Managing Risk and Technical Debt

17-313 Fall 2026

Foundations of Software Engineering

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

Eduardo Feo Flushing



Administrivia

- P4A deadline: Tuesday, end of day. Plan your interviews.
- Gradebook is online (since October 19; check the Slack announcement).
- P4B posted soon.
- Recitation tomorrow.

Learning Goals

- Prioritize project risks and choose appropriate controls.
- Identify technical debt and explain its future cost.
- Justify a decision to accept or reduce a specific debt.

Outline

- What could prevent our project from succeeding?
- How do we assess and control risk?
- How do today's choices affect tomorrow's work?
- How do we decide what debt to accept or repay?

What Is Software Risk?

- A risk is a measure of the potential inability to achieve overall program objectives within defined cost, schedule, and technical constraints.
- A risk is about the future: a possible event that has not happened yet.
- This is different from an **issue**: a problem that has already occurred.



Conrow, E. 2003. Effective Risk Management: Some Keys to Success, 2nd ed. Reston, VA, USA: American Institute of Aeronautics and Astronautics (AIAA).

Likelihood and Impact



Probability (likelihood) of failing to achieve a particular outcome.



Impact (consequences) of failing to achieve that outcome.

Likelihood alone cannot determine priority.

Internal and External Risks

Originates Within the Project



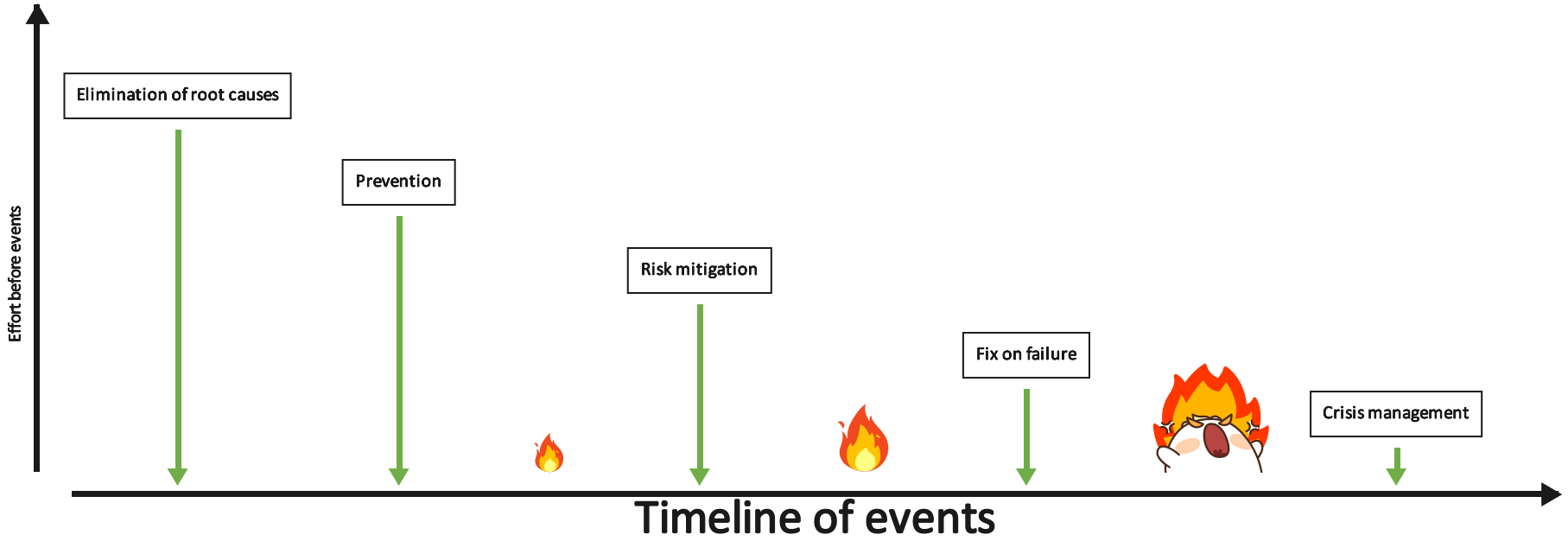
Scope, schedule, staffing, design decisions.

Originates Outside the Project



Vendor changes, market shifts, natural events.

Levels of Risk Management



McConnell, S. Rapid Development: Taming Wild Software Schedules. 1996.

Levels of Risk Management

1. **Elimination of root causes:**
   - Identify and eliminate factors that make it possible for risks to exist at all.
2. **Prevention:**
   - Implement and execute a plan to identify risks and prevent them from becoming problems.
3. **Risk mitigation:**
   - Plan ahead to provide resources to cover risks if they occur, but do nothing to eliminate them.
4. **Fix on failure:**
   - Detect and react to risks quickly, but only after they have occurred.
5. **Crisis management:**
   - Fire fighting; address risks only after they become problems.

Not covered in this class

McConnell, S. Rapid Development: Taming Wild Software Schedules. 1996.

Outline

- What could prevent our project from succeeding?
- **How do we assess and control risk?**
- How do today's choices affect tomorrow's work?
- How do we decide what debt to accept or repay?

From Levels to Tasks

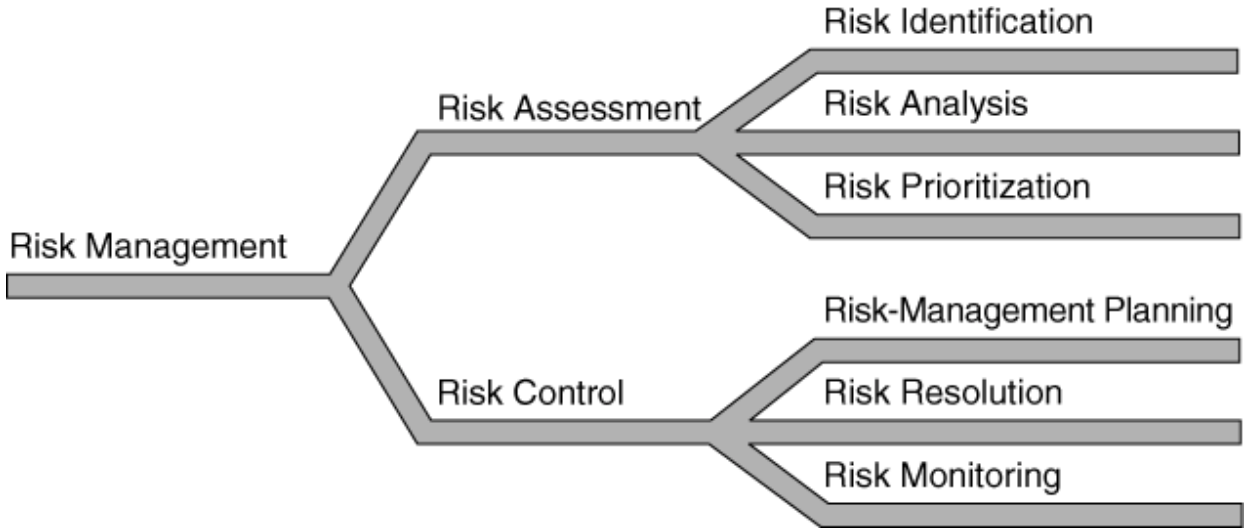
To eliminate causes, prevent failures, or prepare for their consequences, we need to:

- **Assess risks:** identify what could go wrong, analyze causes and consequences, and prioritize.
- **Control risks:** plan and implement responses, then monitor their effectiveness.

Example: duplicated validation rules may become inconsistent.

Approach	What assessment helps us understand	What control puts into practice
Root-cause elimination	Multiple copies require manual synchronization.	Replace the copies with one shared implementation.
Prevention	Developers may overlook copies when making changes.	Add consistency tests and review checks.
Mitigation	Inconsistent validation could disrupt users.	Prepare a rollback procedure to limit disruption.

Tasks of Risk Management



Assessment identifies and prioritizes risks. Control plans, implements, and monitors the chosen responses.

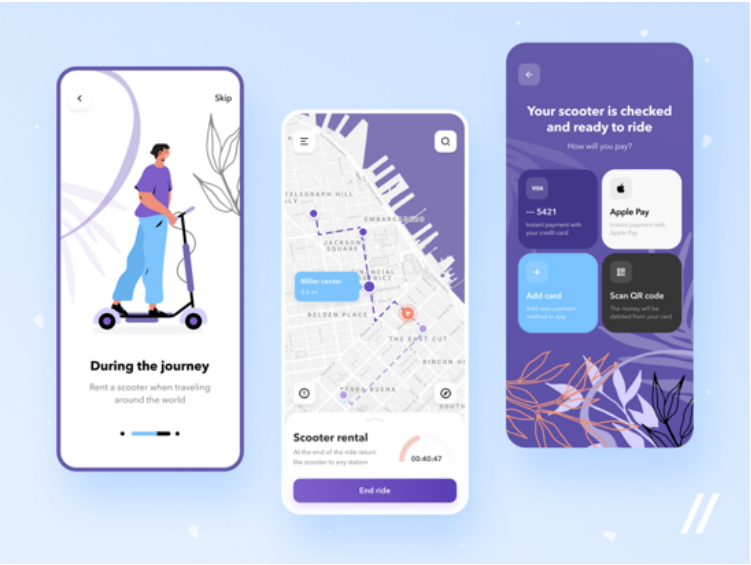
McConnell, S. Rapid Development: Taming Wild Software Schedules. 1996.

Activity 1a: Risk Identification

Your team is building a scooter-rental app: riders find a nearby scooter on a map, unlock it, ride, and end the rental from their phone.

What risks exist for the scooter app?

Identify three plausible risks, including at least one beyond a code defect.



2 min individually · 2 min discussing

Qualitative Risk Assessment

TABLE III. Risk assessment matrix

RISK ASSESSMENT MATRIX				
SEVERITY / PROBABILITY	Catastrophic (1)	Critical (2)	Marginal (3)	Negligible (4)
Frequent (A)	High	High	Serious	Medium
Probable (B)	High	High	Serious	Medium
Occasional (C)	High	Serious	Medium	Low
Remote (D)	Serious	Medium	Medium	Low
Improbable (E)	Medium	Medium	Medium	Low
Eliminated (F)	Eliminated			

MIL-STD-882E

Severity × probability: a contextual low/medium/high rating is often enough.

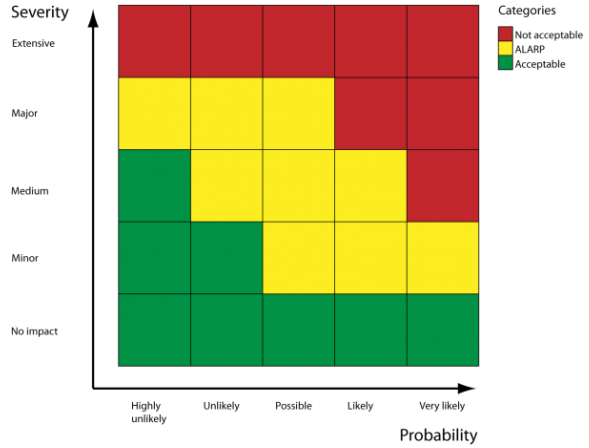
Risk Exposure and Prioritization

Risk	Probability	Size of Loss (weeks)	Risk Exposure (weeks)
Overly optimistic schedule	50%	5	**2.5**
Additional features added by marketing	35%	8	**2.8**

Activity 1b: Risk Analysis

For the three risks from Activity 1a, rate likelihood and impact, and state your assumptions.

Select your two most important risks to carry forward.



2 min individually · 2 min discussing

Numbers Need Assumptions

- Risk analysis can become a numbers game, used to justify a decision that was already made.
- A number is only as good as the assumptions behind it.

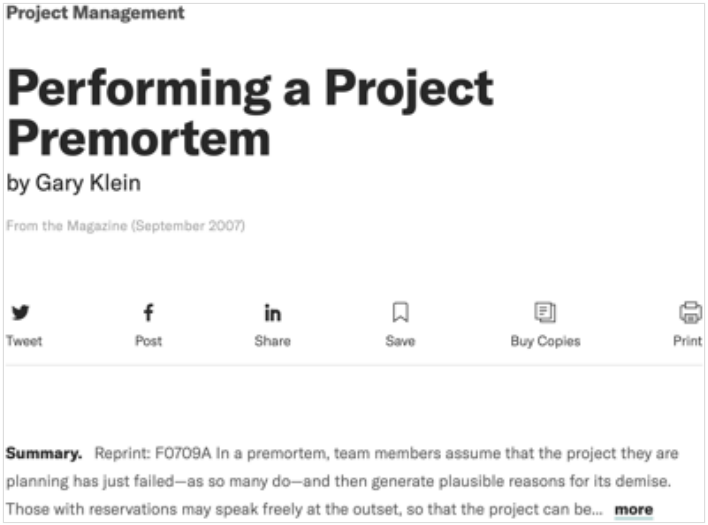
- Record the assumptions behind each estimate.
- Acknowledge the uncertainty in the estimate itself.
- Revise estimates as you learn more.

Risk Control and Ownership

- What steps can be taken to avoid or mitigate the risk?
- Can you better understand or eliminate the cause?
- Who is responsible if this risk materializes?

Premortems

- Imagine the project has already failed.
- Ask: "What happened?"
- Unlike a typical critique, a premortem assumes failure up front, so people speak freely instead of defending the plan.



Project Management

Performing a Project Premortem

by Gary Klein

From the Magazine (September 2007)

Tweet Post Share Save Buy Copies Print

Summary. Reprint: F0709A In a premortem, team members assume that the project they are planning has just failed—as so many do—and then generate plausible reasons for its demise. Those with reservations may speak freely at the outset, so that the project can be... **more**

This is the technique behind Activity 1a: imagining failure surfaces risks people might otherwise hold back.

Klein, G. Performing a Project Premortem. Harvard Business Review, 2007.

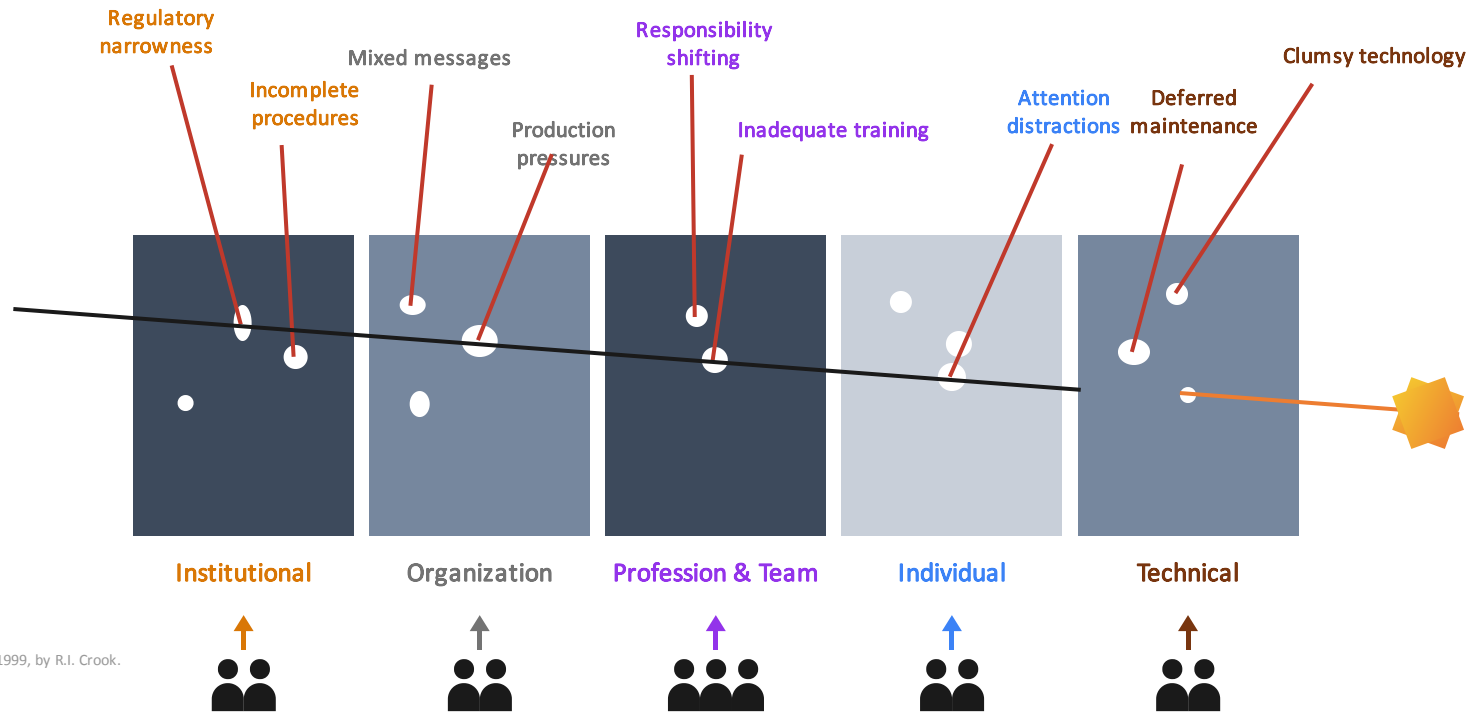
Activity 1c: Risk Elimination and Mitigation

For your highest-priority risk, how would you eliminate or mitigate it?

1. Decide: can the cause be eliminated, or only its likelihood or impact reduced?
2. Specify a response, an owner, and a review trigger.

2 min completing the response · 2 min reporting and debrief

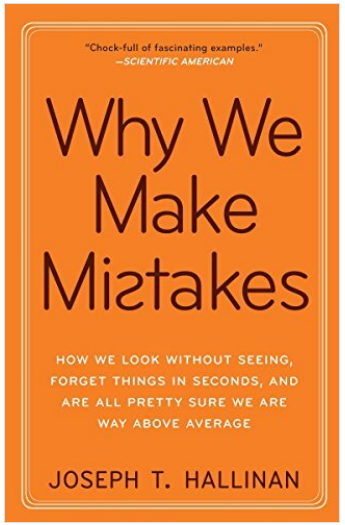
Swiss Cheese Model



Modified from Reason, 1999, by R.I. Crook.

Why Human Error Persists

- We cannot reliably find our own typos: it is hard to review attention to our own, familiar work.
- Boredom and repetitive tasks increase error rates.
- We cannot remove human error, but we can design processes that prevent and detect mistakes.



Automation and Human Review

Automate what we can. Review what we cannot.

- Automated checks are fast, consistent, and catch what they were built to catch.
- For everything else, a human reviewer brings judgment and context an automated check does not have.

Code Review: What Deserves Attention?

- Understand the author's intent before judging the implementation.
- Focus on design and failure cases, not style.
- Automate formatting and other mechanical checks; don't spend review time on them.
- Give feedback respectfully: a reviewer critiques code, not the person who wrote it.

A Short Checklist

- A checklist does not replace judgment; it protects against forgetting something you already know matters.
- Checklists help manage complex processes where a missed step is easy and costly.

Before merging a change, ask:

- Does this change have a test for the behavior it adds or fixes?
- Does it handle the failure cases, not just the happy path?
- Would someone unfamiliar with this code understand it in six months?

Part II: Technical Debt and Future Project Risk

Technical Debt: Example

Technical debt: a shortcut now that makes future changes more costly.

To meet the sprint deadline, a team copies the same discount-calculation code into checkout and invoice generation.

Later, the discount rules change.



Concept	Shared example
The debt	Two separate implementations of the same calculation.
The interest	Every rule change requires updating and testing both copies.
The related risk	Someone updates only one copy, so the checkout total and invoice disagree.

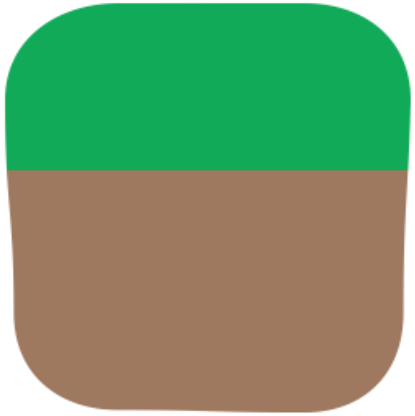
Outline

- What could prevent our project from succeeding?
- How do we assess and control risk?
- **How do today's choices affect tomorrow's work?**
- How do we decide what debt to accept or repay?

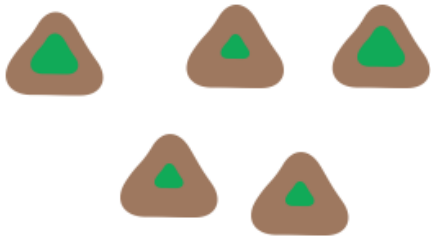
The Technical Debt Metaphor

*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.

martinfowler.com/bliki/TechnicalDebt.html

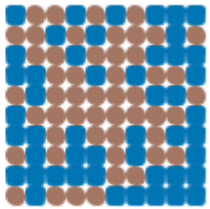
What Counts as Technical Debt?

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

Avram, Kruchten, Nord, Ozkaya. Reasoning about Technical Debt.

Internal Quality and the Cost of Change

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

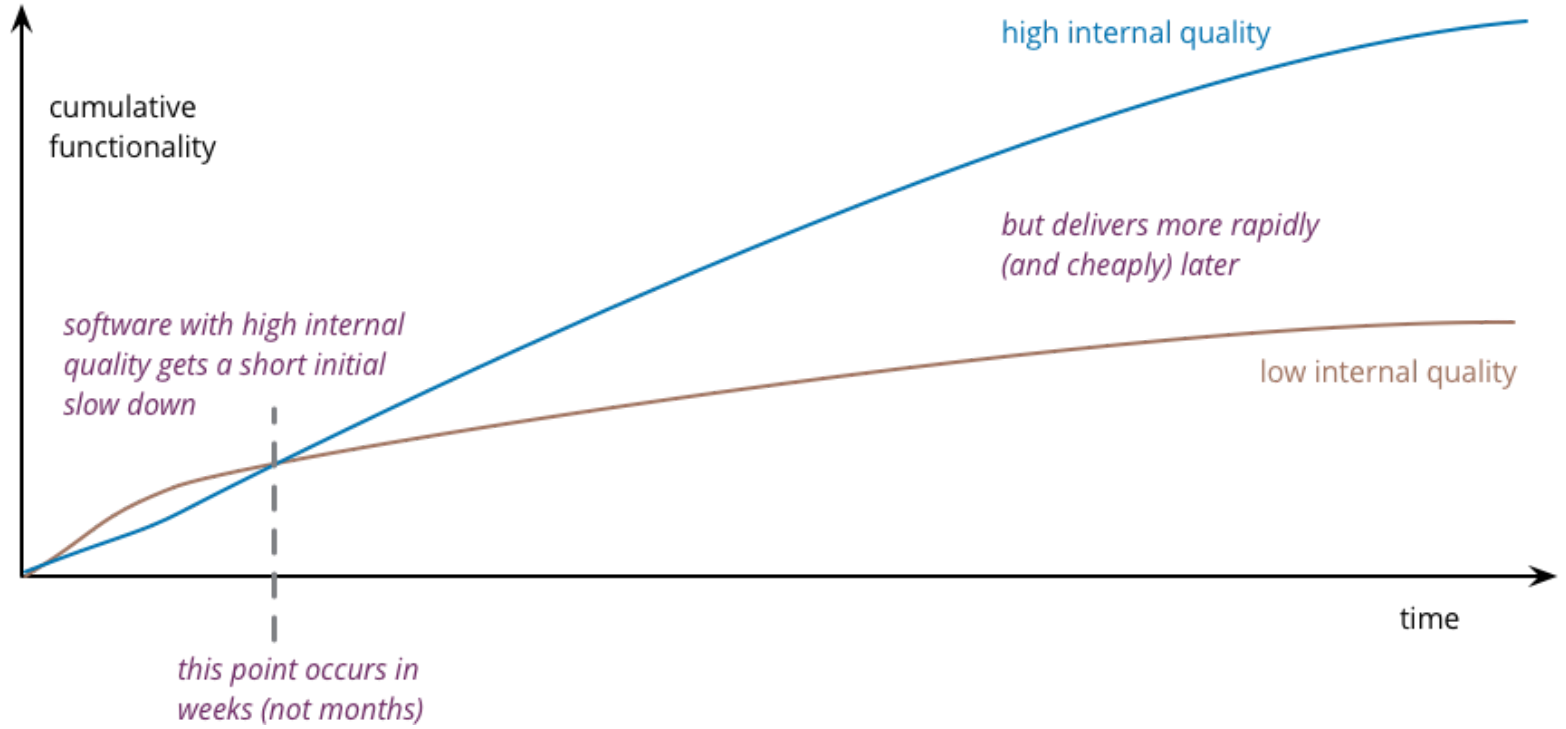
...to an equivalent one without





free of cruft, features can be added more quickly

Internal Quality as an Investment



Where Debt Comes From

- Tightly-coupled components.
- Poorly-specified requirements.
- Time pressure, with no time allotted to pay it back.
- Lack of knowledge: the team didn’t yet know a better way.
- Lack of ownership: no one is clearly responsible for noticing or fixing it.

All Software Comes With Debt

Bit rot (also known as software decay or environmental drift) is the gradual, natural deterioration of a software system's performance, reliability, and security over time, even if the software itself doesn't change.



Dependencies, APIs, and environments keep changing, even when your code does not.

The Consequences of Excessive Debt

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



Different Kinds of Technical Debt

A reasonable call at the time?

Knowingly taken on?

	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."*

martinfowler.com/bliki/TechnicalDebtQuadrant.html

Activity 2: Technical Debt in CoolCamel

Describe two plausible examples of technical debt in the CoolCamel scenario from exam 1:

1. One deliberate, prudent example.
2. One reckless, inadvertent example.

For each: discuss the reason for incurring the debt (did it add value?) and a debt payback strategy.