Technical Debt in Practice

Dr. Neil Ernst
Department of Computer Science
University of Victoria
nernst@uvic.ca

2 | Technical Debt

Research angle: Identify and understand when, and why, we take short-cuts in our engineering approach to software.

Practical angle: unpaid technical debt generates interest:

increased defect counts,

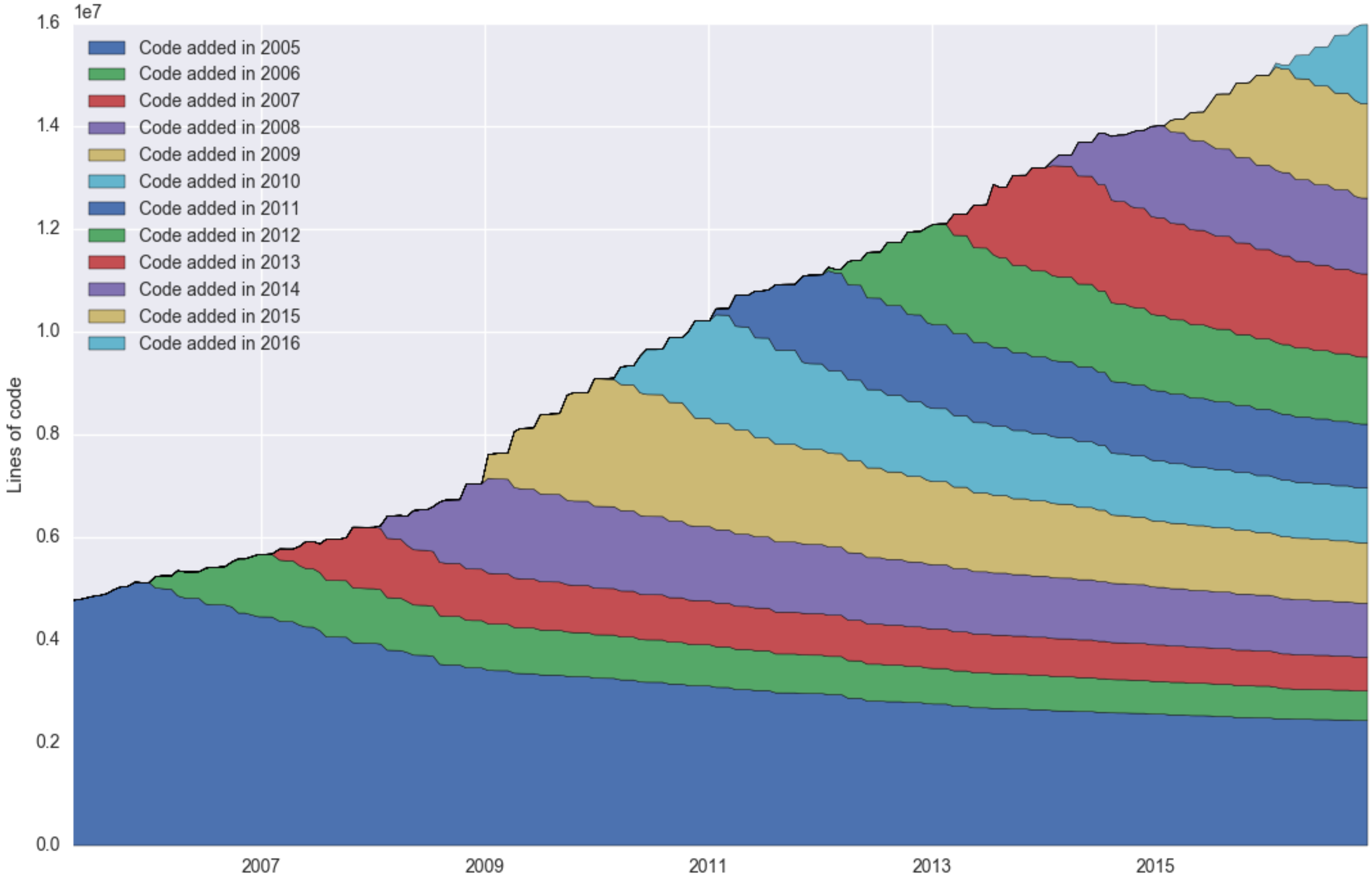
low quality (e.g. latency)

slow releases.

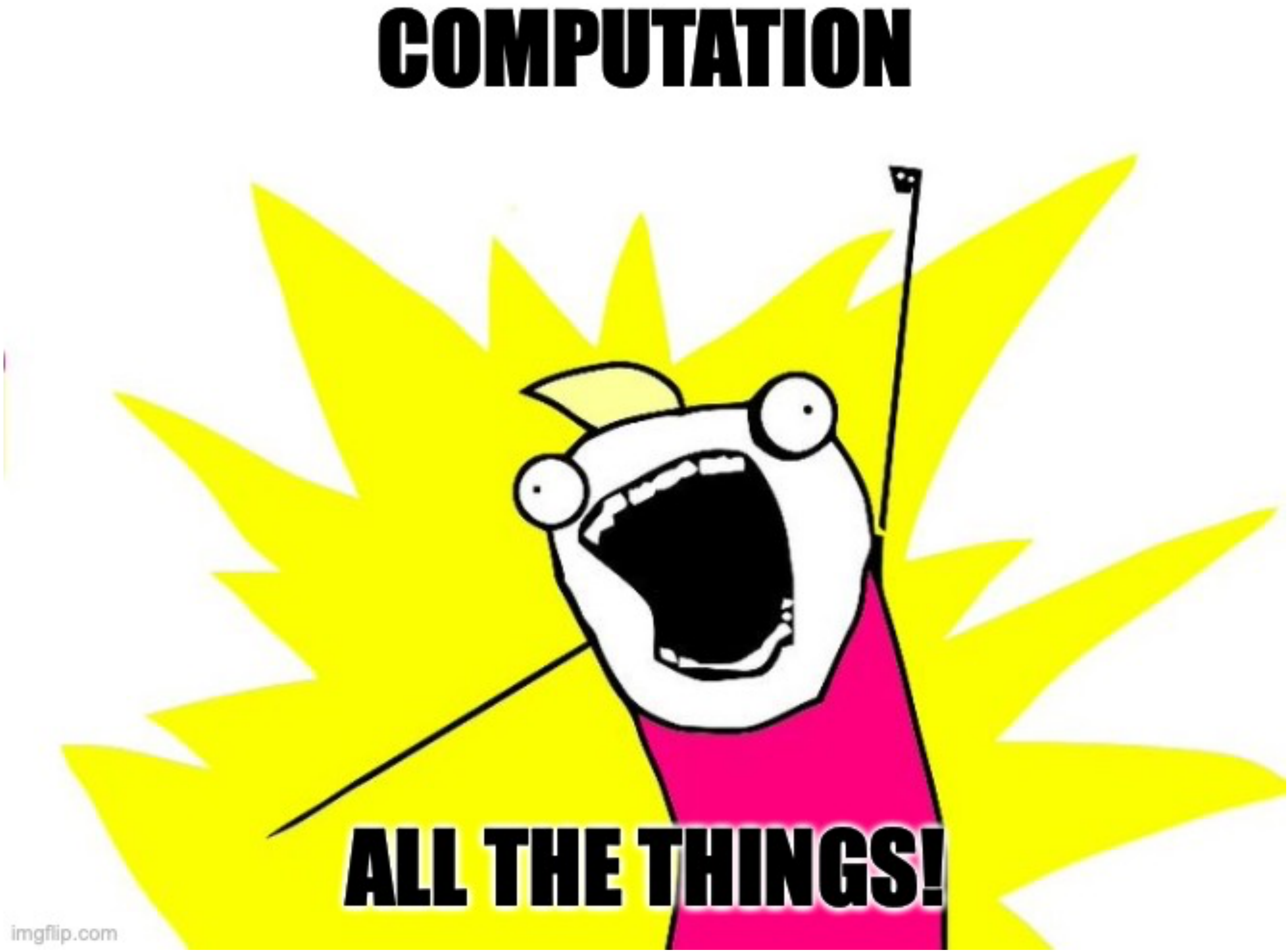
However: TD is everywhere and incurring debt is **not always bad!**

source: https://erikbern.com/2016/12/05/the-half-life-of-code.html

3 Software Will Not Go Away



Linux Kernel, additions by year


COMPUTATION
ALL THE THINGS!
imgflip.com

Software enters the Moneyball era

Moneyball: **identify** the key attributes in winning games, **measure** players against those attributes, **manage** teams to maximize those attributes

- On Base + Slugging
- Wins Above Replacement

Software analytics: **identify** key attributes in delivering software, measure delivery against those attributes, **manage** teams to maximize those attributes

- Mean time to repair
- Cycle time (feature idea to customer)
- Technical Debt

Technical Debt in Practice

- **What It Is**
- Why It Matters
- Identifying TD
- Managing TD
- Avoiding TD

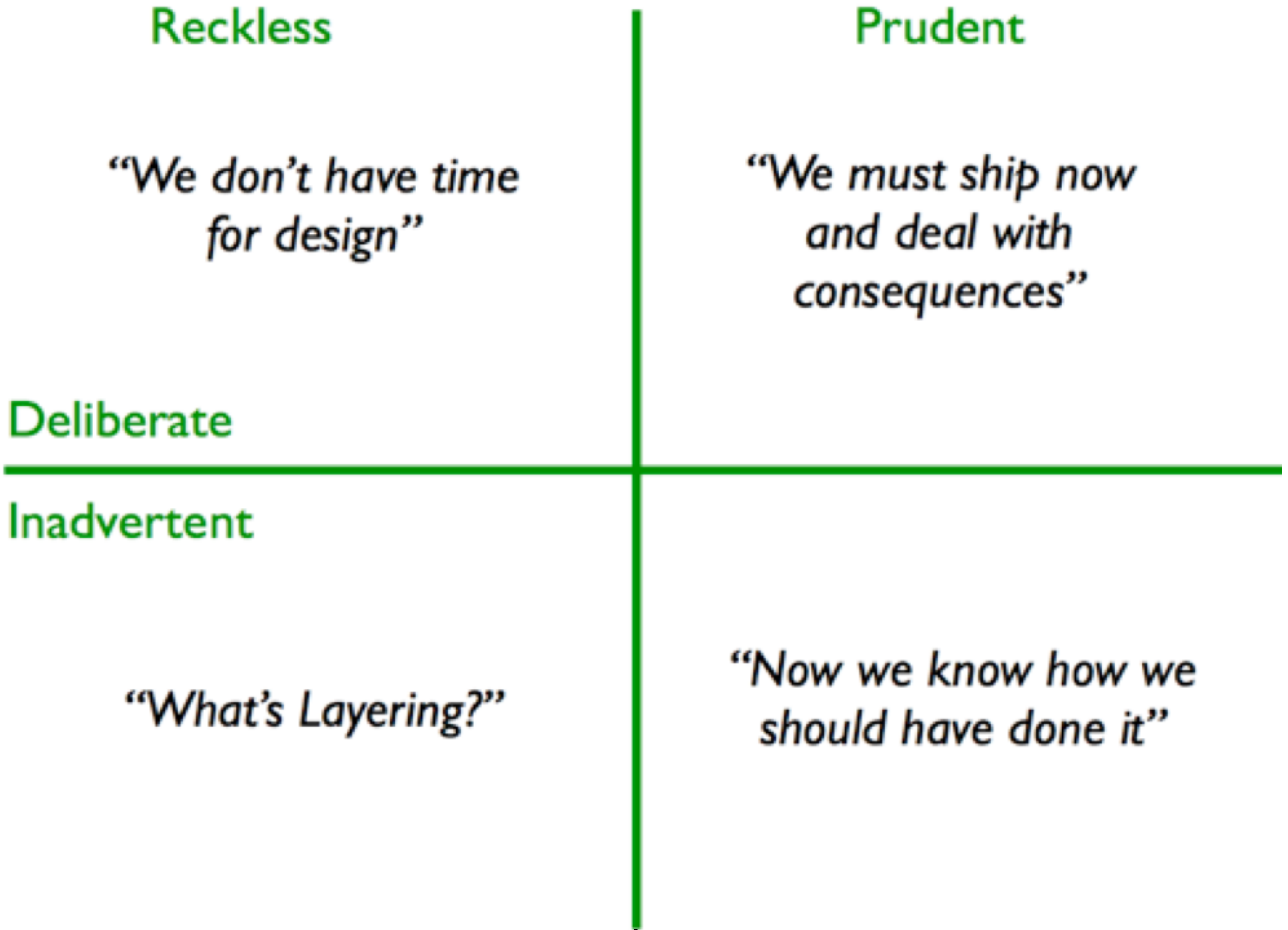
“Technical debt occurs when a design or construction approach is taken that's **expedient in the short term**, but that creates a technical context that **increases complexity and cost in the long term**.”

Steve McConnell (*Code Complete*)

“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.

Ward Cunningham

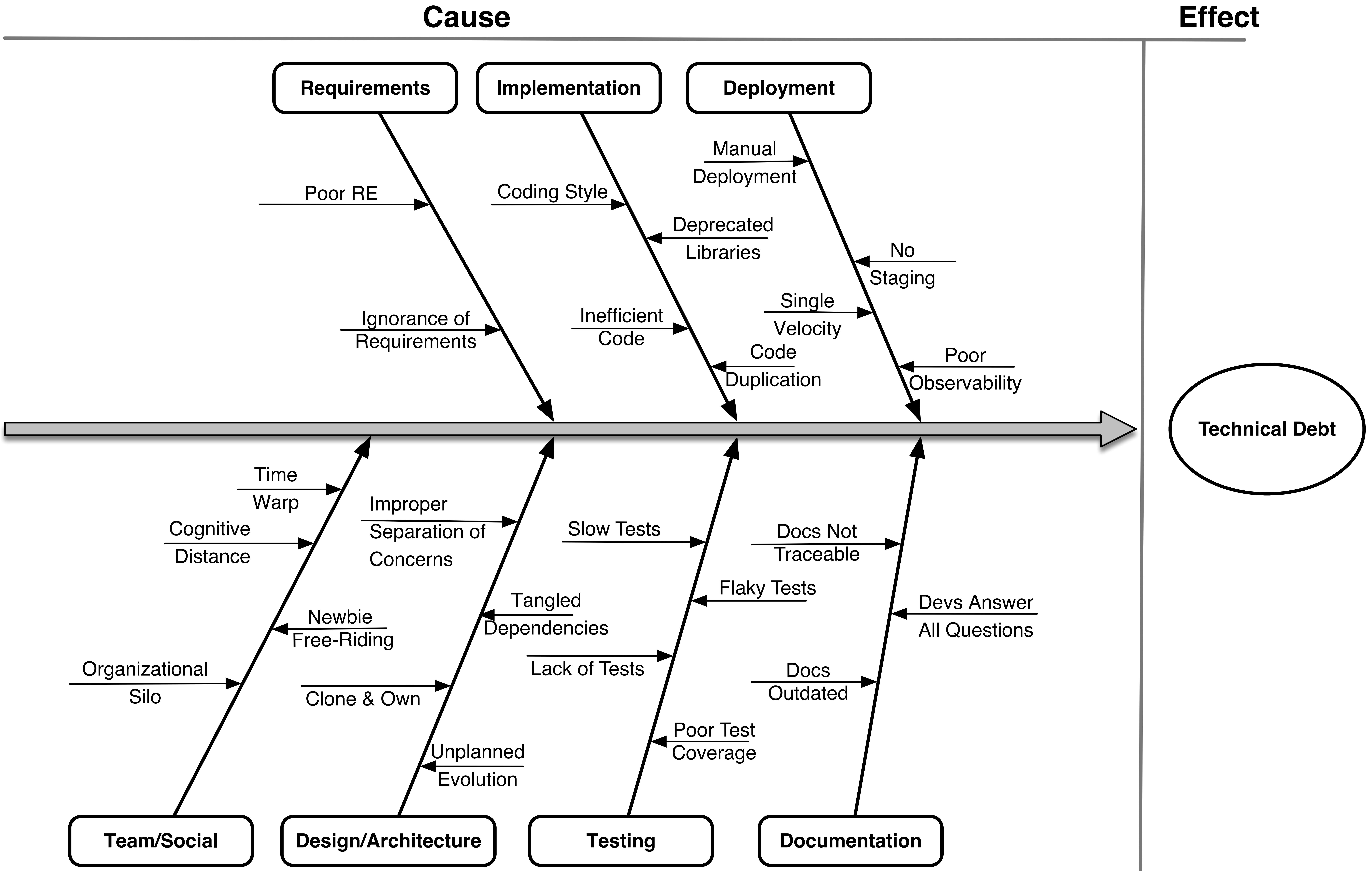
Ward Cunningham on TD: http://c2.com/doc/oopsla92.html



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

	Visible	Invisible
Positive Value	Visible Feature	Hidden, architectural feature
Negative Value	Visible defect	Technical debt

Kruchten, P. 2009. *What colour is your backlog?* Agile Vancouver Conference. http://pkruchten.wordpress.com/Talks.


Cause
Effect
Requirements
Implementation
Deployment
Poor RE
Ignorance of Requirements
Coding Style
Deprecated Libraries
Inefficient Code
Code Duplication
Manual Deployment
No Staging
Single Velocity
Poor Observability
Technical Debt
Time Warp
Cognitive Distance
Newbie Free-Riding
Organizational Silo
Improper Separation of Concerns
Tangled Dependencies
Clone & Own
Unplanned Evolution
Slow Tests
Flaky Tests
Lack of Tests
Poor Test Coverage
Docs Not Traceable
Devs Answer All Questions
Docs Outdated
Team/Social
Design/Architecture
Testing
Documentation

Technical Debt in Practice

- What It Is
- ➜ **Why It Matters**
- Identifying TD
- Managing TD
- Avoiding TD

Technical Debt in Big Science

Consider the ALMA telescope in Chile

Design → Construction → Commissioning → Science Operations

Over $1B budget

Expected to operate for decades

→ Design choices made 20 years ago constrain implementation today

e.g. Tango middleware

→ A big part is social debt: organizational shortcuts like poor teaming

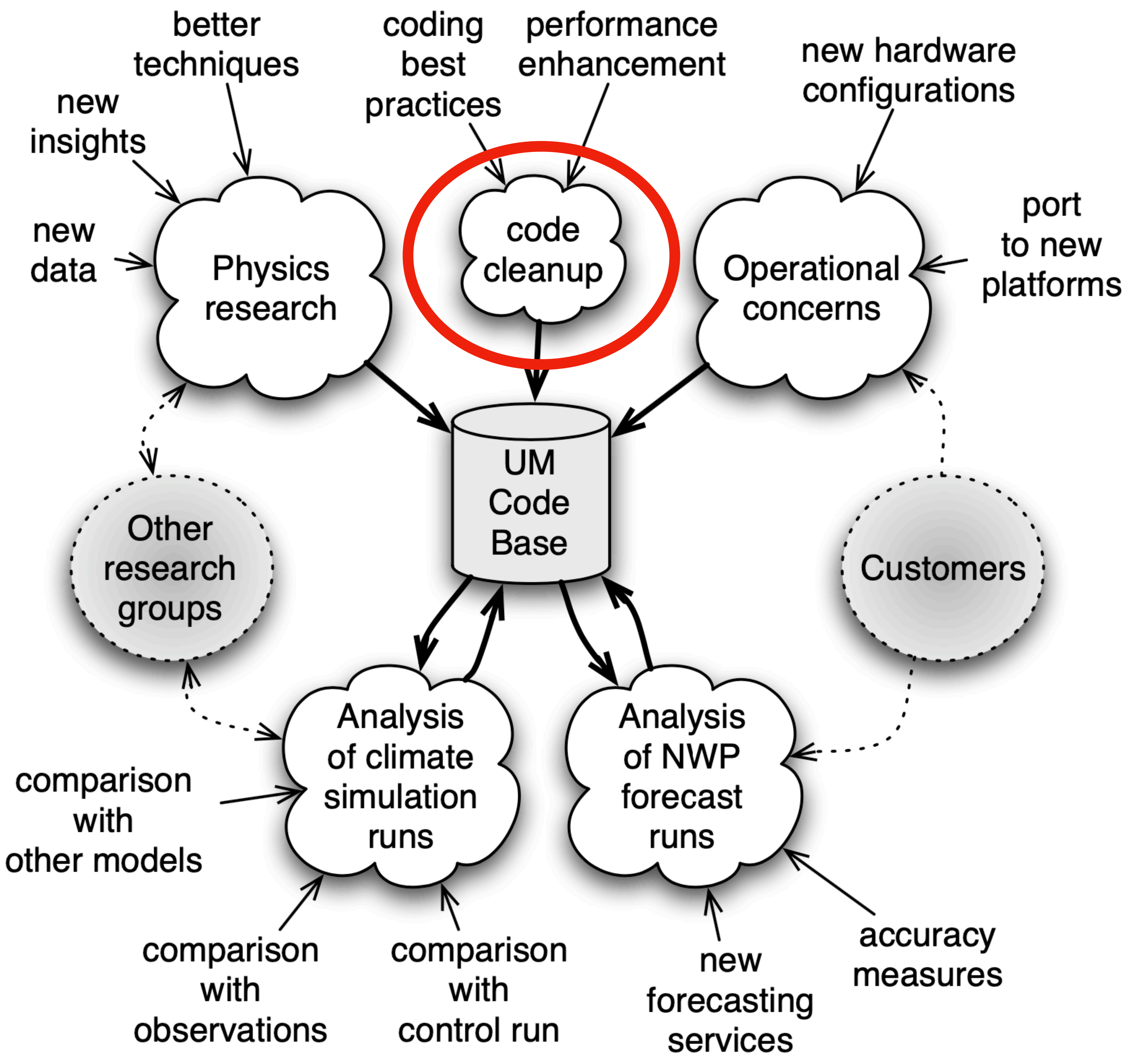
Technical Debt in Research Computing

LHC High Luminosity:

“Most of the current software, which defines our capabilities, was designed 15-20 years ago: there are many software sustainability challenges.”

Square Kilometre Array:

“we try and keep technical debt under control, maintaining a system where we can estimate what’s the amount of technical debt we are dealing with, and using capacity allocation to prevent it from diverging to an uncontrollable amount”



Easterbrook and Johns, Engineering the Software for Understanding Climate Change, *Computing in Science & Engineering*, 2009

Conway's law creates long-term risk

> “
>
> organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.
>
> —M. Conway

SKA - Central intentions and distributed design

System HQ

Data Processor

Design

Signal Processor

Design

Data Transport

Design

Array - Aus.

Design

Array - S.A.

Design

Science Management

Design

Regional Centre

Design

So What To Do?

- Identify, manage, avoid
- Research software development:
  - **many stakeholders**: local department computing, admin, faculty, students
  - **many constraints**: low budgets, staff turnover, pressure to publish, security, etc,
  - **Legacy** systems to maintain, for little reward (currently!)new technology constantly emerging
  - **Lack of resources** and time to do the above!

Technical Debt in Practice

- What It Is
- Why It Matters
- ➔ **Identifying TD**
- Managing TD
- Avoiding TD

20 | Identify

Self-admitted TD: code flags to return to (“fixme” or “TD”)

TD tools

- Sonarqube, Codescene, Code Sonar, Code Inspector …
- Key: properly configure the tool.
- Track the change over time!
- Expect to find 7-15% debt in your backlog

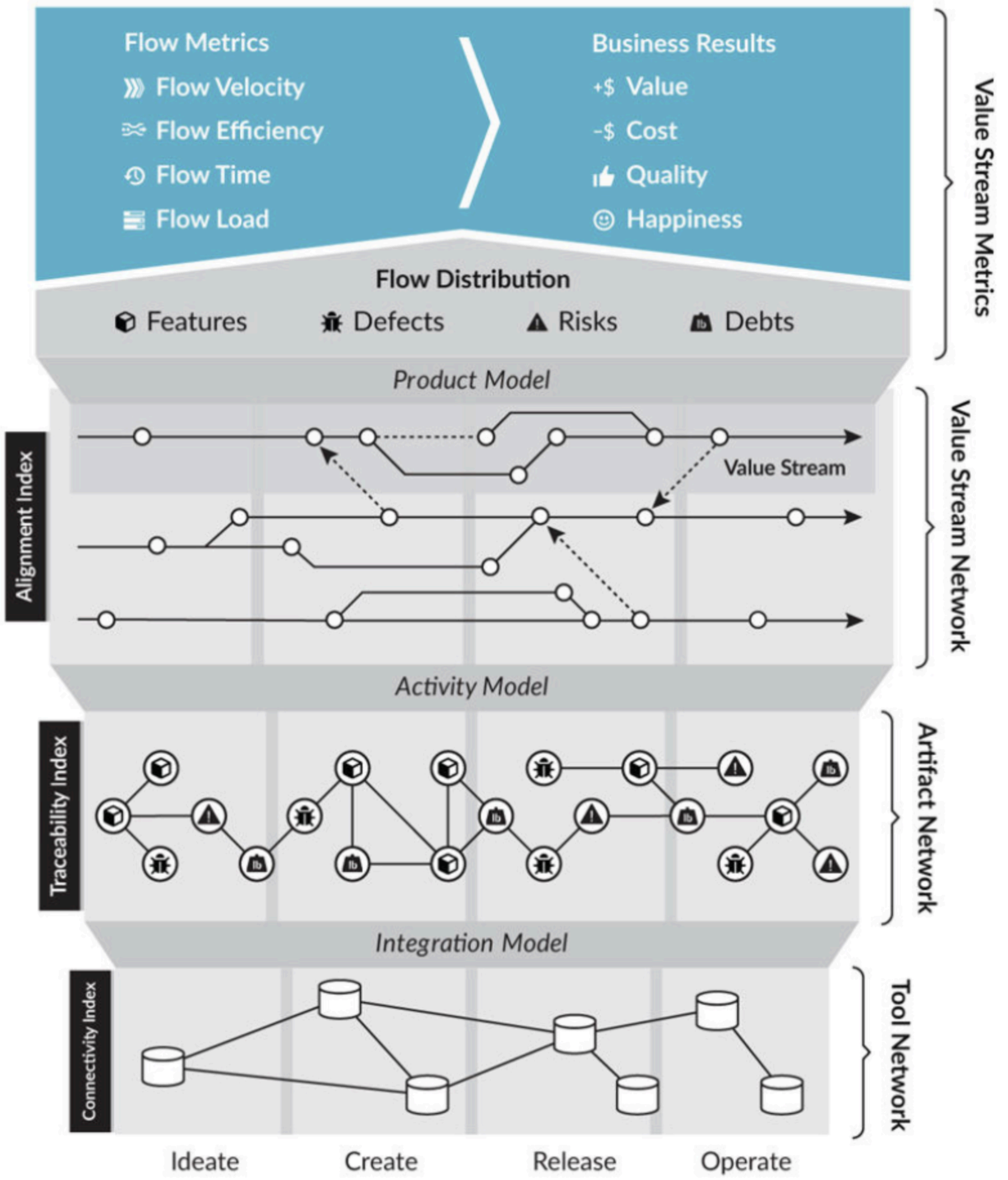
TD is not just code or design

- Tests, Infrastructure as Code, social - look broadly

Self-Admitted Technical Debt

```java
            addedDirs.put(vPath, vPath);

            if (!skipWriting) {
                final ZipEntry ze = new ZipEntry(vPath);

                // ZIPs store time with a granularity of 2 seconds, round up
                final int millisToAdd = roundUp ? ROUNDUP_MILLIS : 0;

                if (fixedModTime != null) {
                    ze.setTime(modTimeMillis);
                } else if (dir != null && dir.isExists()) {
                    ze.setTime(dir.getLastModified() + millisToAdd);
                } else {
                    ze.setTime(System.currentTimeMillis() + millisToAdd);
                }
                ze.setSize(0);
                ze.setMethod(ZipEntry.STORED);
                // This is faintly ridiculous:
                ze.setCrc(EMPTY_CRC);
                ze.setUnixMode(mode);

                if (extra != null) {
                    ze.setExtraFields(extra);
                }

                zOut.putNextEntry(ze);
            }
        }

        /*
         * This is a hacky construct to extend the zipFile method to
         * support a new parameter (extra fields to preserve) without
         * breaking subclasses that override the old method signature.
         */
        private static final ThreadLocal<ZipExtraField[]> CURRENT_ZIP_EXTRA = new ThreadLocal<>();
```

Flow Framework™



Technical Debt in Practice

What It Is

Why It Matters

Identifying TD

➔ **Managing TD**

Avoiding TD

24 | Manage

Technical Debt Item: an issue tracker tag or label identifying incurred debt

Risk registers: how risky is the design & how committed are we to that choice?

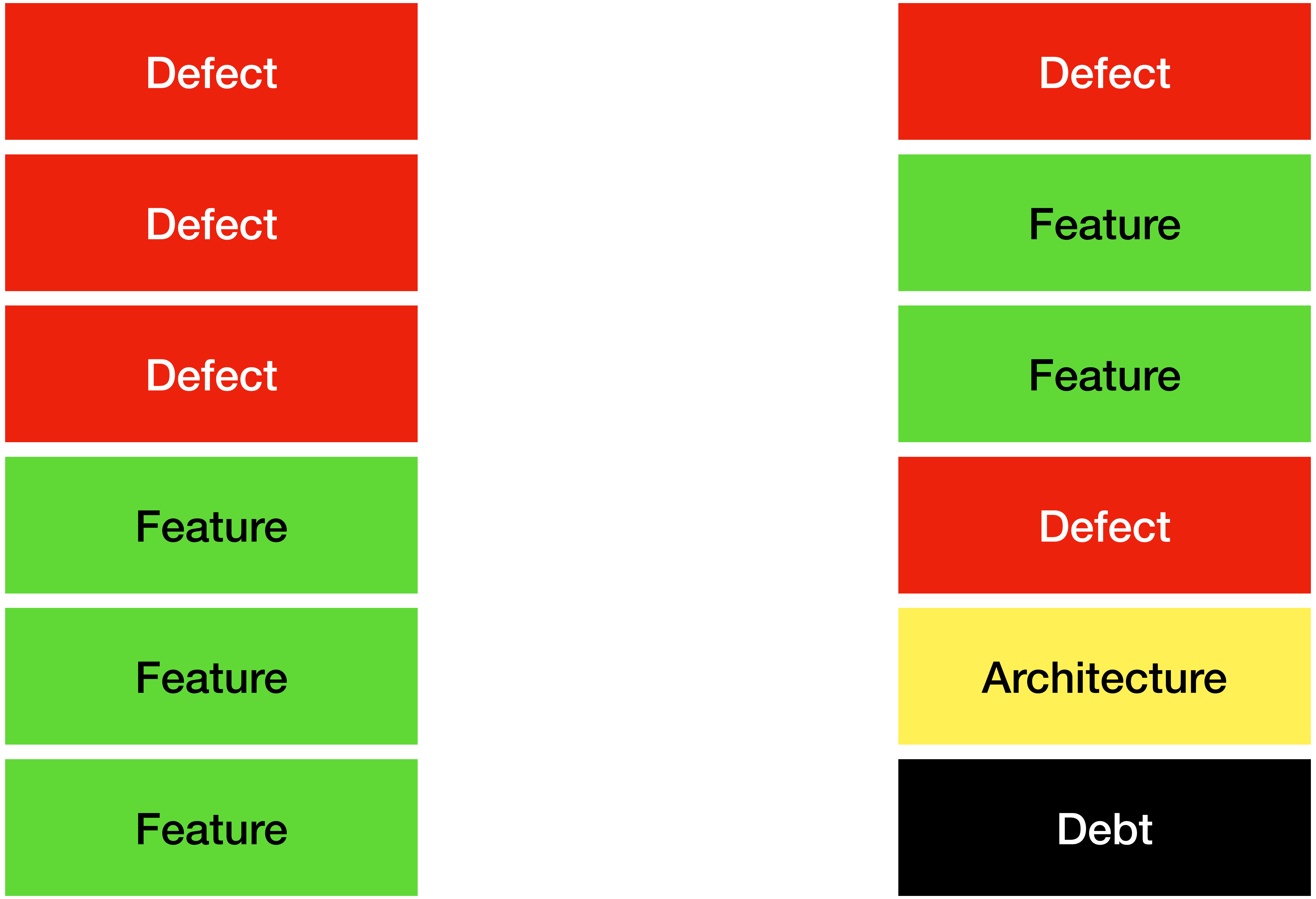
Metrics: MTTR, Cycle time (feature delivery), Risk exposure (trends)

Budget: Make the case for TD time: efficiency, developer satisfaction, actual costs

	Visible	Invisible
Positive Value	Visible Feature	Hidden, architectural feature
Negative Value	Visible defect	Technical debt

Kruchten, P. 2009. *What colour is your backlog?* Agile Vancouver Conference. http://pkruchten.wordpress.com/Talks.

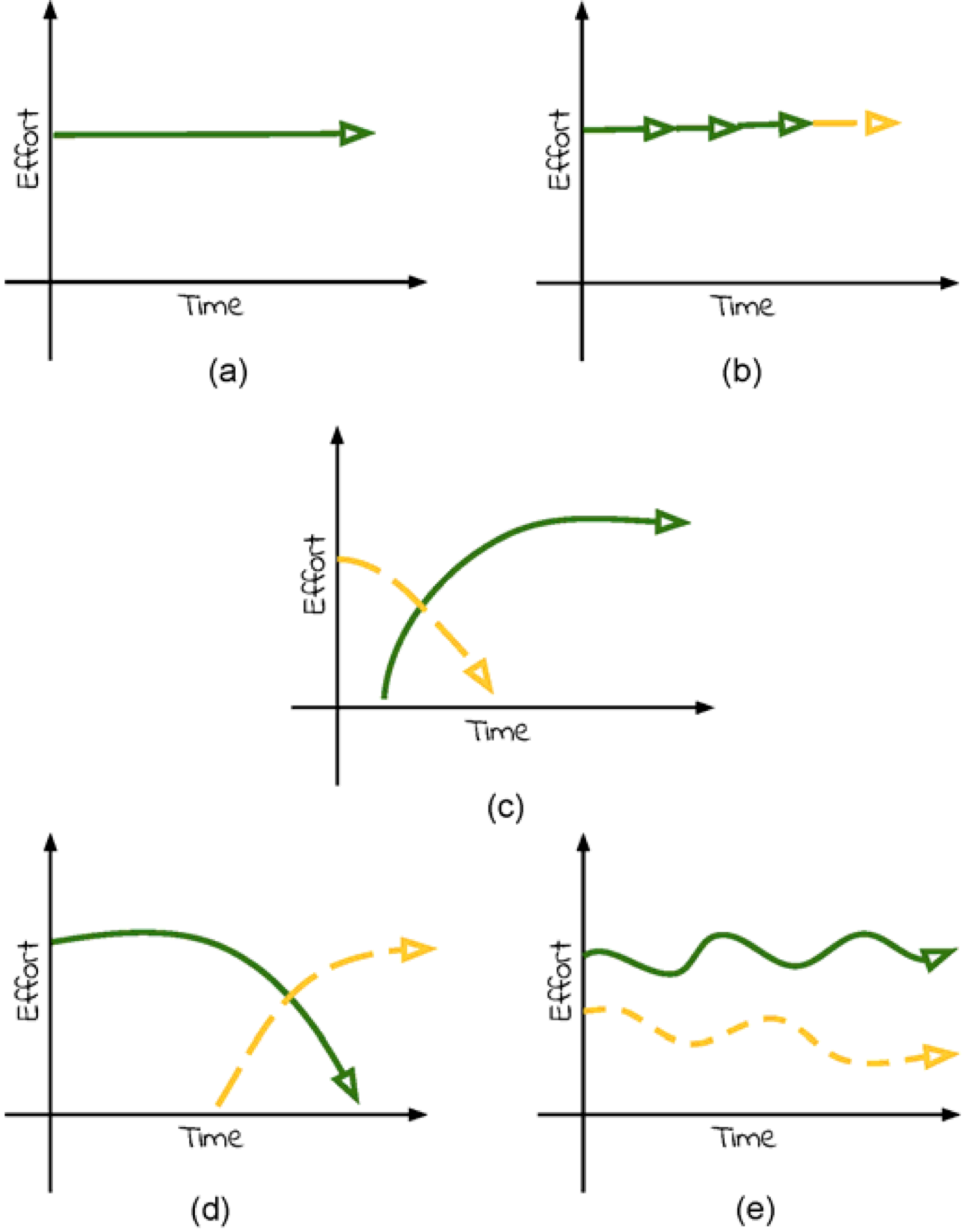
Backlogs


Defect
Defect
Defect
Feature
Feature
Feature
Defect
Feature
Feature
Defect
Architecture
Debt

Iterative Patterns

Green = dev work
Yellow = Arch/TD work

(a) YAGNI
(b) Hardening
(c) Iteration Zero
(d) Rework
(e) Runway (SAFe)



Technical Debt in Practice

What It Is

Why It Matters

Identifying TD

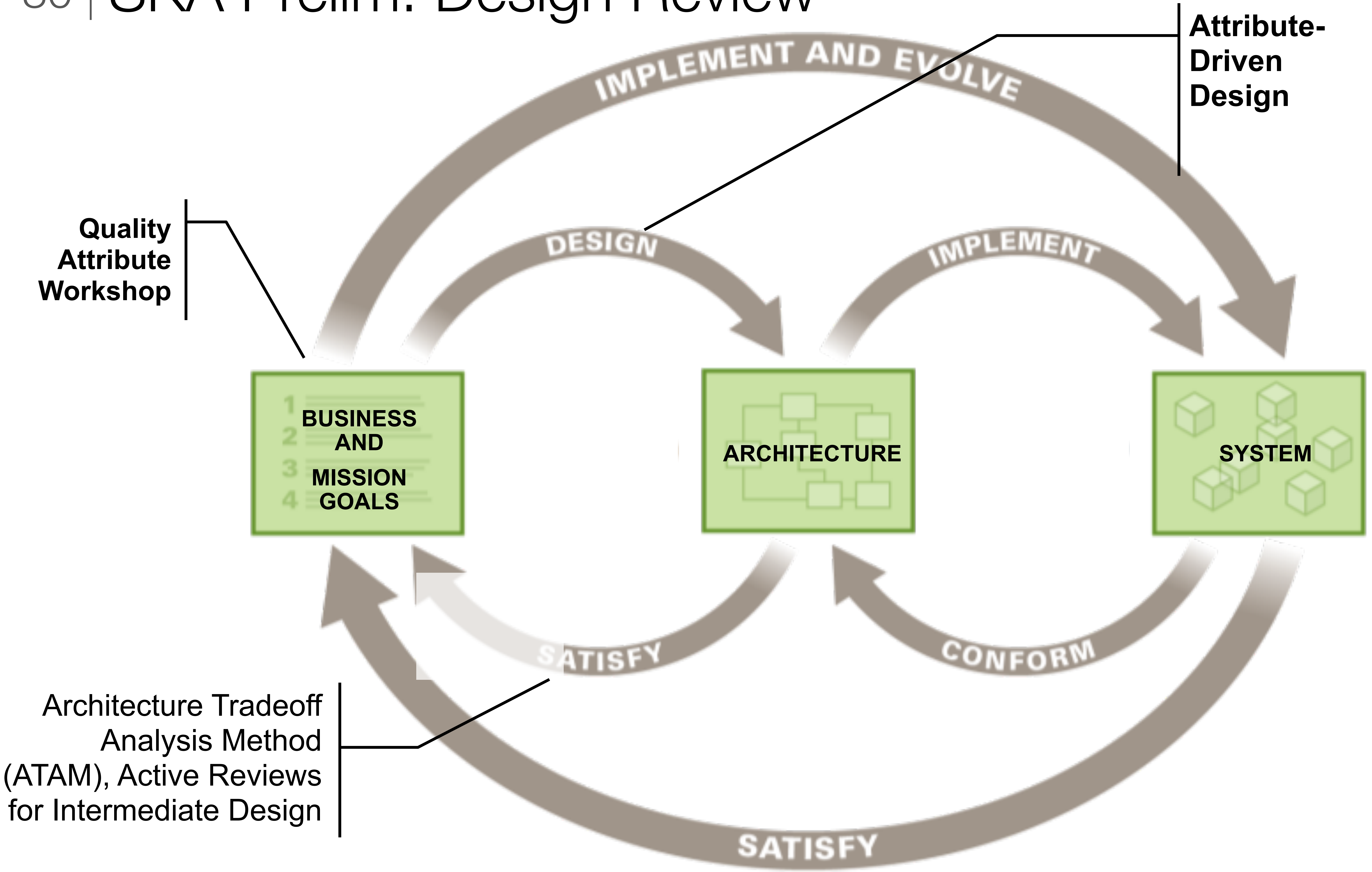
Managing TD

➜ **Avoiding TD**

Future-Proofing Approaches

- Modularize for evolution
  - Tradeoff: integration risk
- Modularize for release
  - Tradeoff: duplication
- Defer decisions until Last Responsible Moment
  - Tradeoff: schedule impact, duplicated work
- Evaluate architecture approach regularly with business goal scenarios
  - Tradeoff: cost, process buy-in

SKA Prelim. Design Review



Technical Debt in Research Software

Very rare to see Peer Review of research code (outside large projects)

Most scientists can probably remember at least once when the code made a mistake (Rogoff Excel error)

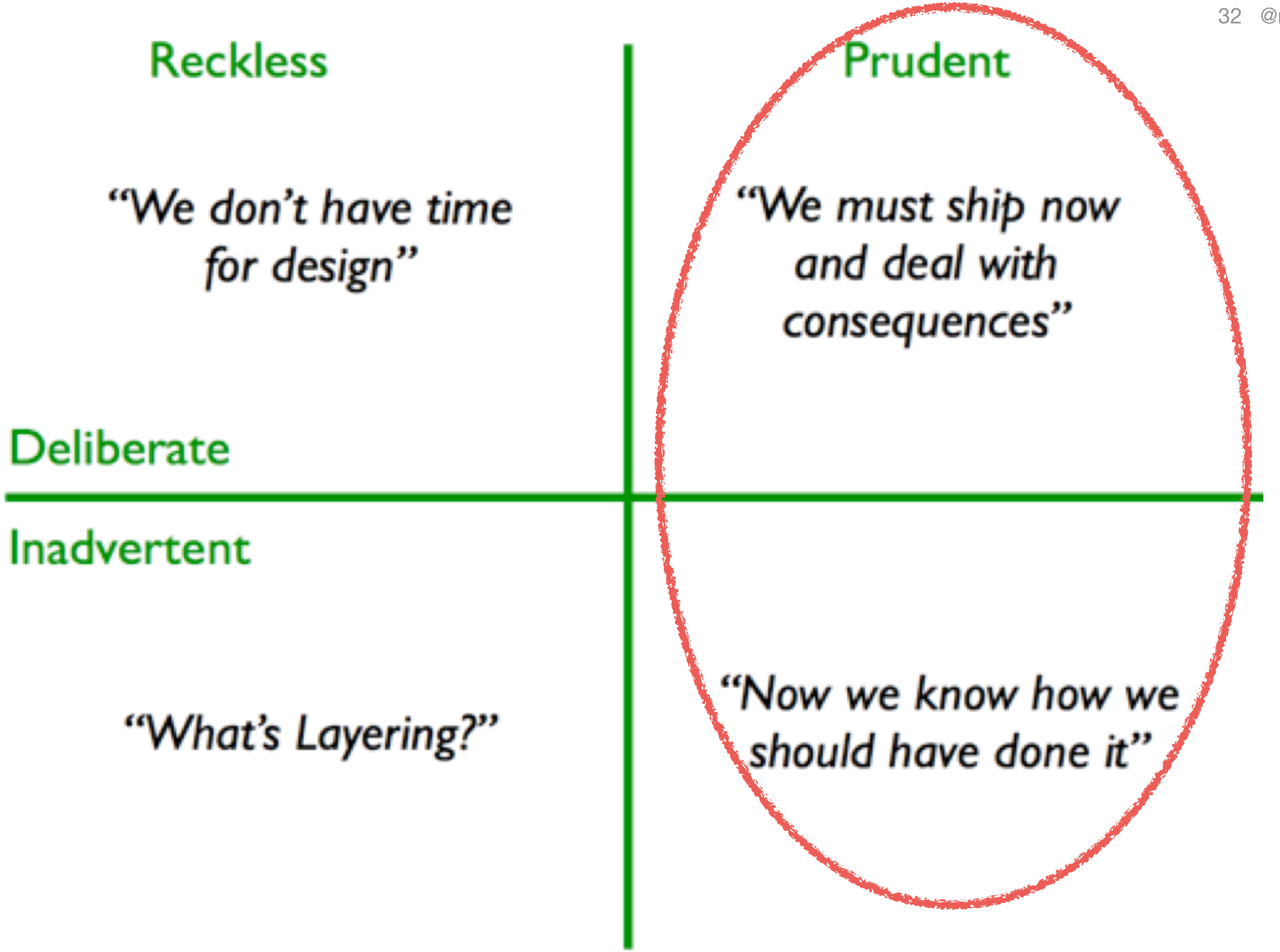
At big data volumes, even supposedly non-core activities—like data storage—can become sources of error, bit rot, etc.

Avoid inadvertent TD:

- Initiatives like Software Carpentry, this conference!, Soc. for Research S/W
- Archival data repositories like Zenodo and Figshare
- Reproducibility efforts

https://theconversation.com/the-reinhart-rogoff-error-or-how-not-to-excel-at-economics-13646

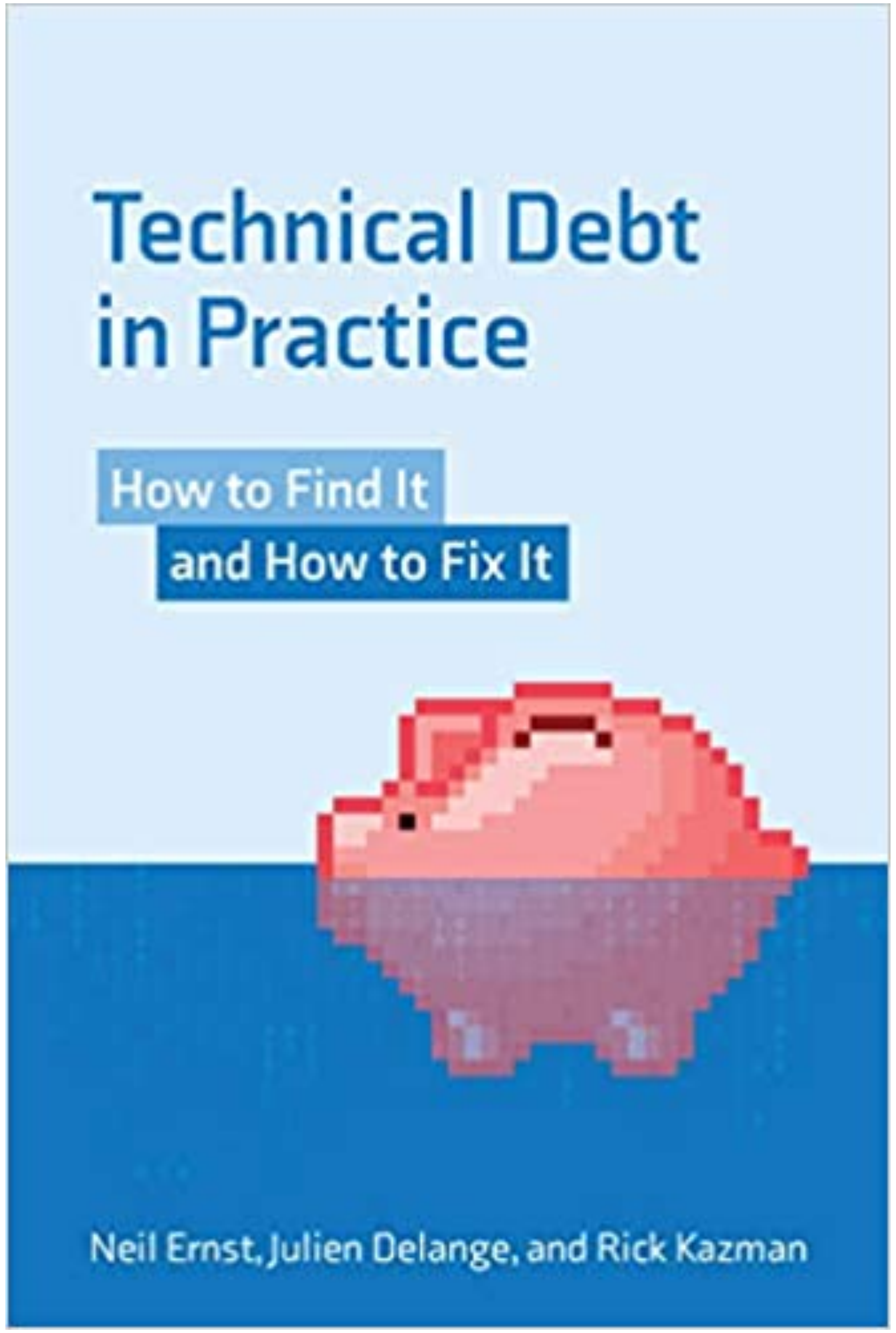
Avoid


Reckless
Prudent
“We don’t have time for design”
“We must ship now and deal with consequences”
Deliberate
Inadvertent
“What’s Layering?”
“Now we know how we should have done it”

Software Moneyball

Software analytics: **identify** key attributes in delivering software, **measure** delivery against those attributes, **manage** teams to maximize those attributes and **avoid** TD

It has never been easier to automate this!



Neil Ernst
nernst@uvic.ca
@neilernst

New book: Technical Debt in Practice (Aug 2021, MIT Press)