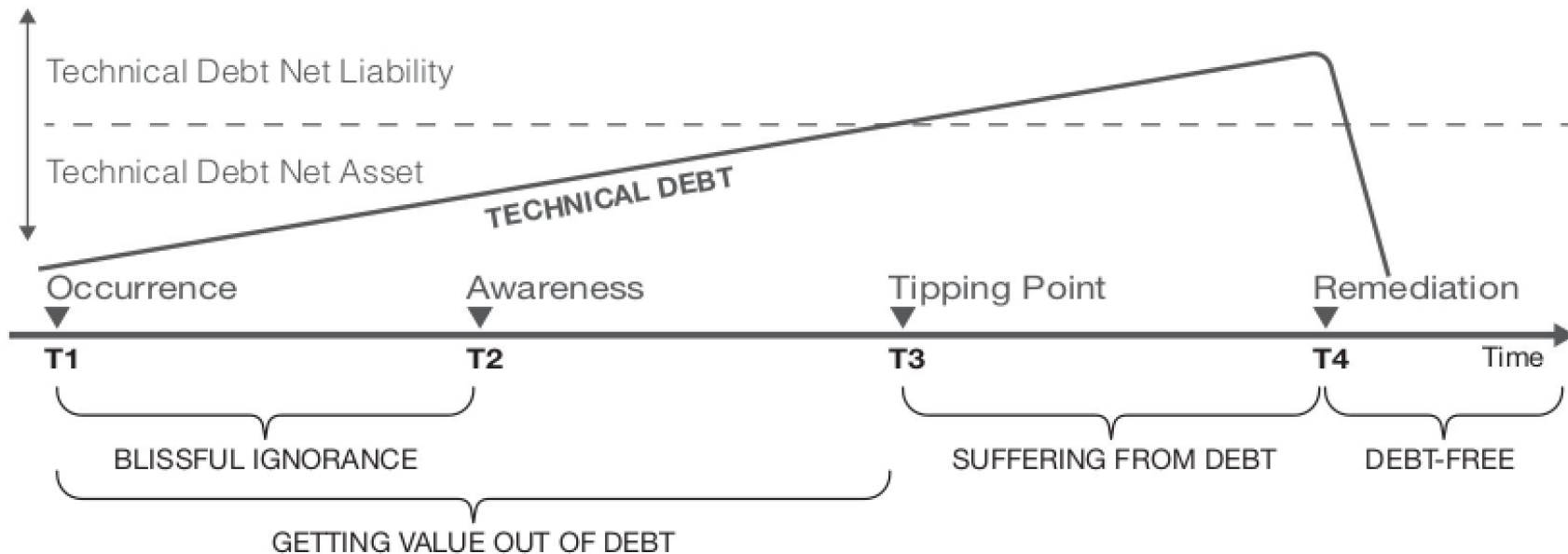Philippe Kruchten, Robert Nord, Ipek Ozkaya

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

GETTING VALUE OUT OF DEBT

SUFFERING FROM DEBT

DEBT-FREE

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

S3D

Carnegie
Mellon
University

Group Activity

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

1. Deliberate, prudent
2. Reckless, inadvertent

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

Carnegie
Mellon
University

How can we avoid (inadvertent) technical debt?

Common Anti-Patterns

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

Carnegie
Mellon
University

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
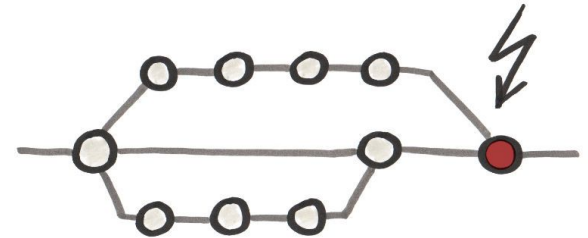
	COMMENT	DATE
o	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
o	ENABLED CONFIG FILE PARSING	9 HOURS AGO
o	MISC BUGFIXES	5 HOURS AGO
o	CODE ADDITIONS/EDITS	4 HOURS AGO
o	MORE CODE	4 HOURS AGO
o	HERE HAVE CODE	4 HOURS AGO
o	AAAAAAAA	3 HOURS AGO
o	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
o	MY HANDS ARE TYPING WORDS	2 HOURS AGO
o	HAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
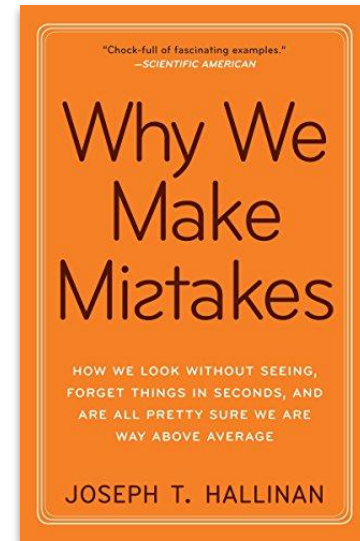MESSAGES GET LESS AND LESS INFORMATIVE.

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!

"Chock-full of fascinating examples."
—SCIENTIFIC AMERICAN

Why We Make Mistakes

HOW WE LOOK WITHOUT SEEING,
FORGET THINGS IN SECONDS, AND
ARE ALL PRETTY SURE WE ARE
WAY ABOVE AVERAGE

JOSEPH T. HALLINAN

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.**