

Technical Debt

17-313: Foundations of Software Engineering
<https://cmu-313.github.io>

Michael Hilton and **Josh Sunshine**
Spring 2026

Administrivia

- Midterm 2 on Thursday
- Midterm 2 review was yesterday, slides online

Learning Goals

- Understand the concepts of software quality and 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 definition #1

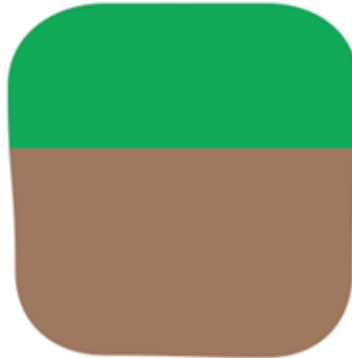
“Although immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. **Shipping first time code is like going into debt.** A little debt speeds development 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...” Ward Cunningham

<https://dl.acm.org/doi/pdf/10.1145/157710.157715>

Technical debt definition #2

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>

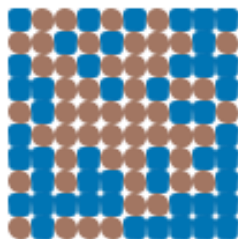
Technical debt definition #3

“In software-intensive systems, technical debt is a collection of design or implementation constructs that are **expedient in the short term**, but set up a technical context that can **make future changes more costly or impossible**. Technical debt presents an **actual or contingent liability** whose impact is limited to **internal system qualities**, primarily maintainability and evolvability.”

<https://www.dagstuhl.de/seminars/seminar-calendar/seminar-details/16162>

Internal quality makes it easier to add features

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

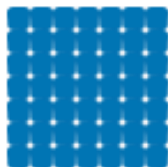


the cruft means new features take longer to build



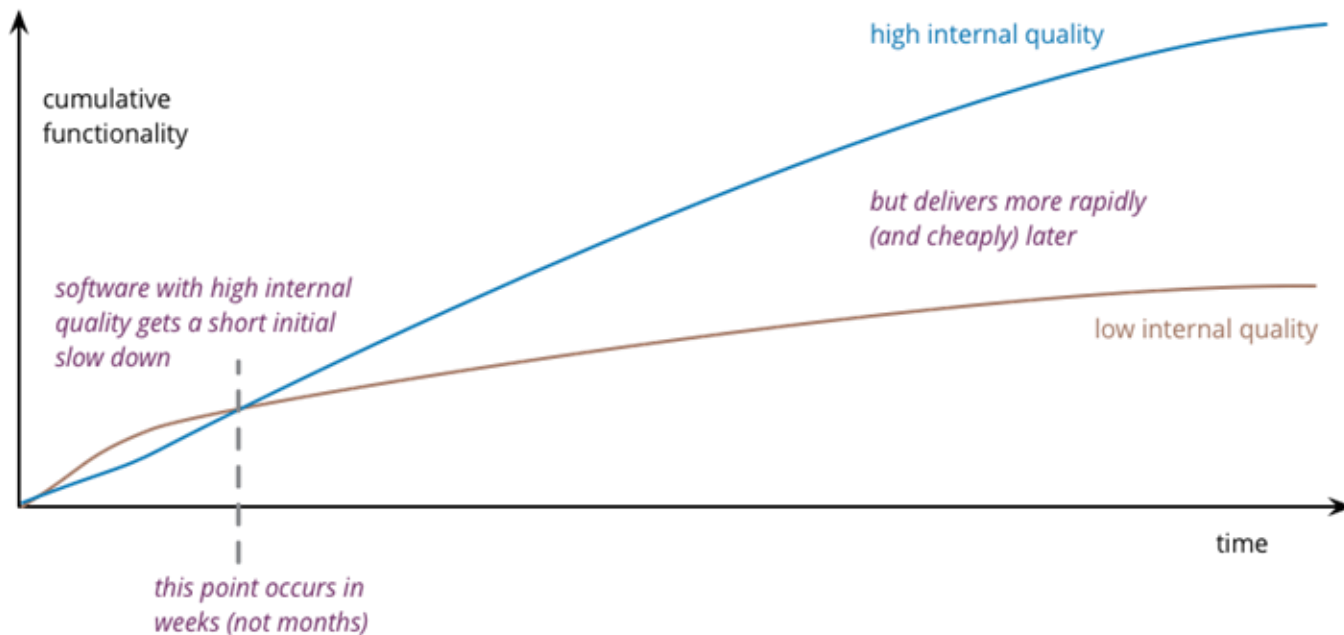
this extra time and effort is the cost of the cruft, paid with each new feature

...to an equivalent one without



free of cruft, features can be added more quickly

High internal quality is an investment



A better analogy?: Pollution



Daily AQI Color	Levels of Concern	Values of Index	Description of Air Quality
Green	Good	0 to 50	Air quality is satisfactory, and air pollution poses little or no risk.
Yellow	Moderate	51 to 100	Air quality is acceptable. However, there may be a risk for some people, particularly those who are unusually sensitive to air pollution.
Orange	Unhealthy for Sensitive Groups	101 to 150	Members of sensitive groups may experience health effects. The general public is less likely to be affected.
Red	Unhealthy	151 to 200	Some members of the general public may experience health effects; members of sensitive groups may experience more serious health effects.
Purple	Very Unhealthy	201 to 300	Health alert: The risk of health effects is increased for everyone.
Maroon	Hazardous	301 and higher	Health warning of emergency conditions: everyone is more likely to be affected.

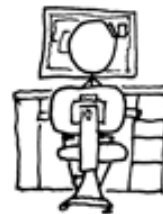
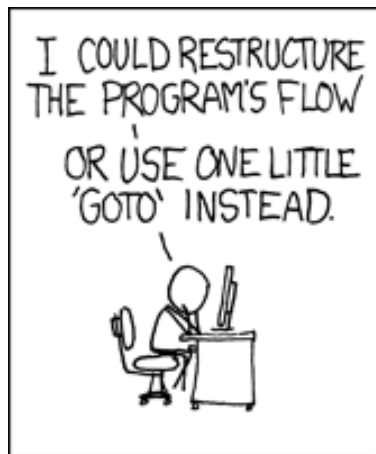
<https://www.airnow.gov/aqi/aqi-basics>

Participation activity part #1

Q1: What are two examples of technical debt that were in NodeBB when you started working on it? Explain both the "design and implementation construct" and how it made "future changes more costly or impossible?"

Q2: What is one example of technical debt you or your team added to NodeBB?

What actions cause technical debt?



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

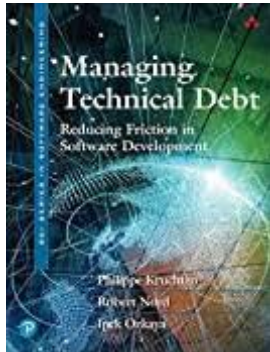


Bad: Too much technical debt

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



How to manage technical debt?

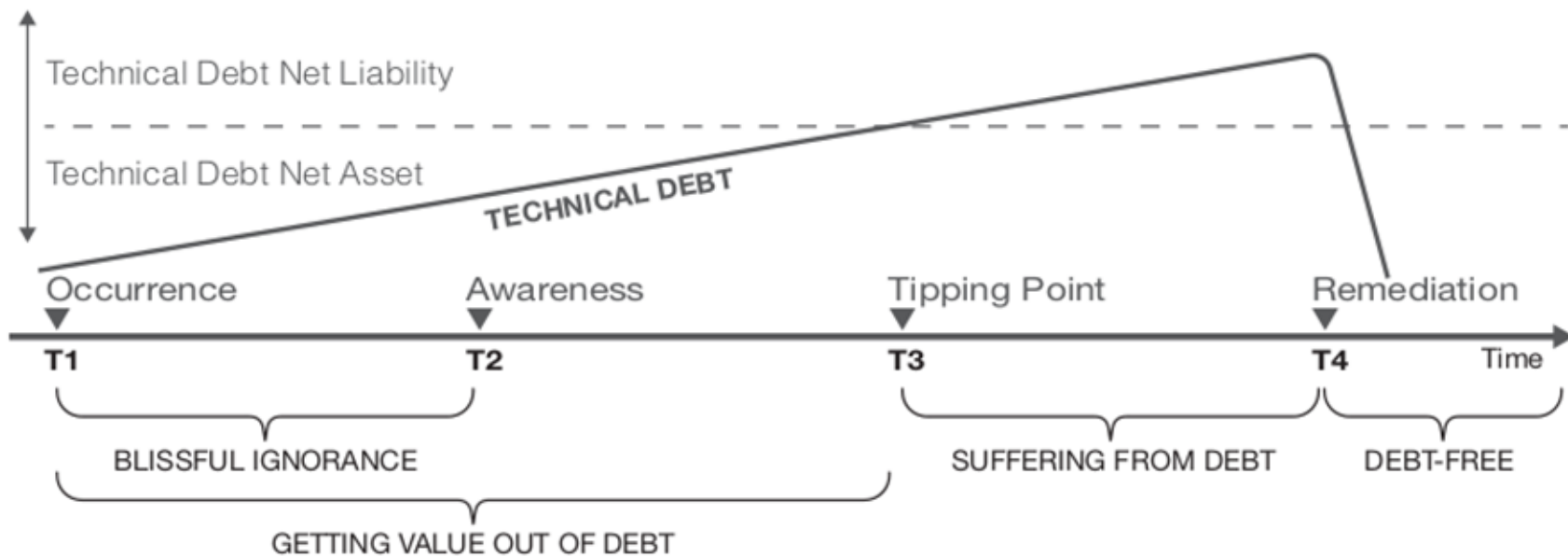


Managing Technical Debt: Reducing Friction in Software Development.
Philippe Kruchten, Robert Nord, Ipek Ozkaya

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.
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?



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	<i>“We don’t have time for design”</i>	<i>“We must ship now and deal with consequences (later)”</i>
Inadvertent	<i>“What’s layering?”</i>	<i>“Now we know how we should have done it”</i>

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

**How can we avoid (inadvertent)
technical debt?**

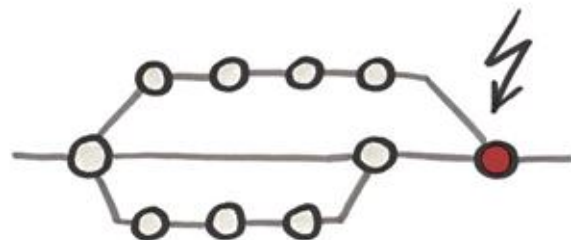
Common Anti-Patterns

- 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



	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE.	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSOKLFT	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAANDS	2 HOURS AGO

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

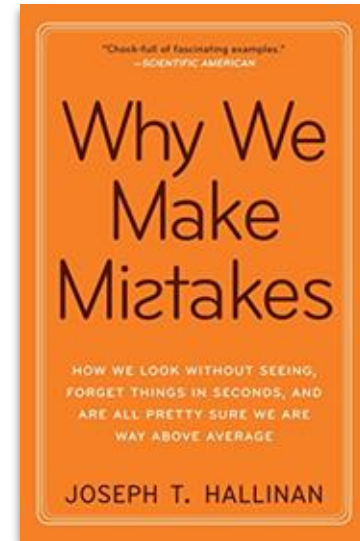
Common Anti-Patterns

- 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!



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. Knight only had \$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.**

Participation activity part #2

Q3: What are two ways AI agents (e.g. Cursor, Claude Code) can improve technical debt?

Q4: What are two ways AI agents (e.g. Cursor, Claude Code) can make technical debt worse?

Impact of AI agents ... The tale of two papers

- <https://www.youtube.com/watch?v=YEth7AqaezI&t=1s>
- <https://cmustrudel.github.io/papers/msr2026agarwal.pdf>

Summary:

- Software Quality is hard
- Life involves tradeoffs

