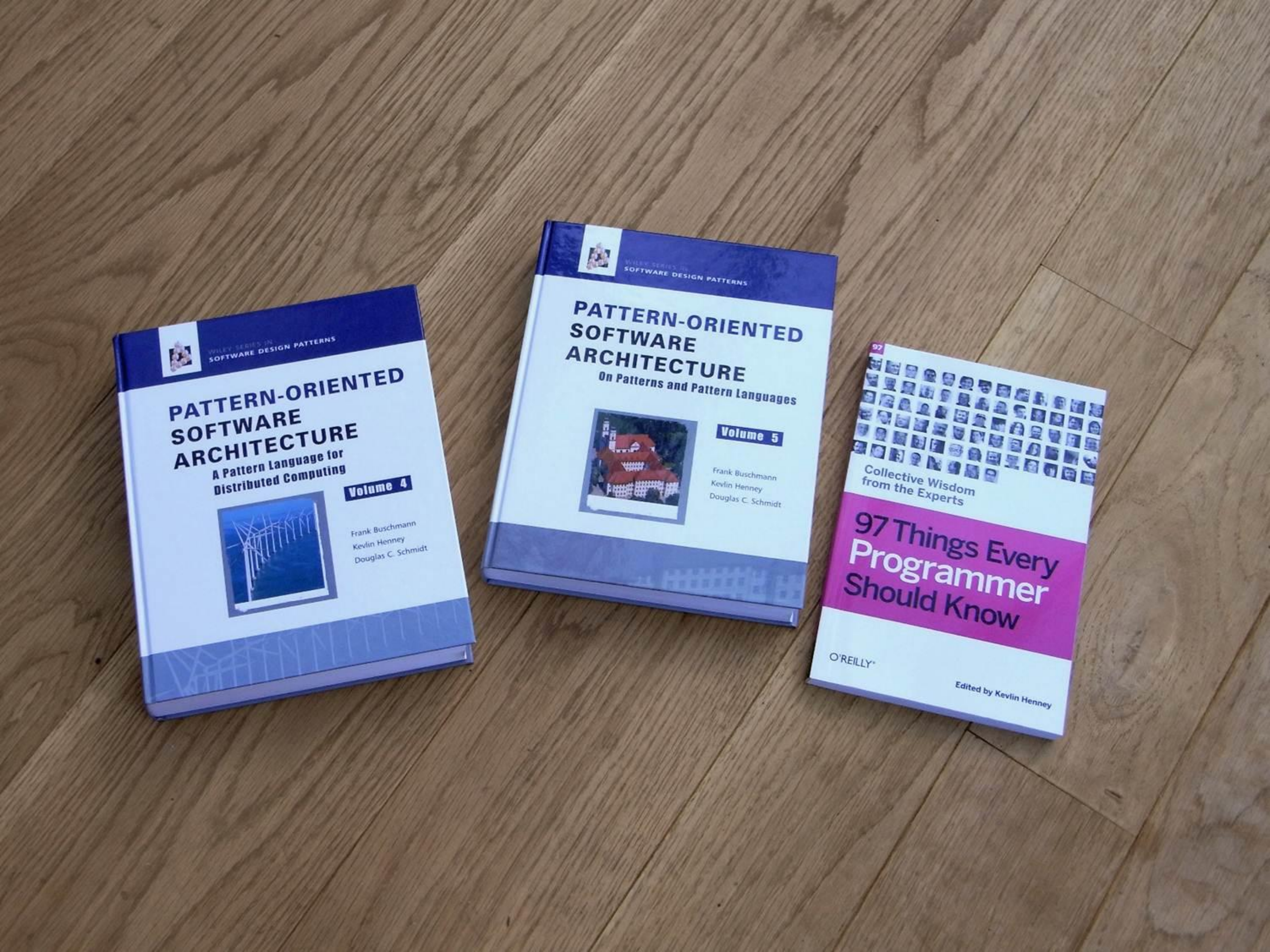
Architecture with Agility

@KevlinHenney

kevlin@curbralan.com


WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
WILEY SERIES IN SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt
97
Collective Wisdom from the Experts
97 Things Every Programmer Should Know
O'REILLY
Edited by Kevlin Henney

If you think good architecture is expensive, try bad architecture.

Brian Foote and Joseph Yoder
Big Ball of Mud

What do we mean by *software architecture*?

In the developed nations, the architect's responsibility is to provide adequate blueprints, work with the construction firm to make sure they are understood, work with building inspectors to validate conformance to the blueprints, then make formal changes to the blueprints as new requirements are identified, and as construction problems are discovered. No shortcuts in this workflow would be excused or accepted.

This analogy applies precisely to software architecture....

Richard A Demers
"Software Architecture Needs Blueprints"

blueprint as a metaphor for a design or plan is much overworked. If the temptation to use it is irresistible, at least remember that a blueprint is a completed plan, not a preliminary one.

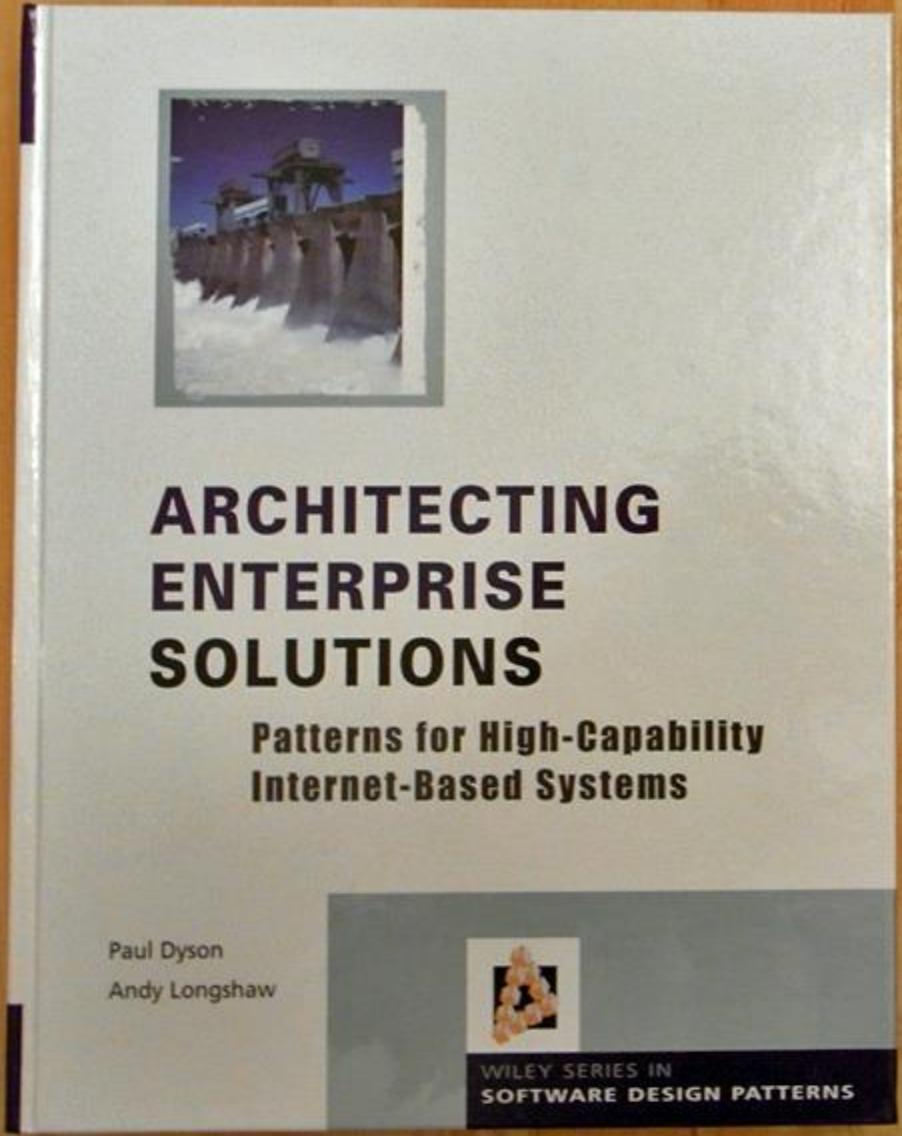
Bill Bryson
Troublesome Words

Architecture is the decisions that you wish you could get right early in a project, but that you are not necessarily more likely to get them right than any other.

Ralph Johnson

All architecture is design but not all design is architecture. Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.

Grady Booch


ARCHITECTING ENTERPRISE SOLUTIONS

Patterns for High-Capability Internet-Based Systems

Paul Dyson

Andy Longshaw

WILEY SERIES IN SOFTWARE DESIGN PATTERNS

Our position is that an architectural definition is something that answers three questions:

- What are the structural elements of the system?
- How are they related to each other?
- What are the underlying principles and rationale that guide the answers to the previous two questions?

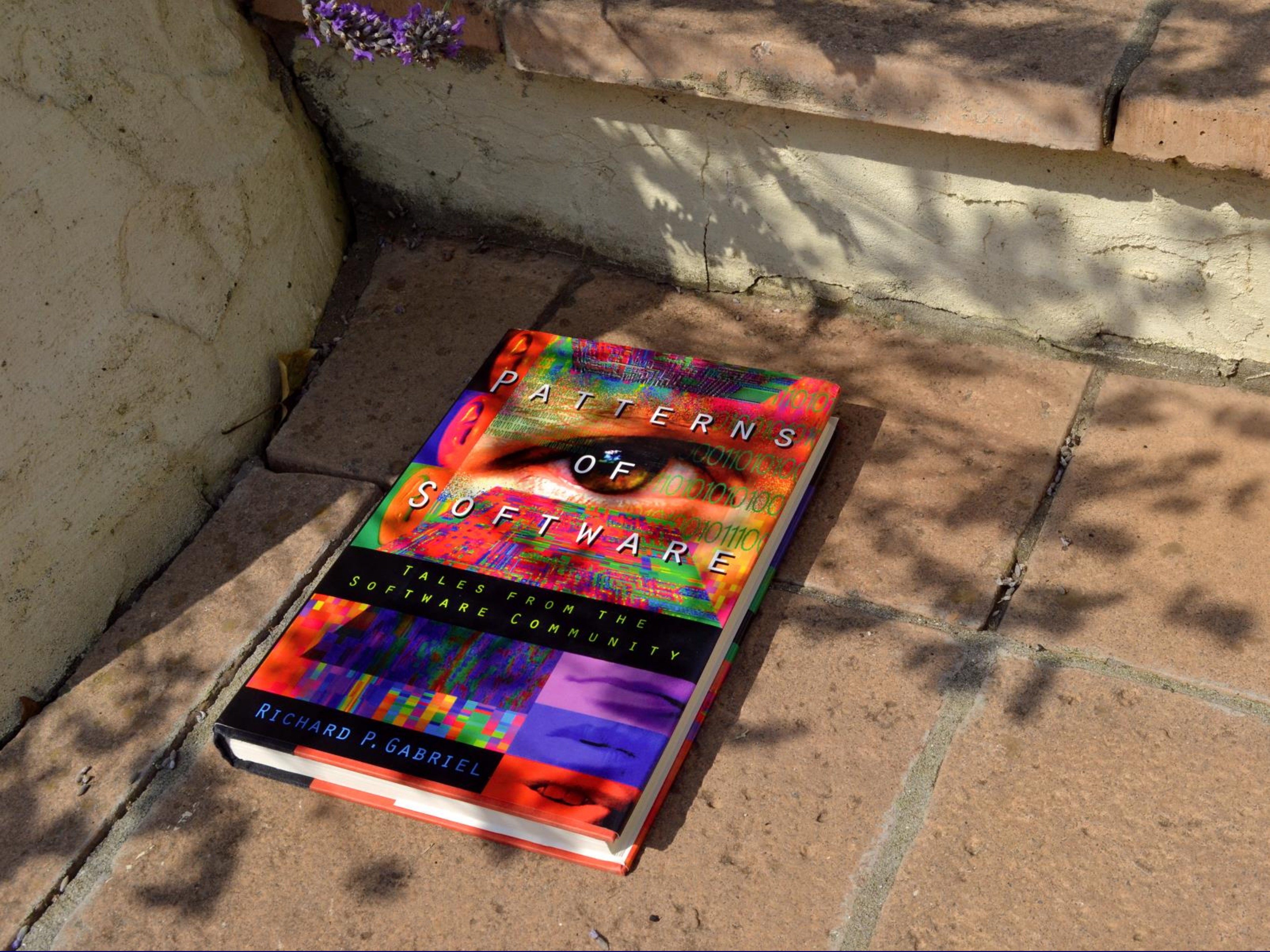
Architecture is a hypothesis, that needs to be proven by implementation and measurement.

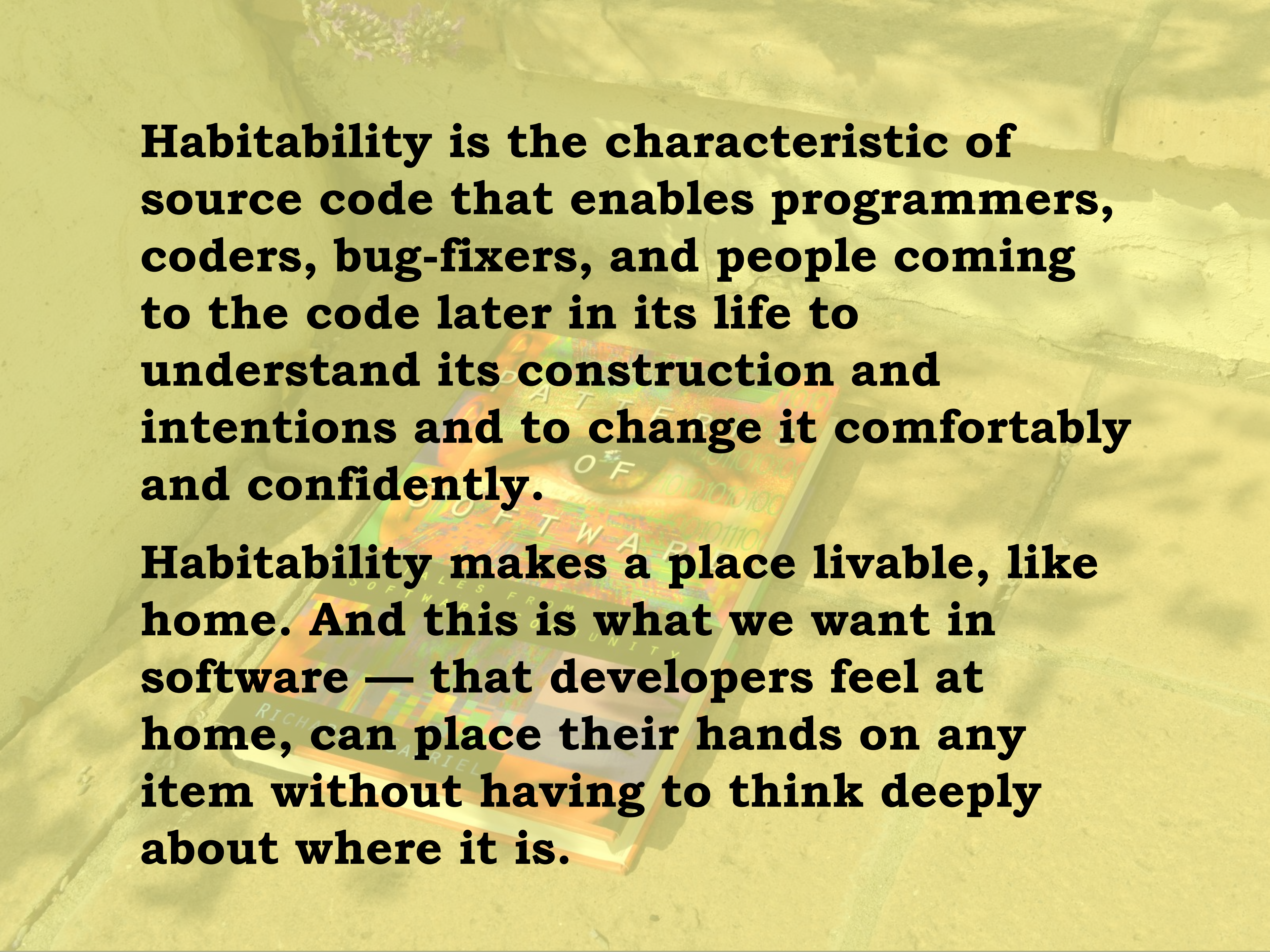
Tom Gilb

empirical, *adjective*

- based on, concerned with, or verifiable by observation or experience rather than theory or pure logic
- pertaining to, or derived from, experience
- capable of being verified or disproved by observation or experiment

Concise Oxford English Dictionary · Oxford English Dictionary · Merriam-Webster's Collegiate Dictionary


PATTERNS OF SOFTWARE
TALES FROM THE SOFTWARE COMMUNITY
RICHARD P. GABRIEL



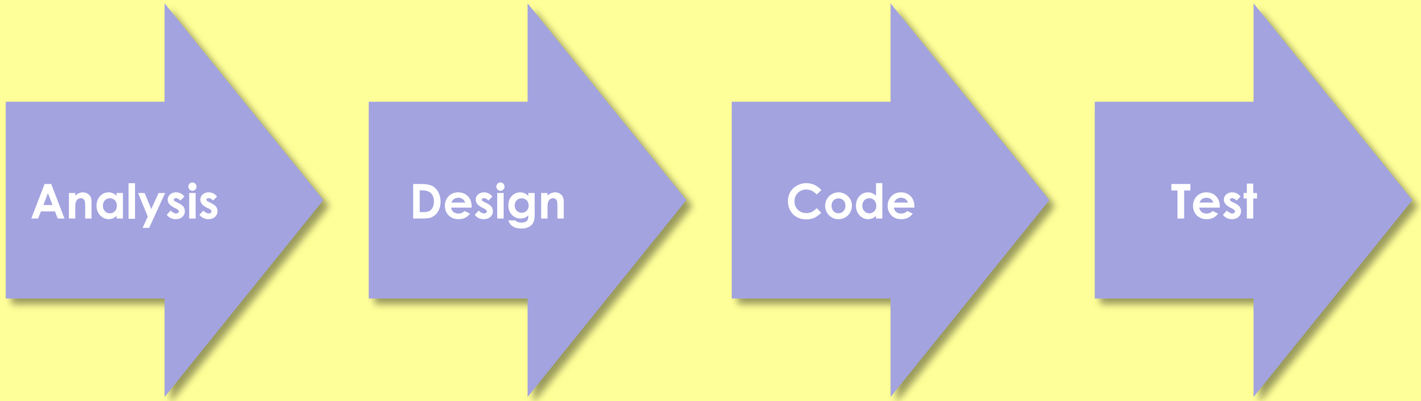
Habitability is the characteristic of source code that enables programmers, coders, bug-fixers, and people coming to the code later in its life to understand its construction and intentions and to change it comfortably and confidently.

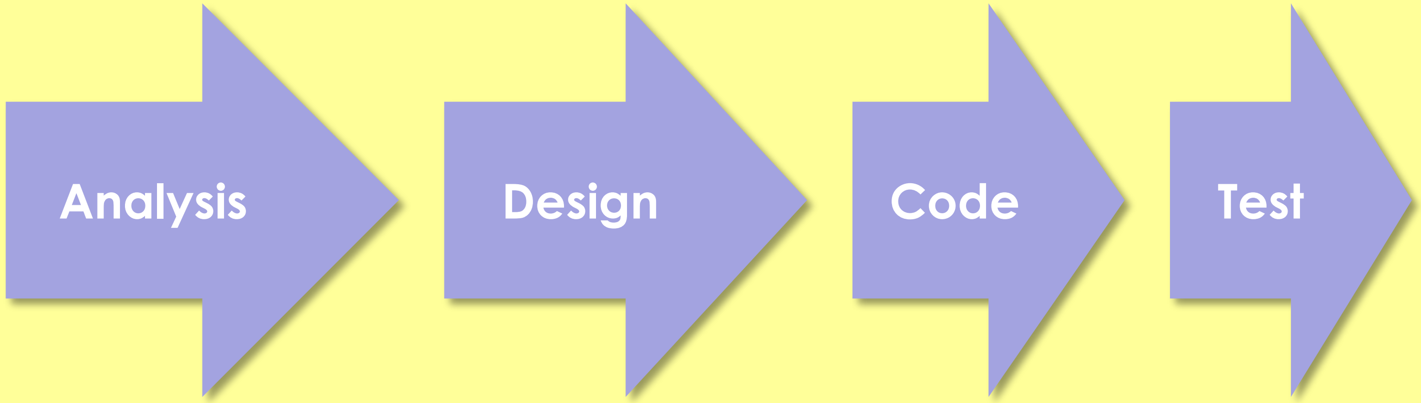
Habitability makes a place livable, like home. And this is what we want in software — that developers feel at home, can place their hands on any item without having to think deeply about where it is.

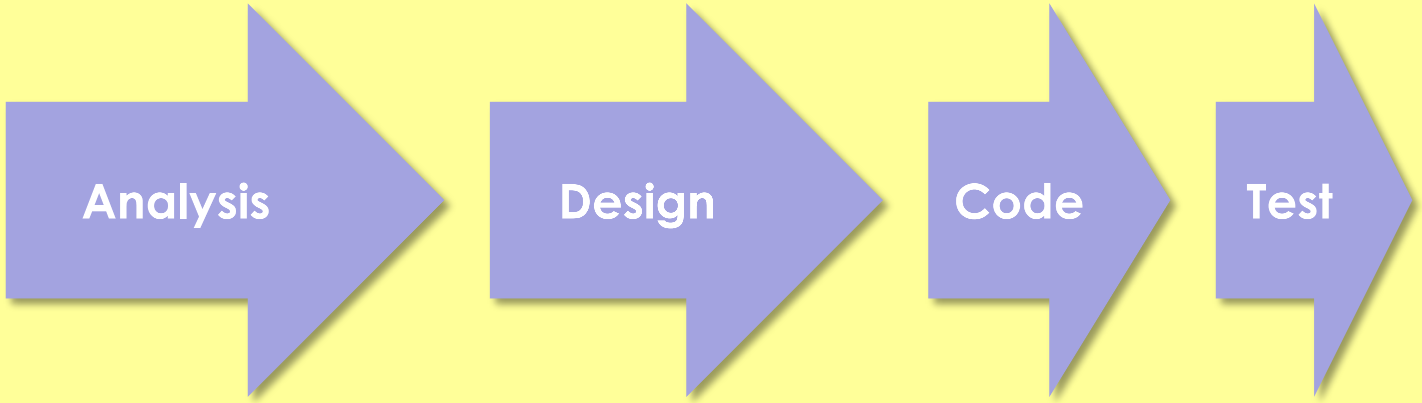
What is the relationship between process and architecture?

Walking on water and developing software from a specification are easy if both are frozen.

Edward V Berard


Analysis
Design
Code
Test


Analysis
Design
Code
Test

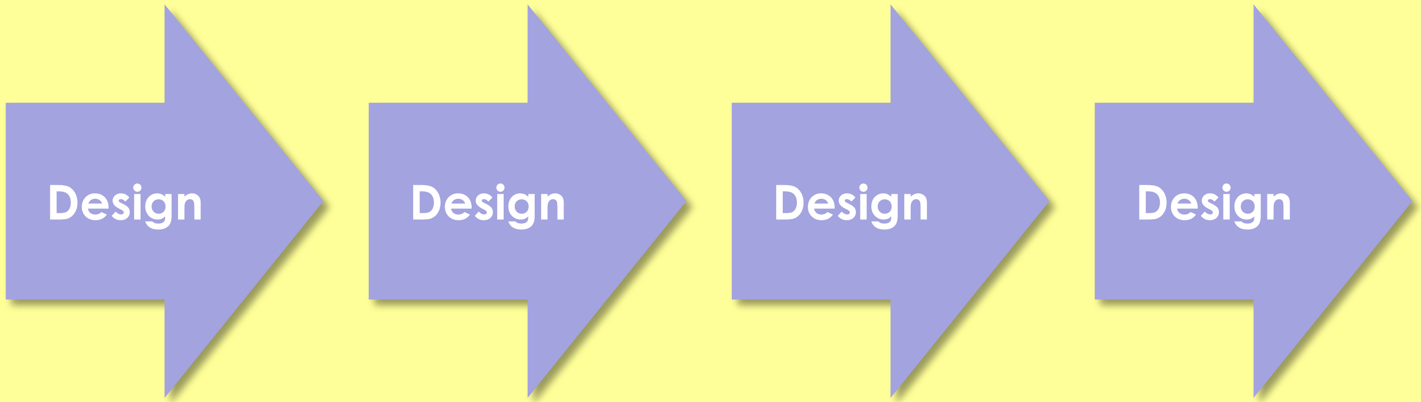

Analysis
Design
Code
Test

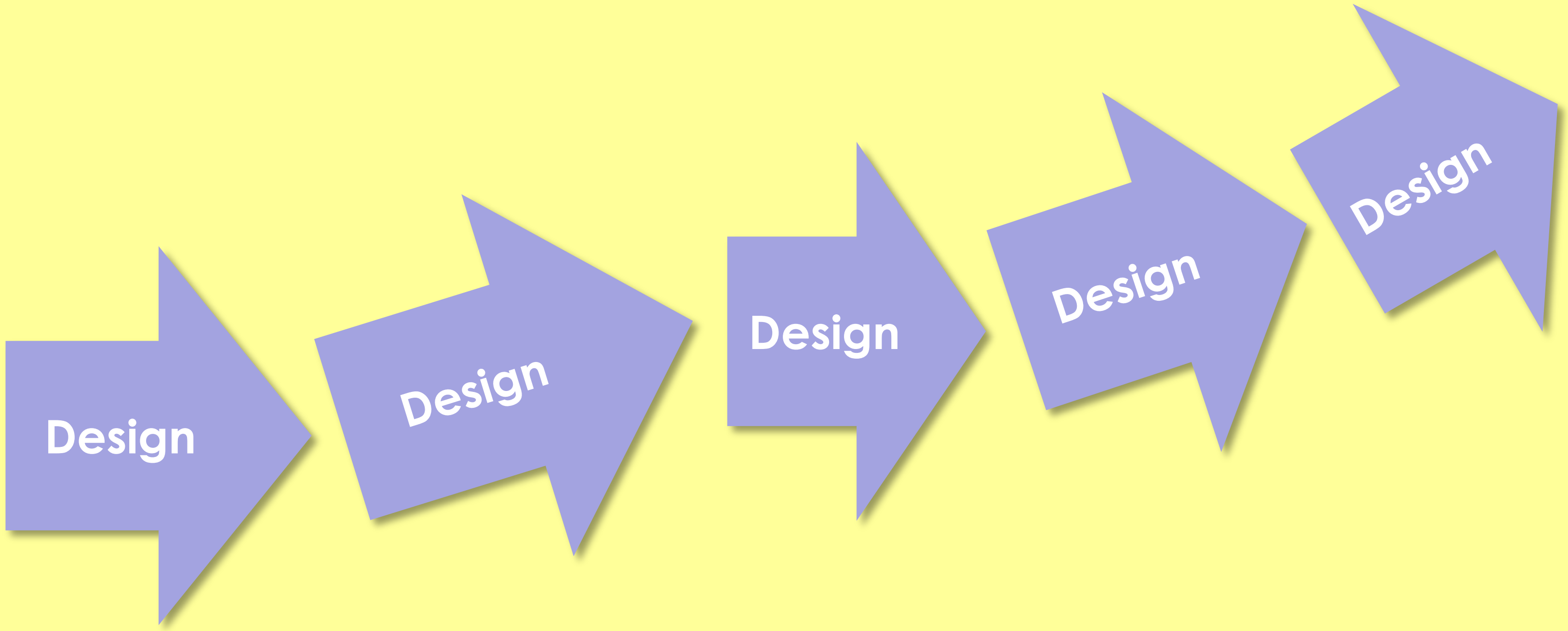
The "defined" process control model requires that every piece of work be completely understood. Given a well-defined set of inputs, the same outputs are generated every time.

Ken Schwaber
Agile Software Development with Scrum

The empirical process control model, on the other hand, expects the unexpected. It provides and exercises control through frequent inspection and adaptation for processes that are imperfectly defined and generate unpredictable and unrepeatable results.

Ken Schwaber
Agile Software Development with Scrum


Design
Design
Design
Design


Design
Design
Design
Design
Design

101 Things I Learned
in Architecture School
Matthew Frederick

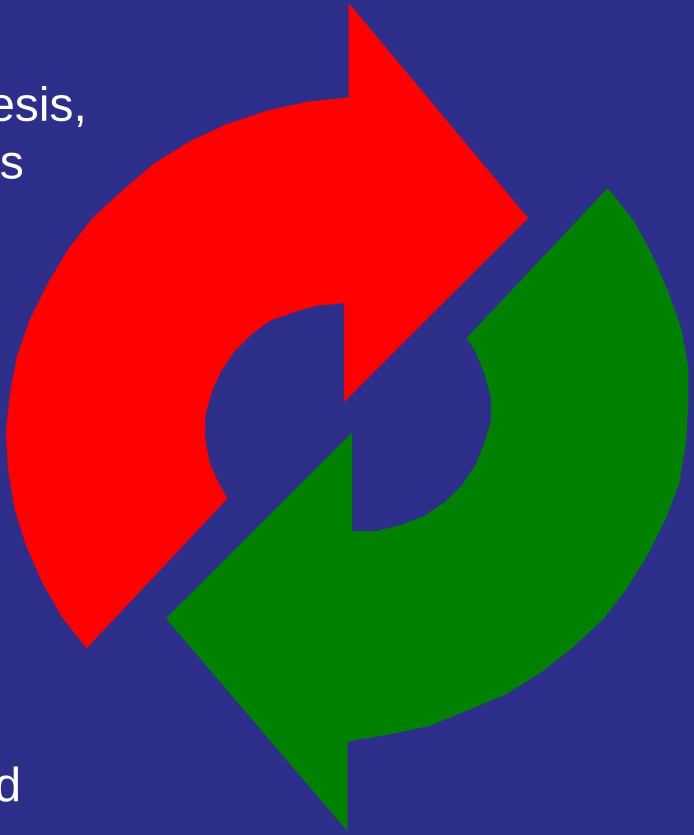
Properly gaining control of the design process tends to feel like one is *losing* control of the design process.

Plan

Establish hypothesis, goal or work tasks

Do

Carry out plan



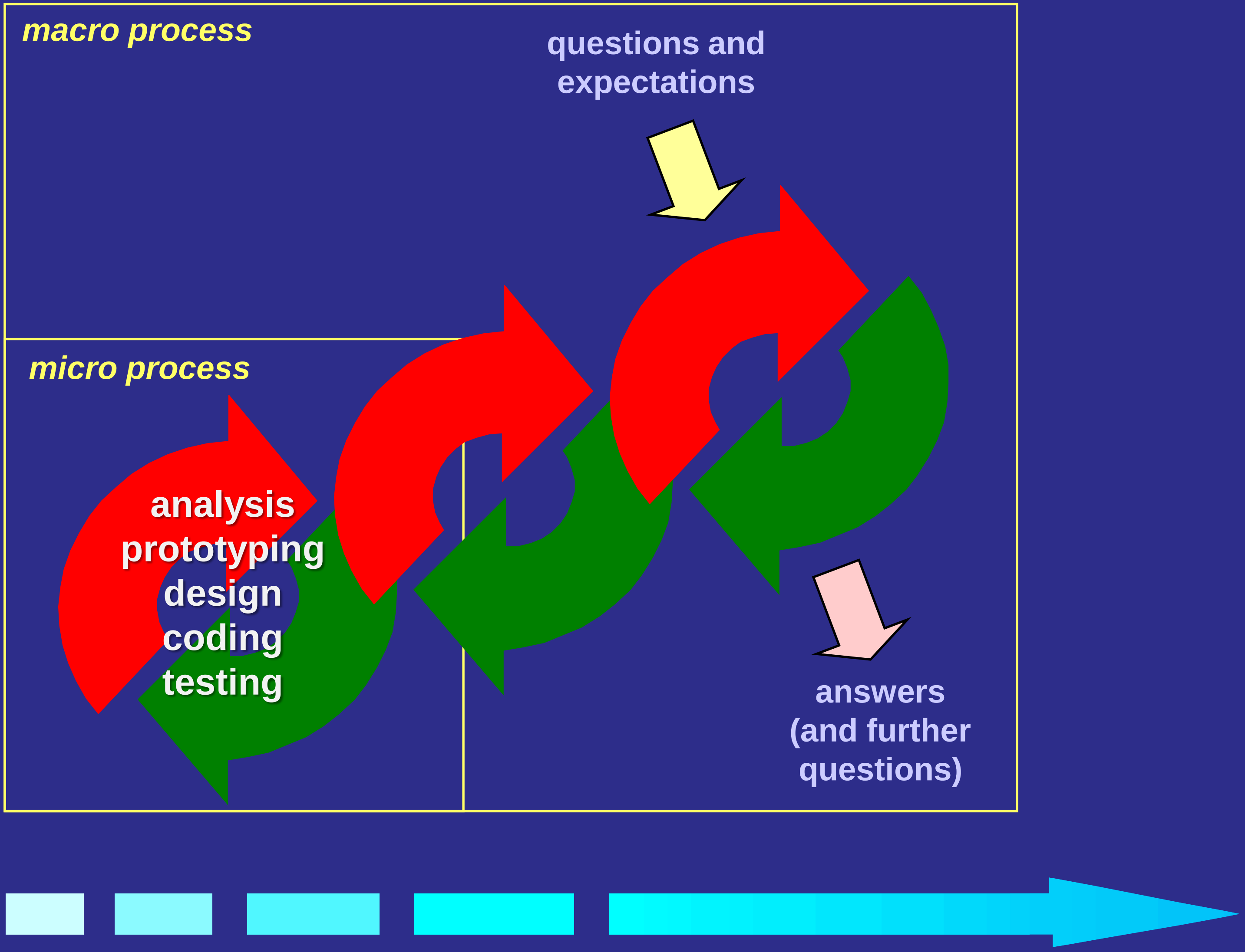
Act

Revise approach or artefacts based on study.

Study

Review what has been done against plan (a.k.a. *Check*).

Deming's PDSA Cycle


macro process
questions and expectations
micro process
analysis
prototyping
design
coding
testing
answers (and further questions)

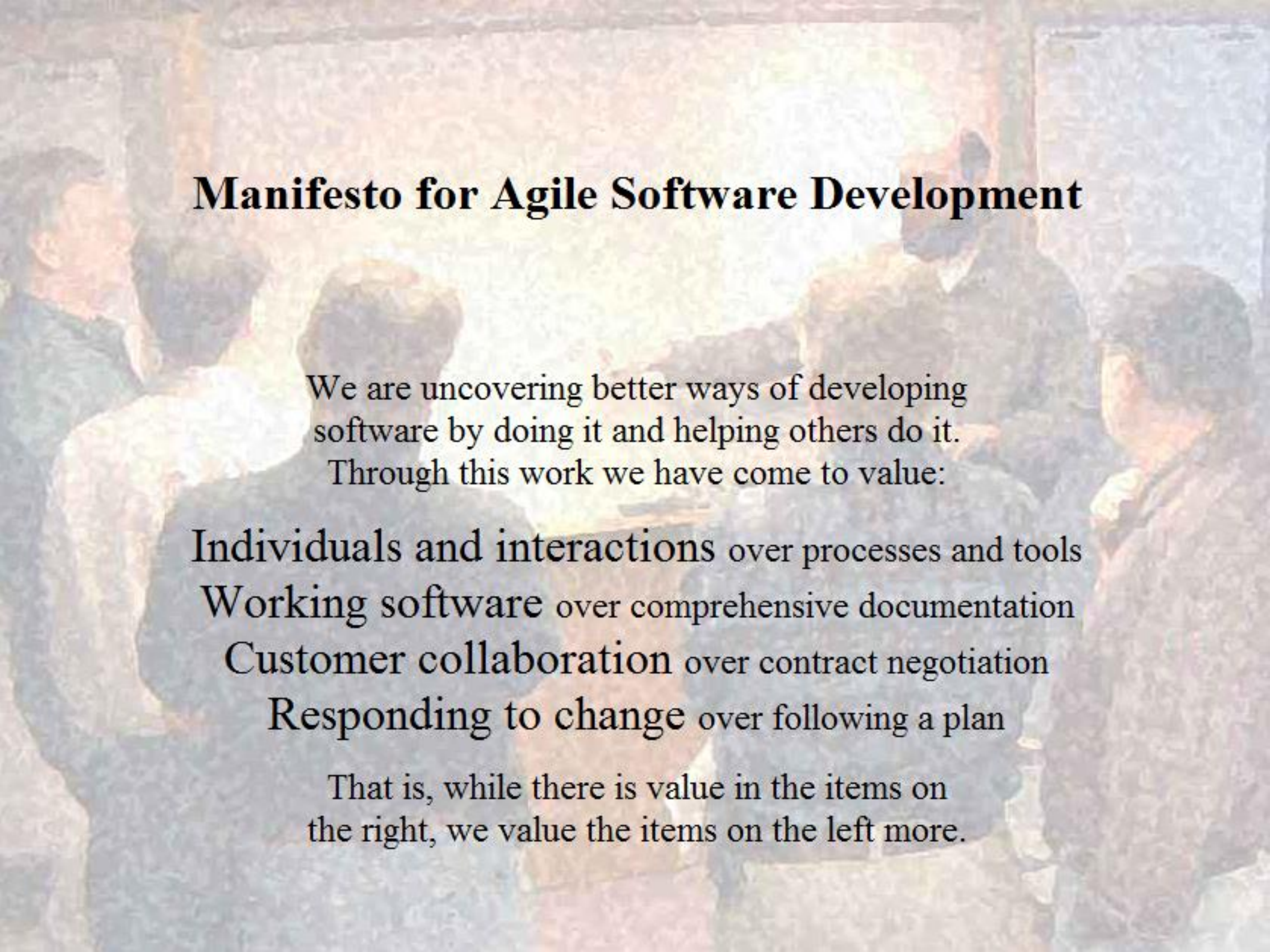
nomic, *noun & adjective*

- a game in which changing the rules of the game is a legal move and part of the game
- the original Nomic was invented by Peter Suber, but the term is now generalised to describe any game that has these properties
- political constitutions, legal systems, software development processes and many games that children spontaneously evolve over an afternoon of play are nomic in nature

agile, *adjective*

- able to move quickly and easily
- having the faculty of quick motion
- easily moved
- nimble, active, ready
- having a quick resourceful and adaptable character

Concise Oxford English Dictionary · Oxford English Dictionary · Merriam-Webster's Collegiate Dictionary



Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

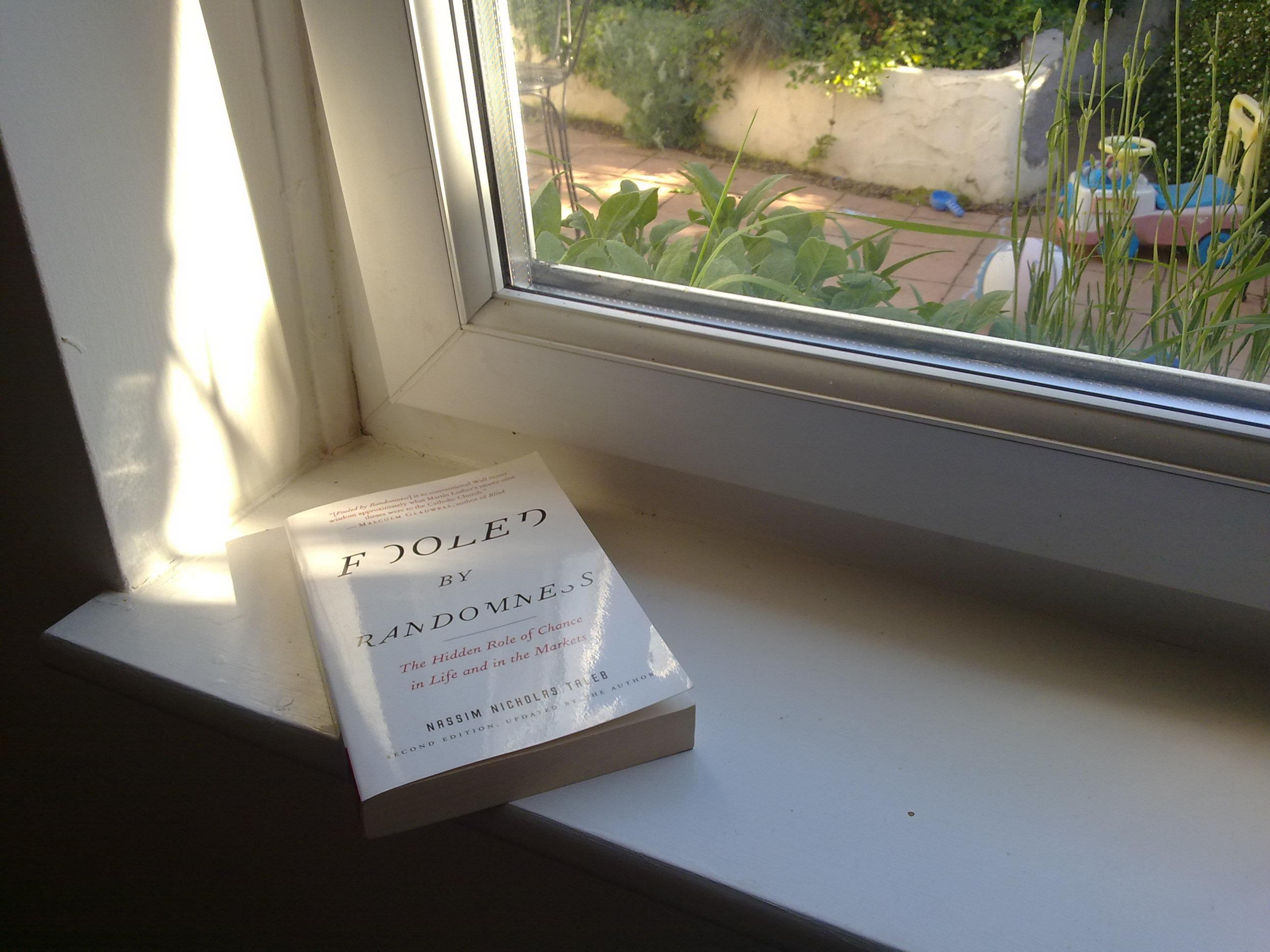
Agile methods balance two things. One is the maximizing of value creation. The other thing is the maximizing of the chances of actually delivering something.

These two goals are sometimes in conflict!

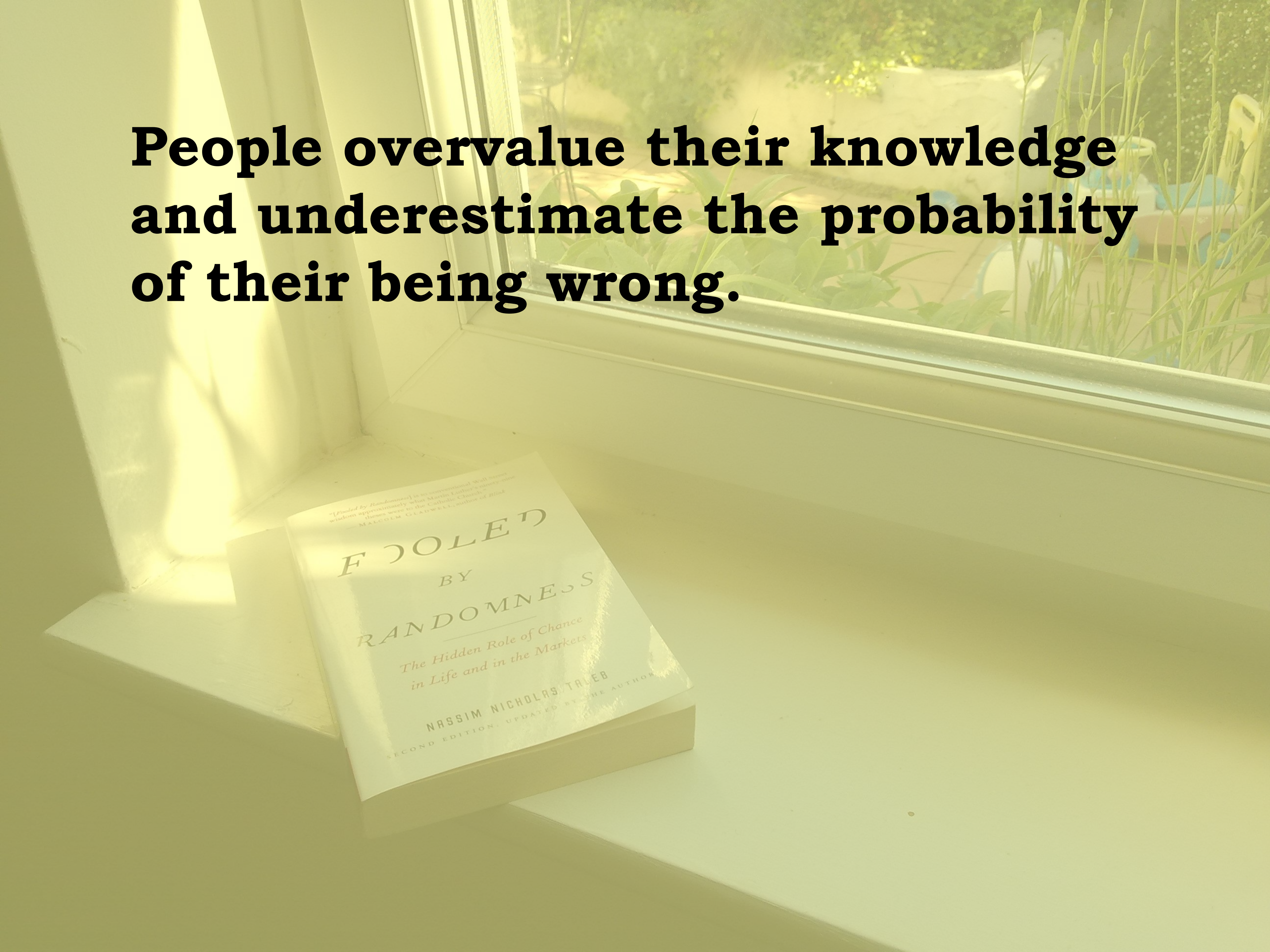
Projects that dogmatically focus on stakeholder value are working on the right things but still risk failing completely.

The simple reason agile focuses on "working software" is that this is one of the primary ways of insuring that the system being worked on will actually work.

Niklas Bjørnerstedt
http://www.leanway.no/?p=280


"[Fooled by Randomness] is to conventional Wall Street wisdom approximately what Martin Luther's ninety-nine theses were to the Catholic Church."
—Malcolm Gladwell, author of Blink
FOOLED BY RANDOMNESS
The Hidden Role of Chance in Life and in the Markets
NASSIM NICHOLAS TALEB
SECOND EDITION, UPDATED BY THE AUTHOR

People overvalue their knowledge and underestimate the probability of their being wrong.


FOOLED BY RANDOMNESS
The Hidden Role of Chance in Life and in the Markets
NASSIM NICHOLAS TALEB
SECOND EDITION, UPDATED BY THE AUTHOR

0. Lack of Ignorance
1. Lack of Knowledge
2. Lack of Awareness
3. Lack of Process
4. Meta-Ignorance

Five Orders of Ignorance
Phillip G Armour

- What you can build is influenced and constrained by how you build it...
- And vice versa
- Architectural thinking is based on knowledge, which requires learning
- Learning occurs throughout a software development project
- Making all the significant decisions up front is not responsible
- Sustainable agility requires good architecture; fast initial development does not — these are often confused

What are some properties of a good architecture?

Firmitas

Utilitas

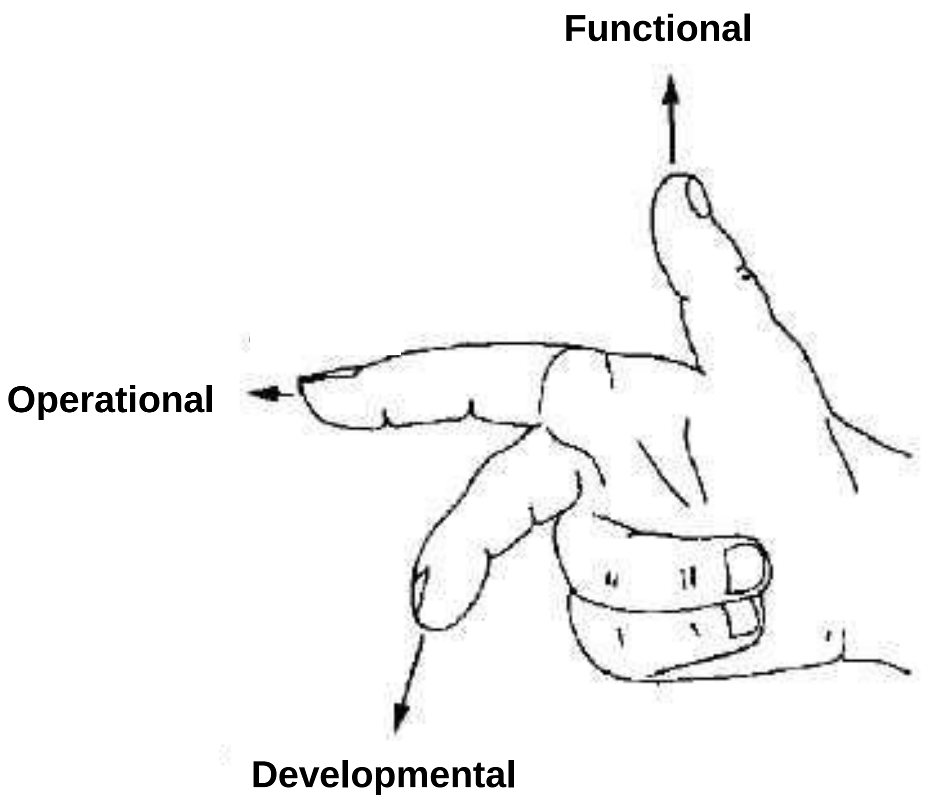
Venustas

Requirements come in many possible flavours, but are commonly cast into two categories: functional and non-functional requirements. As a label, it has to be admitted that *non-functional* is fairly lame. It is unhelpfully vague and amusingly ambiguous.

Most things that are non-functional don't work: washing machines, cars and programs that are non-functional are broken. Also, by prefixing *functional requirements* with *non*, other requirements seem to be relegated to second- or third-class citizenship.

Requirements can be better and more fairly considered under the headings of functional requirements, operational requirements and developmental requirements.

Kevlin Henney
"Inside Requirements"

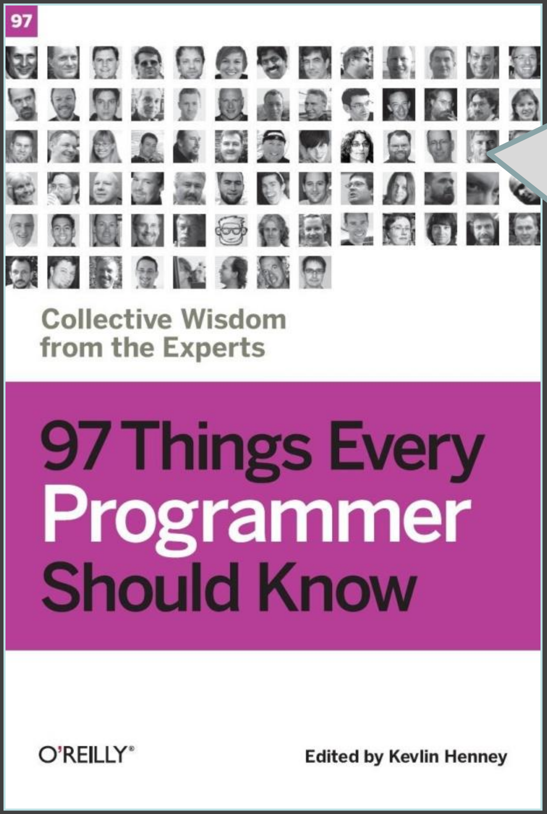

Functional
Operational
Developmental

Computer performance is characterised by the amount of useful work accomplished by a computer system compared to the time and resources used.

Depending on the context, good computer performance may involve one or more of the following:

- Short response time for a given piece of work
- High throughput (rate of processing work)
- Low utilization of computing resource(s)
- High availability of the computing system or application

http://en.wikipedia.org/wiki/Computer_performance


97
Collective Wisdom from the Experts
97 Things Every Programmer Should Know
O'REILLY
Edited by Kevlin Henney

More often than not, performance tuning a system requires you to alter code. When we need to alter code, every chunk that is overly complex or highly coupled is a dirty code bomb lying in wait to derail the effort. The first casualty of dirty code will be your schedule.

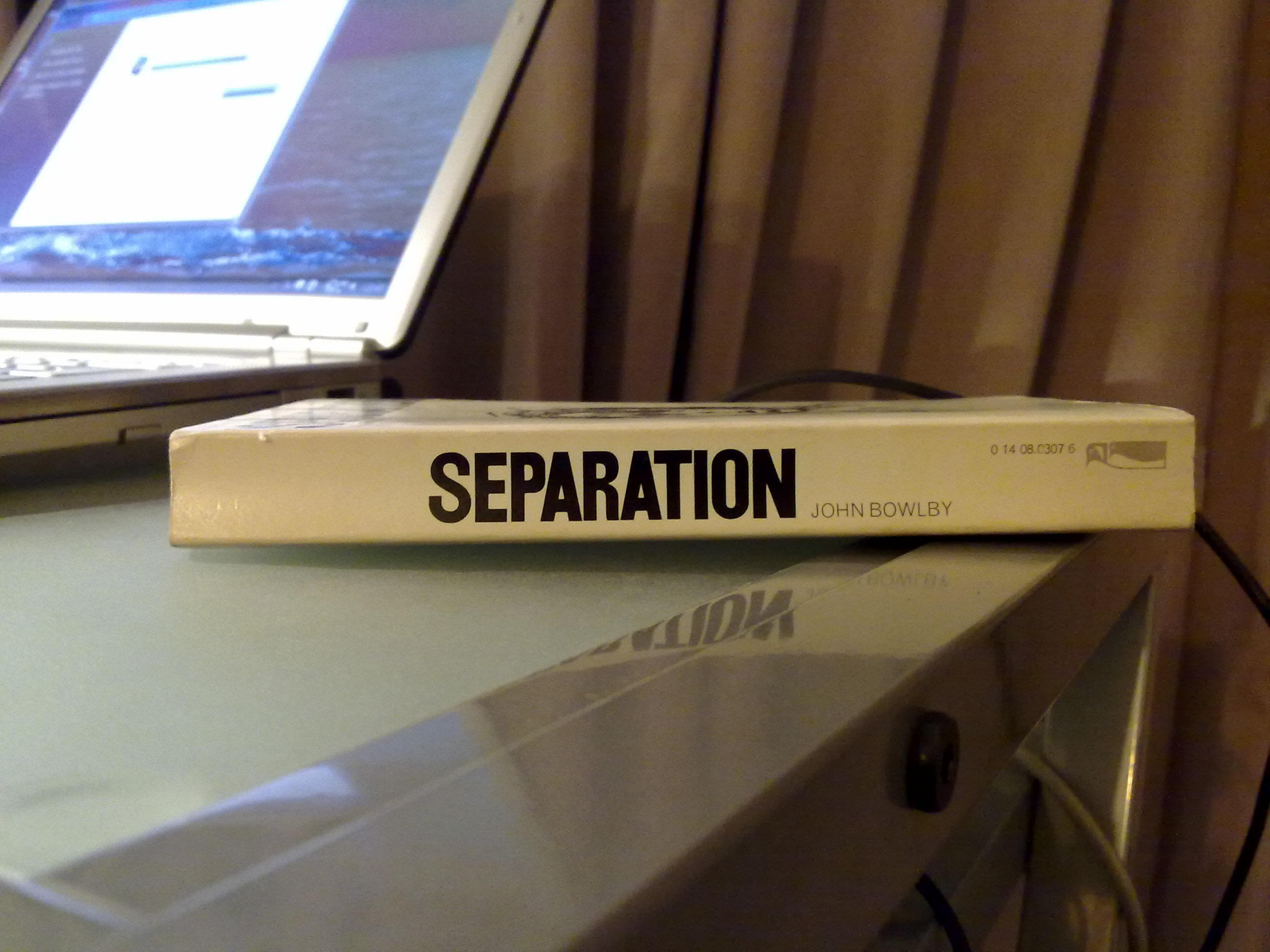
Kirk Pepperdine
"The Road to Performance is Littered with Dirty Code Bombs"

There are standard precautions that can help reduce risk in complex software systems. This includes the definition of a good software architecture based on a clean separation of concerns, data hiding, modularity, well-defined interfaces, and strong fault-protection mechanisms.

Gerard J Holzmann
"Mars Code", CACM 57(2)
http://cacm.acm.org/magazines/2014/2/171689-mars-code/fulltext

The connections between modules are the assumptions which the modules make about each other.

David L Parnas


SEPARATION
JOHN BOWLBY
0 14 08.0307 6

The basic thesis [...] is that organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.

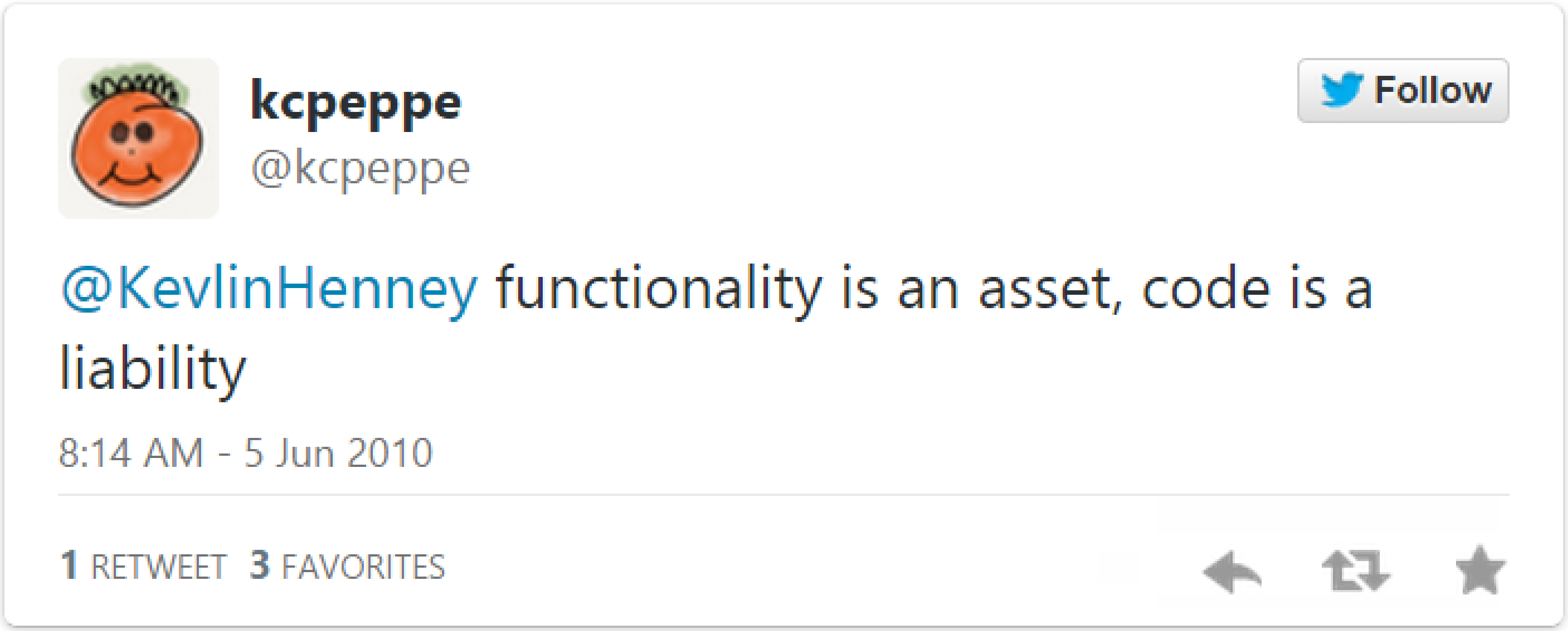
Melvin Conway
How Do Committees Invent?

We have seen that this fact has important implications for the management of system design. [...] A design effort should be organized according to the need for communication.

Melvin Conway
How Do Committees Invent?

Because the design that occurs first is almost never the best possible, the prevailing system concept may need to change. Therefore, flexibility of organization is important to effective design.

Fred Brooks



kcppeppe
@kcpeppe

@KevlinHenney functionality is an asset, code is a liability

8:14 AM - 5 Jun 2010

1 RETWEET 3 FAVORITES

https://twitter.com/kcpeppe/status/15473004648

Everybody knows that TDD stands for Test Driven Development. However, people too often concentrate on the words "Test" and "Development" and don't consider what the word "Driven" really implies. For tests to drive development they must do more than just test that code performs its required functionality: they must clearly express that required functionality to the reader. That is, they must be clear specifications of the required functionality. Tests that are not written with their role as specifications in mind can be very confusing to read.

Nat Pryce and Steve Freeman
"Are Your Tests Really Driving Your Development?"

The difficulty in being able to write a test can be boiled down to the two broad themes of complexity and ignorance, each manifested in a couple of different ways:

- The essential complexity of the problem being solved.
- The accidental complexity of the problem being solved.
- Uncertainty over what the code should actually do.
- Lack of testing know-how.

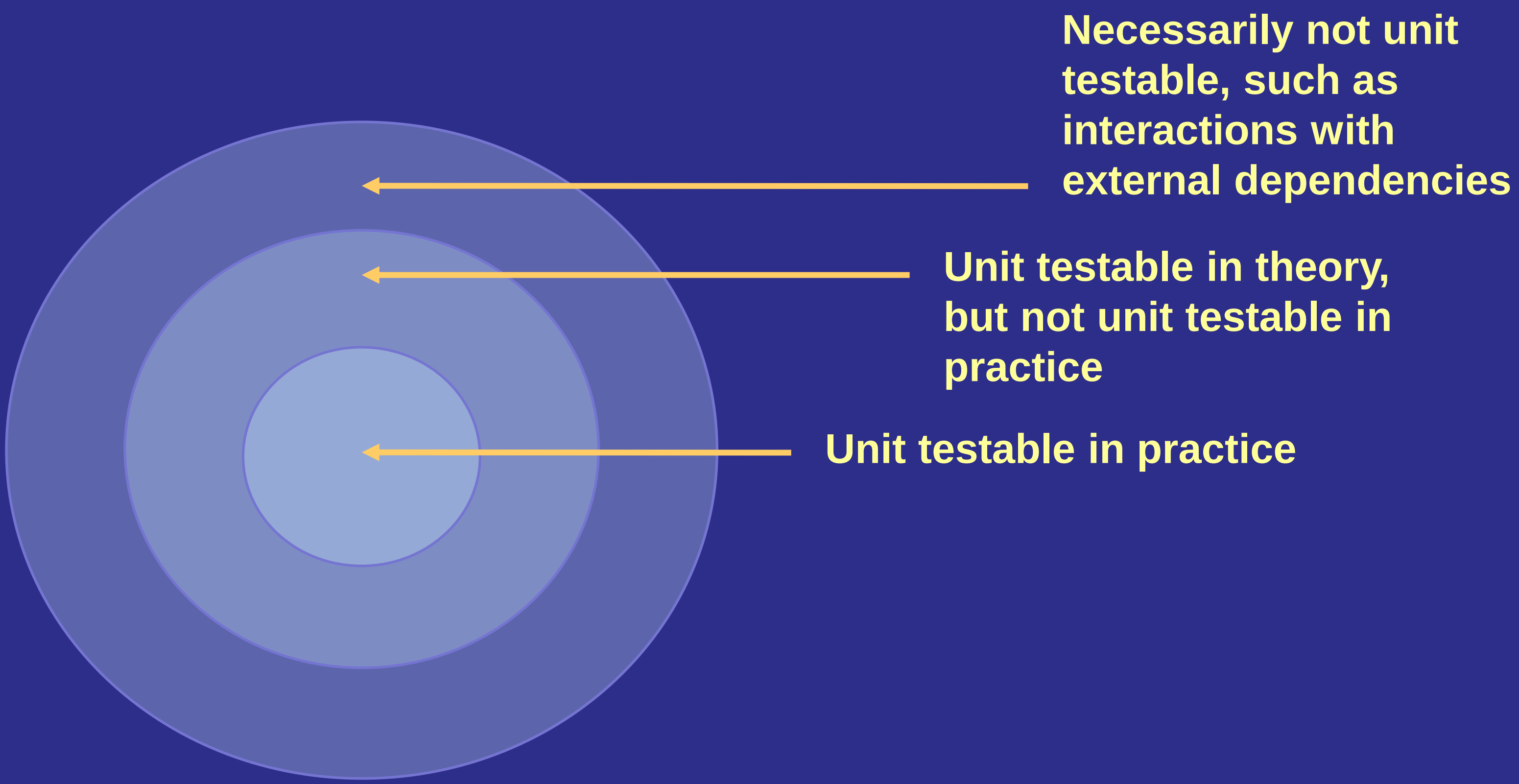
Kevlin Henney
"A Test of Knowledge"
http://www.artima.com/weblogs/viewpost.jsp?thread=340839

A test is not a unit test if:

- **It talks to the database**
- **It communicates across the network**
- **It touches the file system**
- **It can't run at the same time as any of your other unit tests**
- **You have to do special things to your environment (such as editing config files) to run it.**

Tests that do these things aren't bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes.

Michael Feathers
"A Set of Unit Testing Rules"


Necessarily not unit testable, such as interactions with external dependencies
Unit testable in theory, but not unit testable in practice
Unit testable in practice

Sustainable development [...] implies meeting the needs of the present without compromising the ability of future generations to meet their own needs.

Brundtland Report of the World Commission on Environment and Development

- An architecture needs to meet the needs of those who require, who use and who work on the product
- Requirements (and properties) are not simply functional or "non-functional"
- An architecture should be aligned with both its market and its development (and vice versa)
- An architecture should support the changes that it experiences

How can an architecture be evolved and grown?

You have to finish things — that's what you learn from, you learn by finishing things.

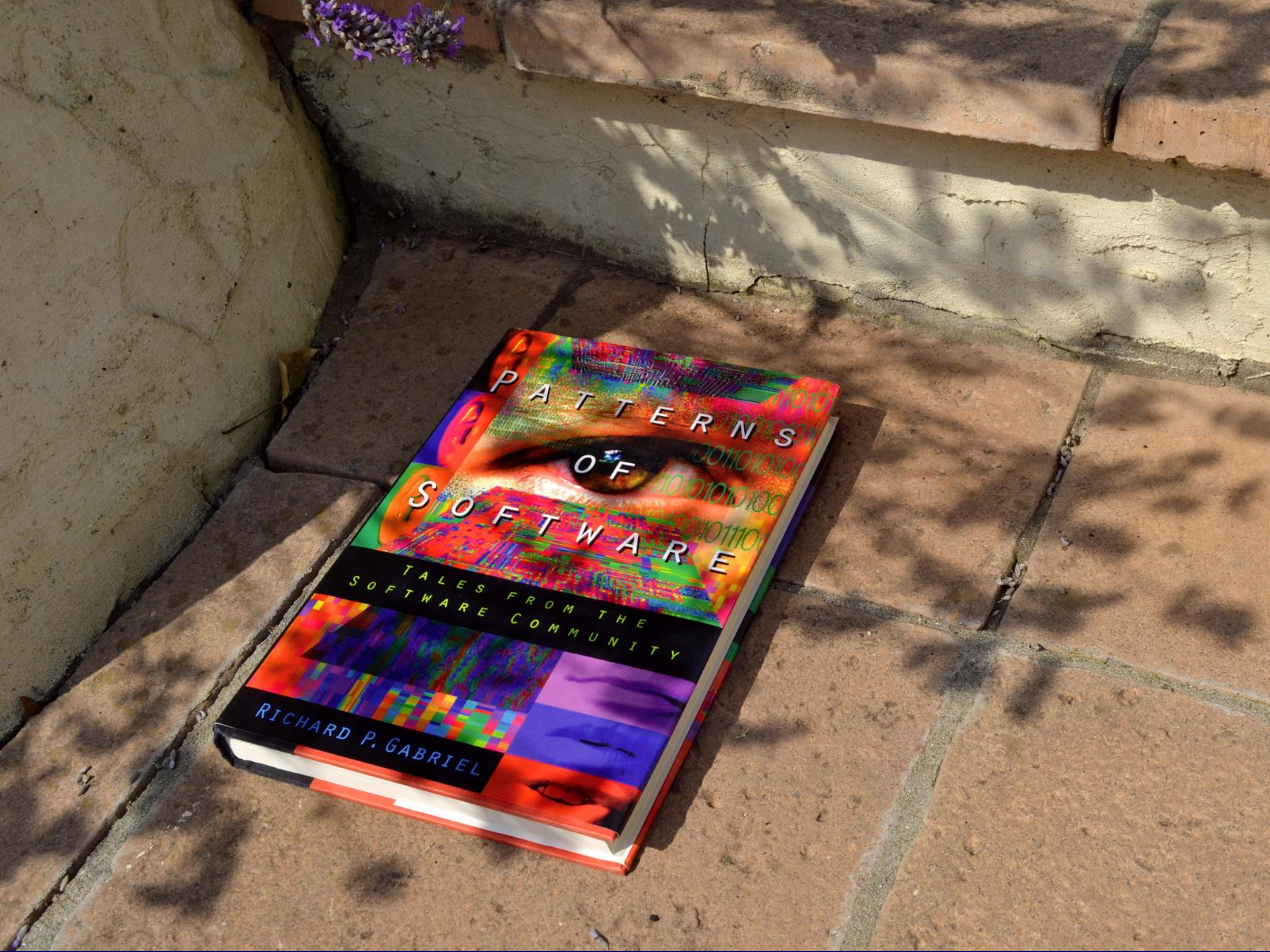
Neil Gaiman

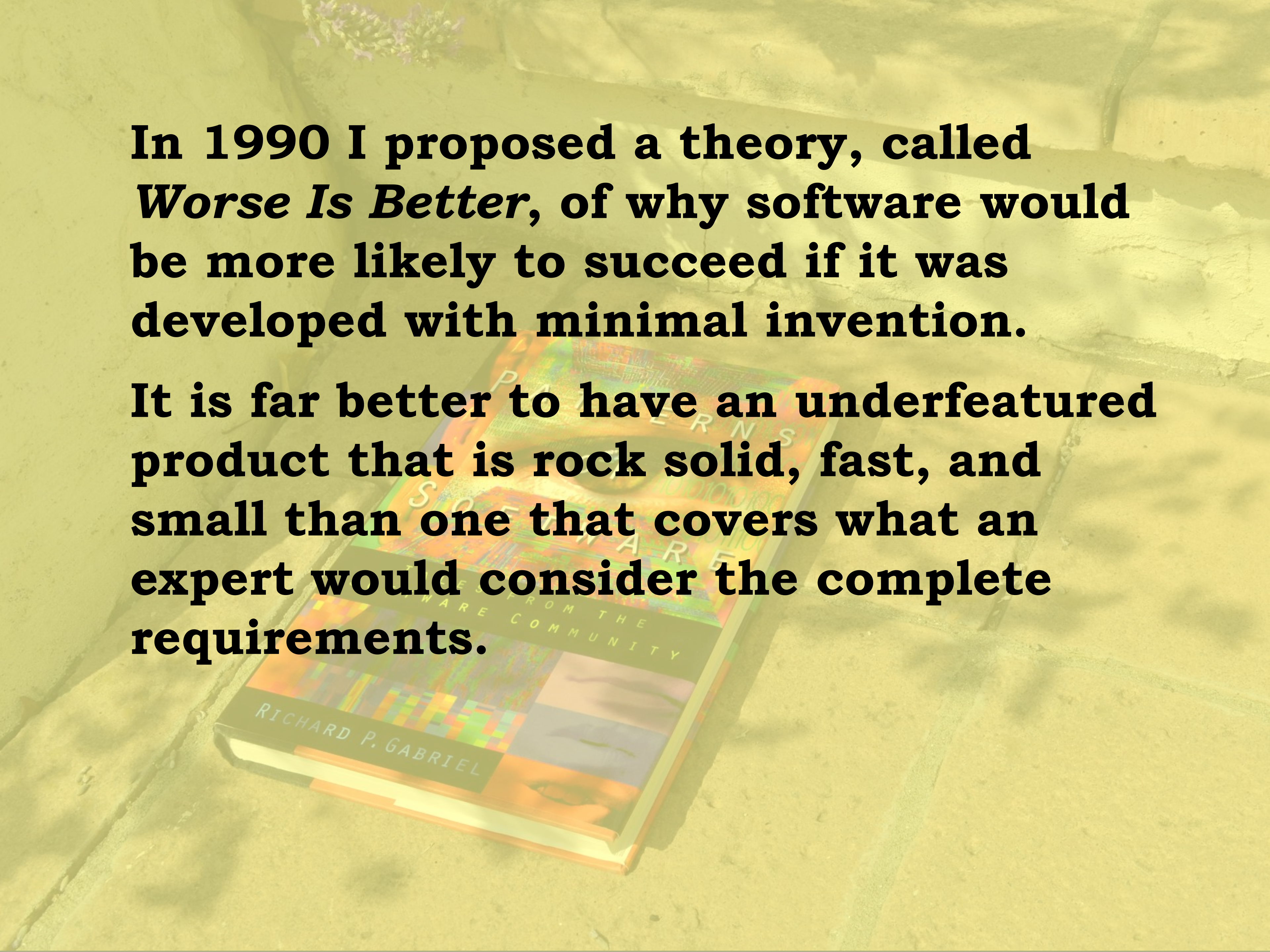
Programming is a design activity.

Jack W Reeves
"What Is Software Design?"

Coding actually makes sense more often than believed. Often the process of rendering the design in code will reveal oversights and the need for additional design effort. The earlier this occurs, the better the design will be.

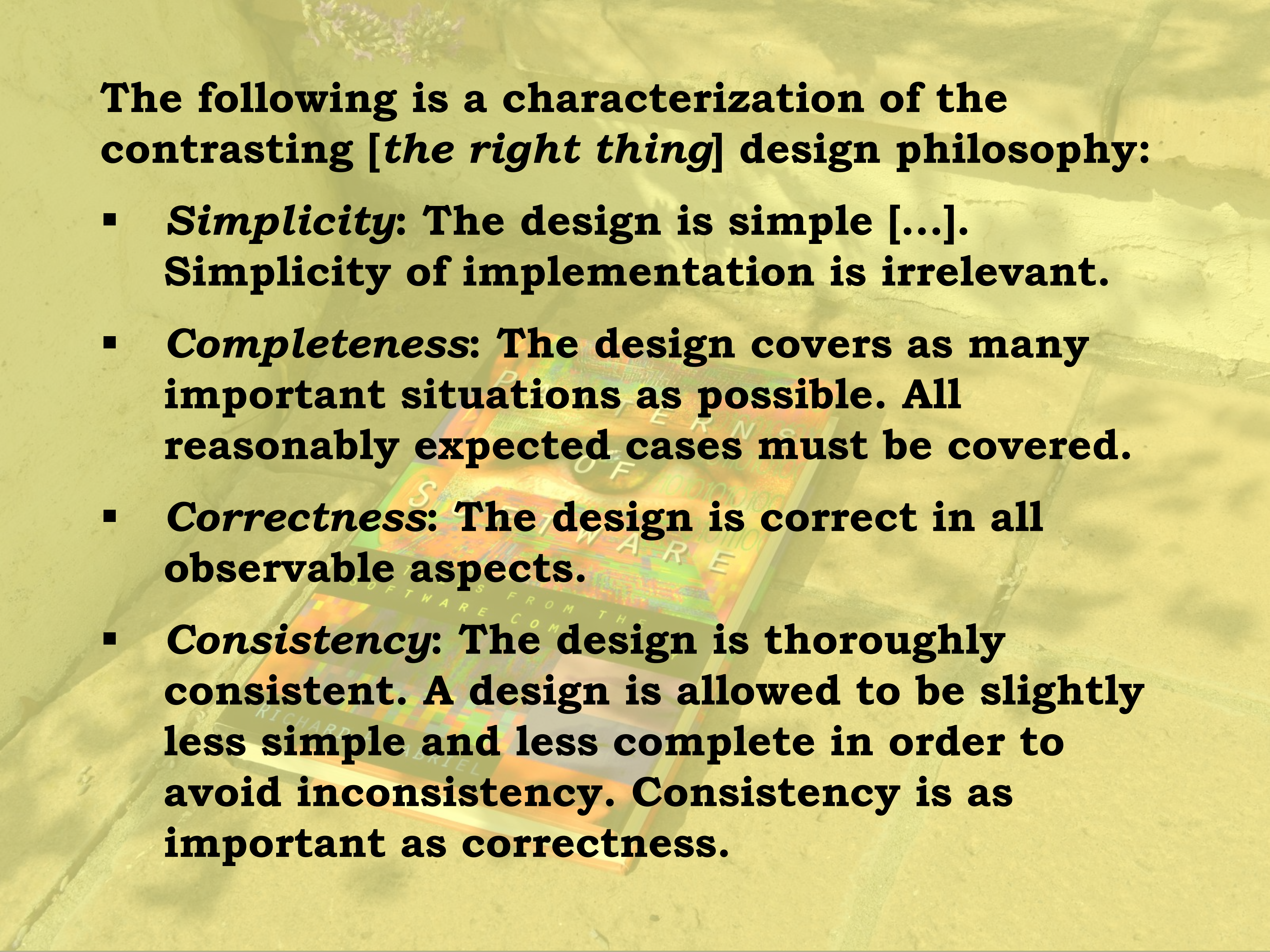
Jack W Reeves
"What Is Software Design?"


PATTERNS OF SOFTWARE
TALES FROM THE SOFTWARE COMMUNITY
RICHARD P. GABRIEL



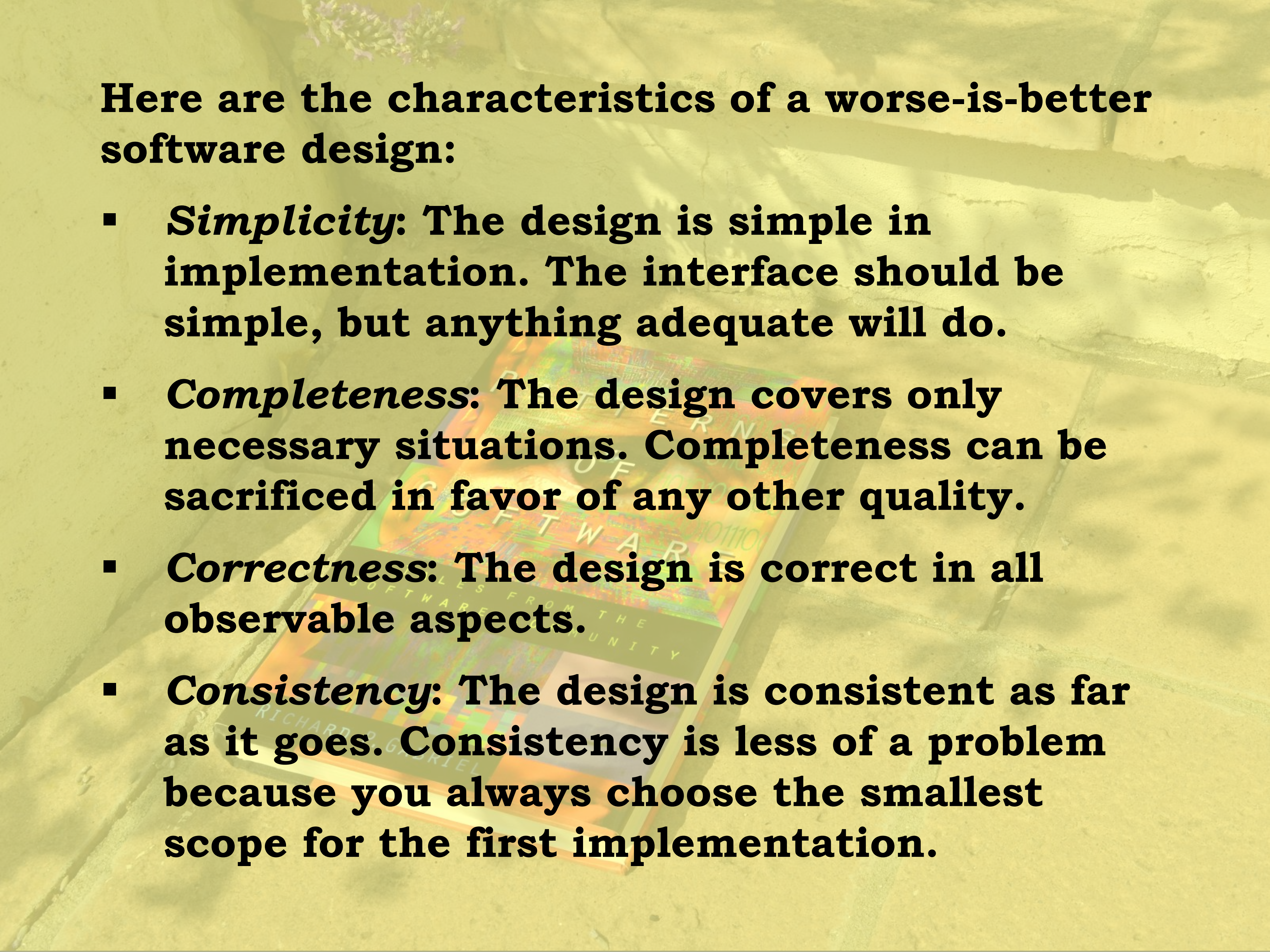
In 1990 I proposed a theory, called *Worse Is Better*, of why software would be more likely to succeed if it was developed with minimal invention.

It is far better to have an underfeatured product that is rock solid, fast, and small than one that covers what an expert would consider the complete requirements.



The following is a characterization of the contrasting [*the right thing*] design philosophy:

- *Simplicity*: The design is simple [...]. Simplicity of implementation is irrelevant.
- *Completeness*: The design covers as many important situations as possible. All reasonably expected cases must be covered.
- *Correctness*: The design is correct in all observable aspects.
- *Consistency*: The design is thoroughly consistent. A design is allowed to be slightly less simple and less complete in order to avoid inconsistency. Consistency is as important as correctness.

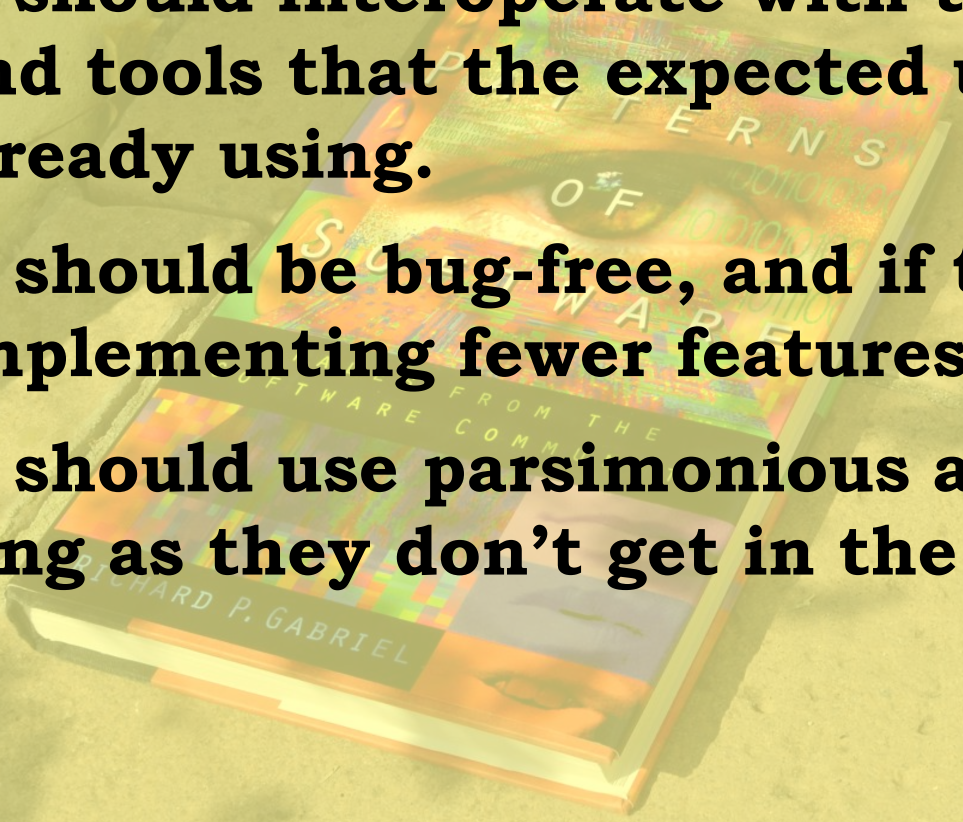


Here are the characteristics of a worse-is-better software design:

- ***Simplicity*: The design is simple in implementation. The interface should be simple, but anything adequate will do.**
- ***Completeness*: The design covers only necessary situations. Completeness can be sacrificed in favor of any other quality.**
- ***Correctness*: The design is correct in all observable aspects.**
- ***Consistency*: The design is consistent as far as it goes. Consistency is less of a problem because you always choose the smallest scope for the first implementation.**

Implementation characteristics are foremost:

- The implementation should be fast.
- It should be small.
- It should interoperate with the programs and tools that the expected users are already using.
- It should be bug-free, and if that requires implementing fewer features, do it.
- It should use parsimonious abstractions as long as they don’t get in the way.



- An architecture should be considered a set of hypotheses to be confirmed
- Architecture involves discovery (and surprise)
- Architecture should be grown iteratively and incrementally
- Start with a walking skeleton and build onto that
- Focus on reducing scope rather than reducing quality
- Aim for complete features rather than "feature complete" development

How can change and uncertainty be handled?

Um. What's the name of the word for things not being the same always. You know, I'm sure there is one. Isn't there?

There's must be a word for it... the thing that lets you know time is happening. Is there a word?

Change.

Oh. I was afraid of that.

Neil Gaiman
The Sandman

The moment design becomes important is when you want to change something.

Kent Beck

Speculative Generality

Brian Foote suggested this name for a smell to which we are very sensitive. You get it when people say, "Oh, I think we need the ability to do this kind of thing someday" and thus want all sorts of hooks and special cases to handle things that aren't required. The result often is harder to understand and maintain. If all this machinery were being used, it would be worth it. But if it isn't, it isn't. The machinery just gets in the way, so get rid of it.

Martin Fowler
Refactoring

You have a problem. You decide to solve it with configuration. Now you have <%= $problems %> problems!

Dan North

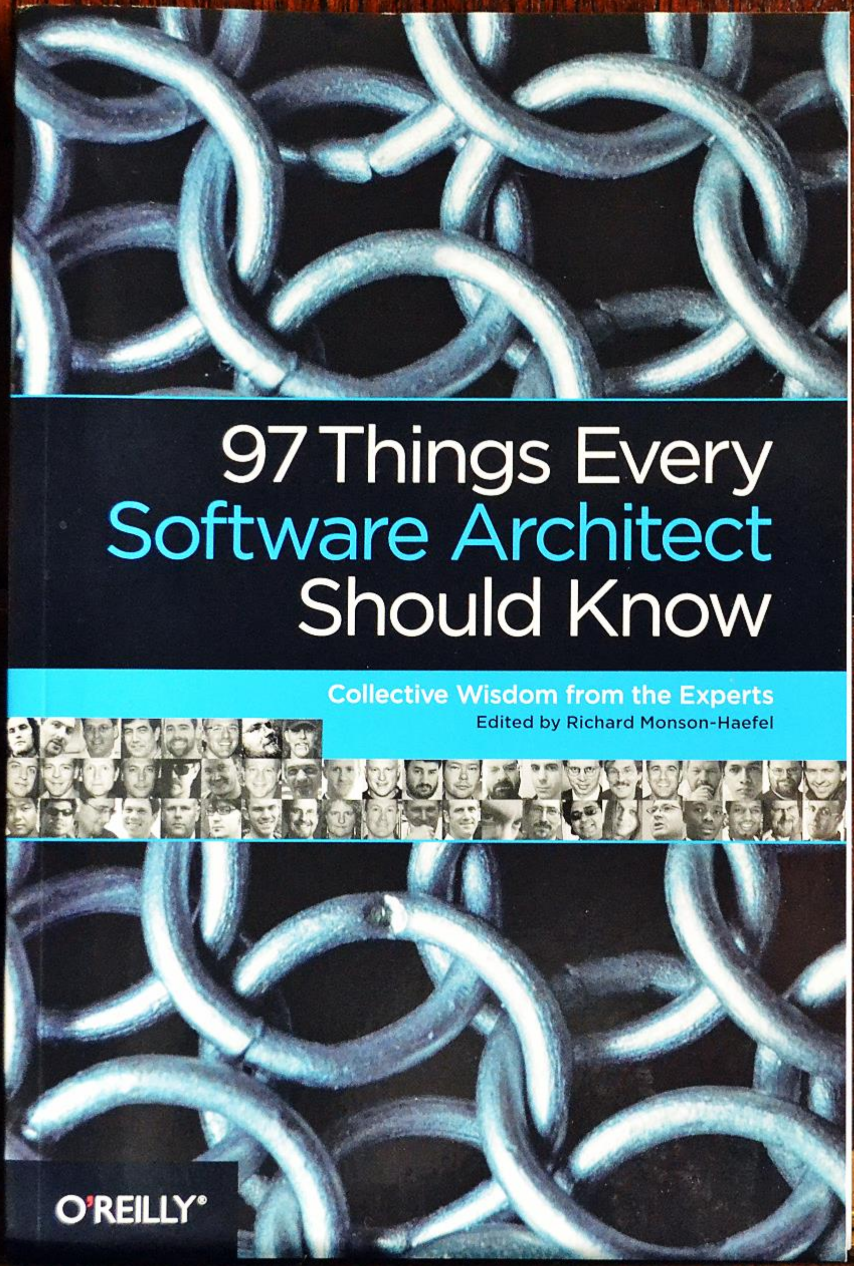
https://twitter.com/tastapod/status/342935892207497219

Public APIs, like diamonds, are forever.

Joshua Bloch
"Bumper-Sticker API Design"
http://www.infoq.com/articles/API-Design-Joshua-Bloch

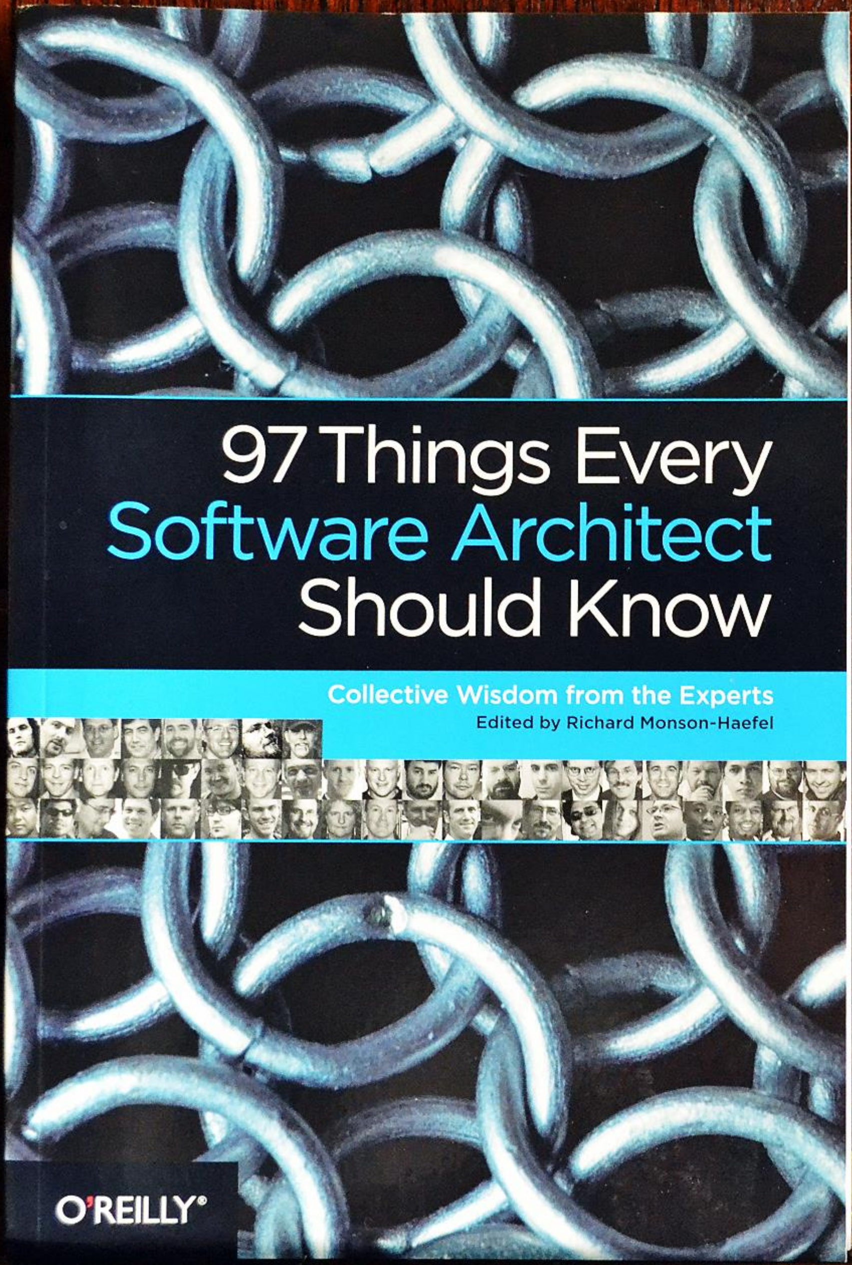
```
interface Iterator
{
    boolean set_to_first_element();
    boolean set_to_next_element();
    boolean set_to_next_nth_element(in unsigned long n) raises(…);
    boolean retrieve_element(out any element) raises(…);
    boolean retrieve_element_set_to_next(out any element, out boolean more) raises(…);
    boolean retrieve_next_n_elements(
        in unsigned long n, out AnySequence result, out boolean more) raises(…);
    boolean not_equal_retrieve_element_set_to_next(in Iterator test, out any element) raises(…);
    void remove_element() raises(…);
    boolean remove_element_set_to_next() raises(…);
    boolean remove_next_n_elements(in unsigned long n, out unsigned long actual_number) raises(…);
    boolean not_equal_remove_element_set_to_next(in Iterator test) raises(…);
    void replace_element(in any element) raises(…);
    boolean replace_element_set_to_next(in any element) raises(…);
    boolean replace_next_n_elements(
        in AnySequence elements, out unsigned long actual_number) raises(…);
    boolean not_equal_replace_element_set_to_next(in Iterator test, in any element) raises(…);
    boolean add_element_set_iterator(in any element) raises(…);
    boolean add_n_elements_set_iterator(
        in AnySequence elements, out unsigned long actual_number) raises(…);
    void invalidate();
    boolean is_valid();
    boolean is_in_between();
    boolean is_for(in Collection collector);
    boolean is_const();
    boolean is_equal(in Iterator test) raises(…);
    Iterator clone();
    void assign(in Iterator from_where) raises(…);
    void destroy();
};
```

```
interface BindingIterator
{
    boolean next_one(out Binding result);
    boolean next_n(in unsigned long how_many, out BindingList result);
    void destroy();
};
```


97 Things Every
Software Architect
Should Know
Collective Wisdom from the Experts
Edited by Richard Monson-Haefel
O'REILLY

The best route to generality is through understanding known, specific examples and focusing on their essence to find an essential common solution. Simplicity through experience rather than generality through guesswork.

Kevlin Henney
"Simplicity before Generality, Use before Reuse"

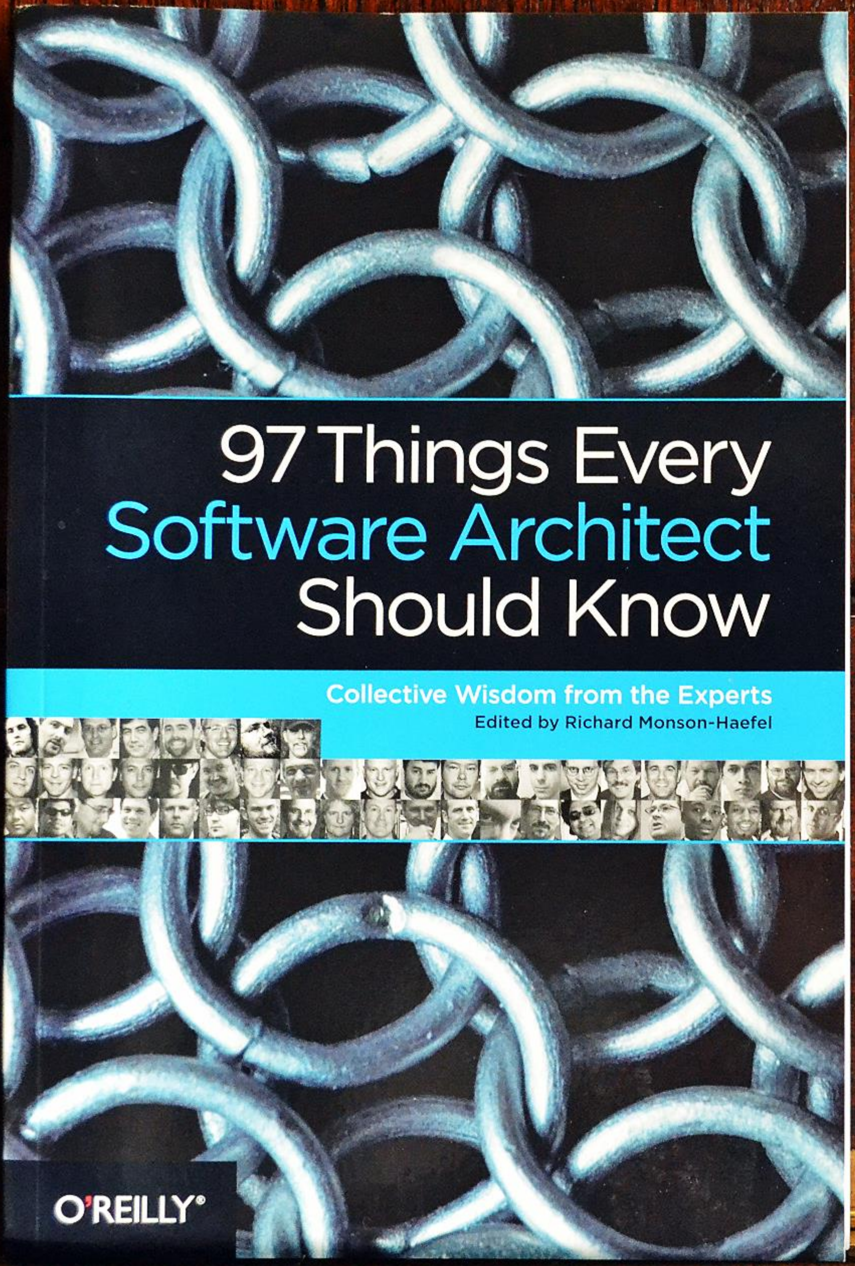

97 Things Every
Software Architect
Should Know
Collective Wisdom from the Experts
Edited by Richard Monson-Haefel
O'REILLY

We can find generality and flexibility in trying to deliver specific solutions, but if we weigh anchor and forget the specifics too soon, we end up adrift in a sea of nebulous possibilities, a world of tricky configuration options, long-winded interfaces, and not-quite-right abstractions.

Kevlin Henney
"Simplicity before Generality, Use before Reuse"

Uncertainty is an uncomfortable position, but certainty is an absurd one.

Voltaire


97 Things Every
Software Architect
Should Know
Collective Wisdom from the Experts
Edited by Richard Monson-Haefel
O'REILLY

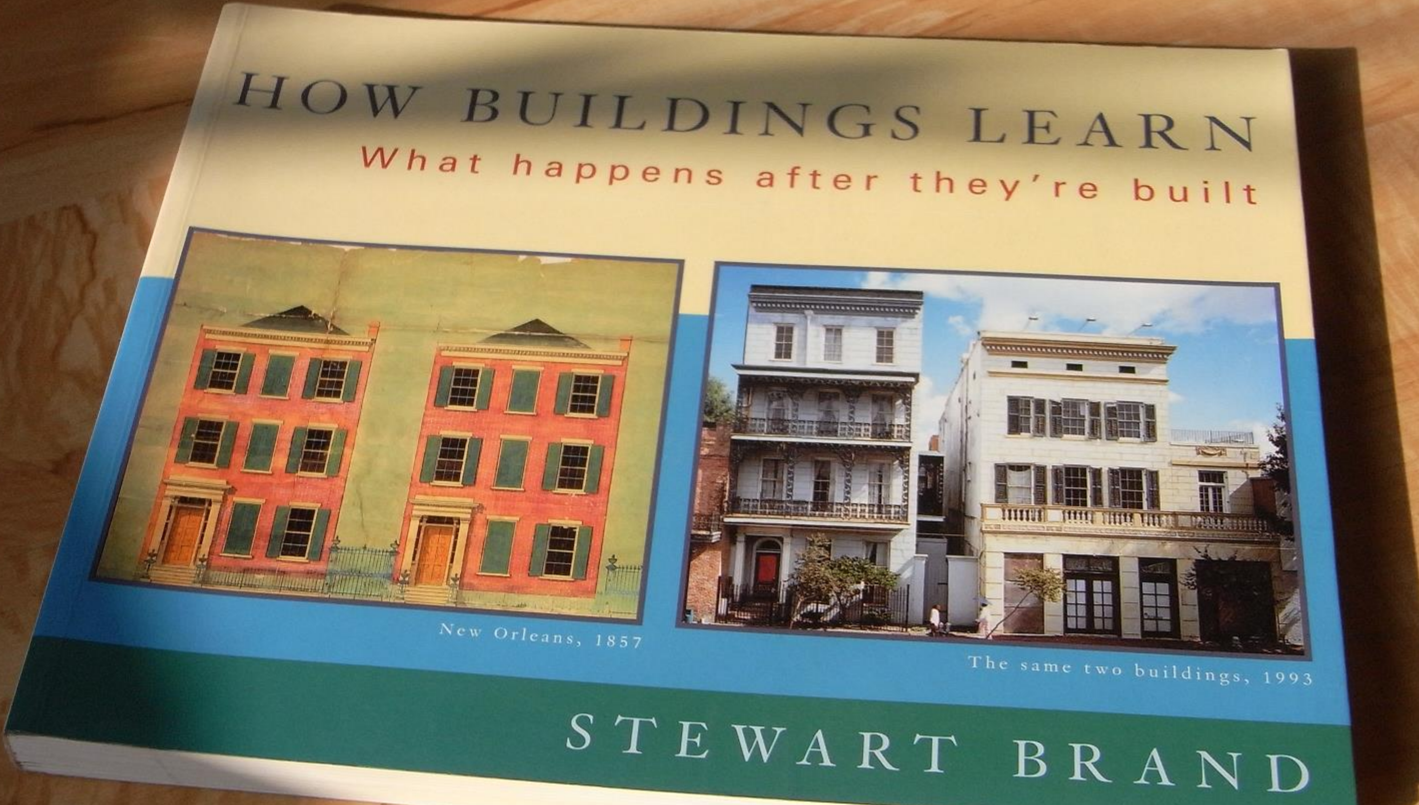
When a design decision can reasonably go one of two ways, an architect needs to take a step back. Instead of trying to decide between options A and B, the question becomes "How do I design so that the choice between A and B is less significant?" The most interesting thing is not actually the choice between A and B, but the fact that there is a choice between A and B.

Kevlin Henney
"Use Uncertainty As a Driver"

We propose [...] that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.

David L Parnas

"On the Criteria to Be Used in Decomposing Systems into Modules"



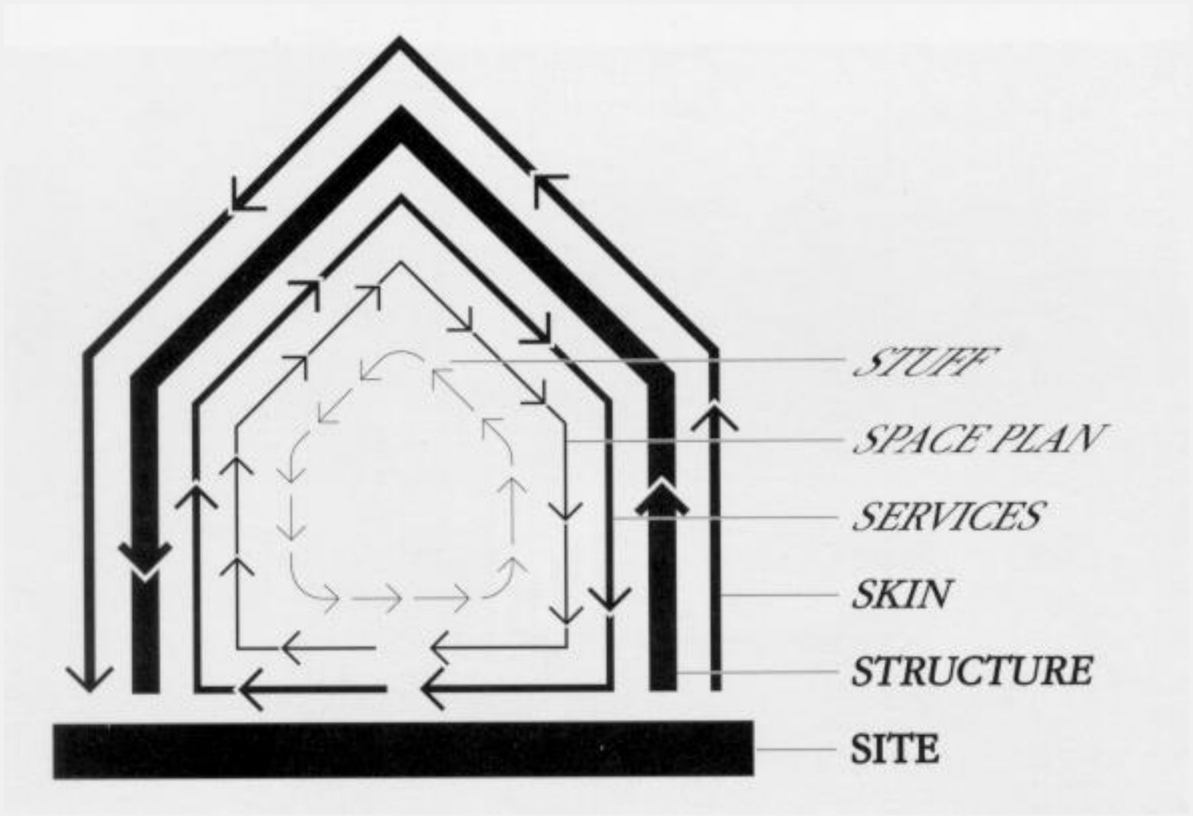
HOW BUILDINGS LEARN

What happens after they're built

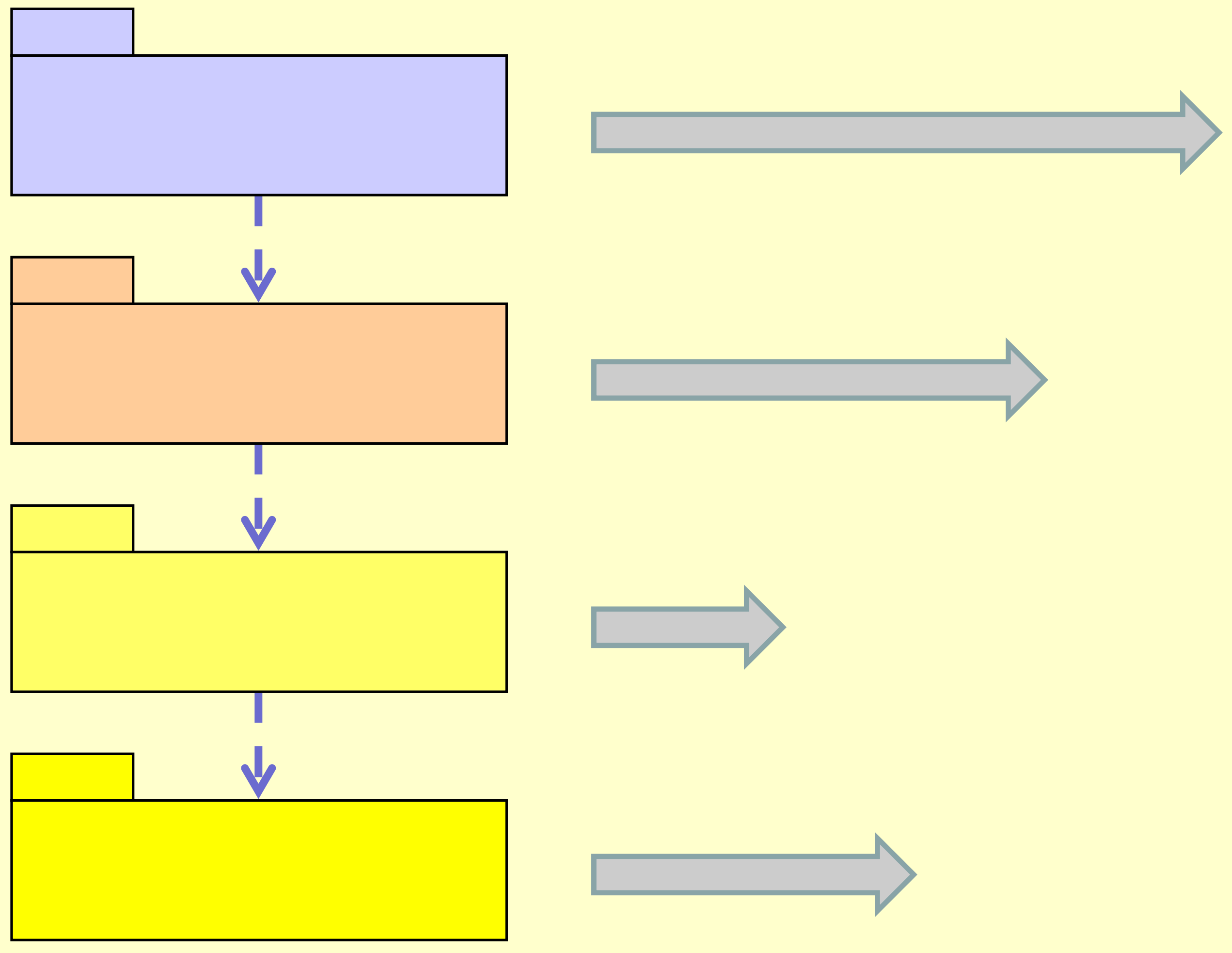
New Orleans, 1857

The same two buildings, 1993

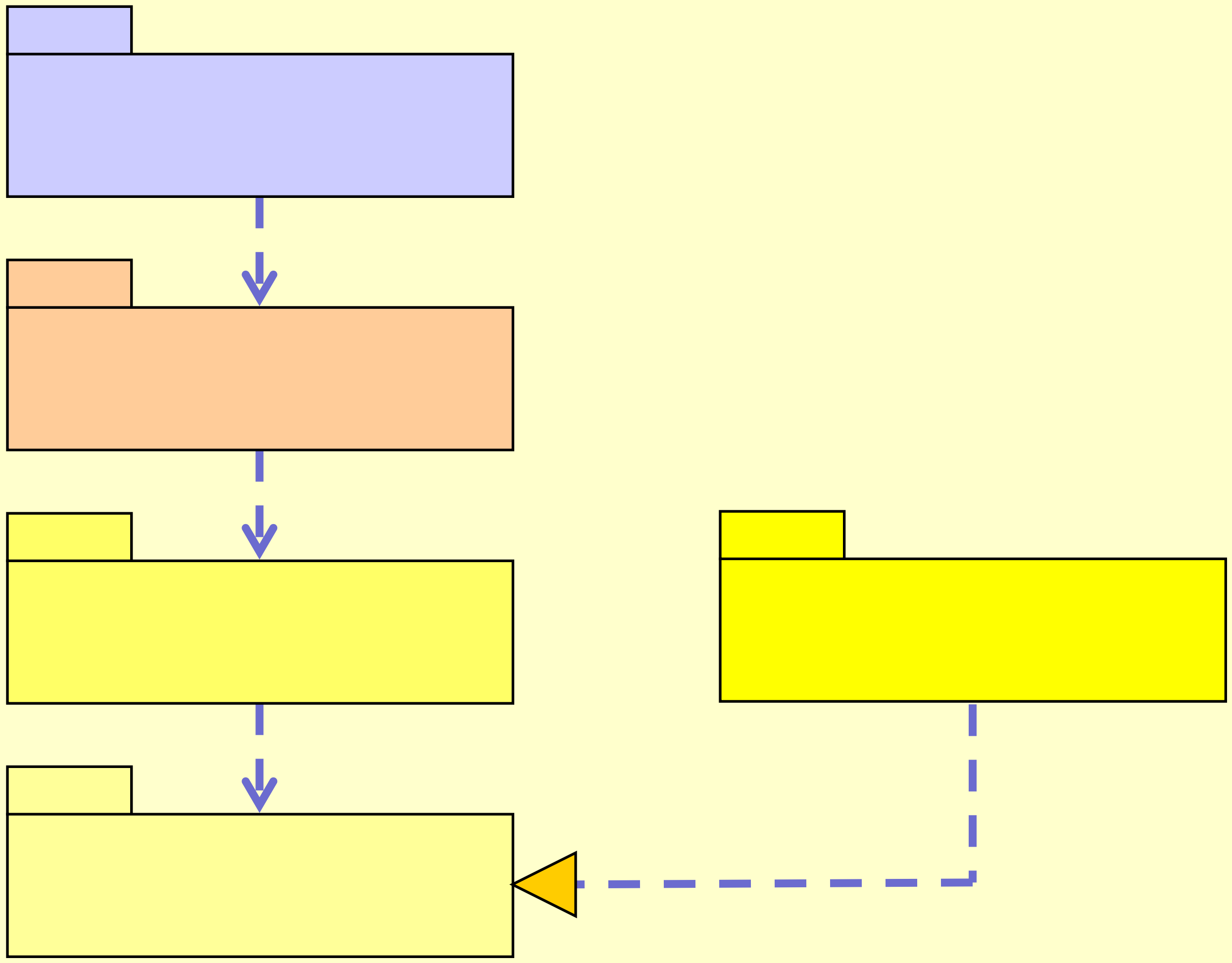
STEWART BRAND



Stewart Brand, *How Buildings Learn*
See also *http://www.laputan.org/mud/*

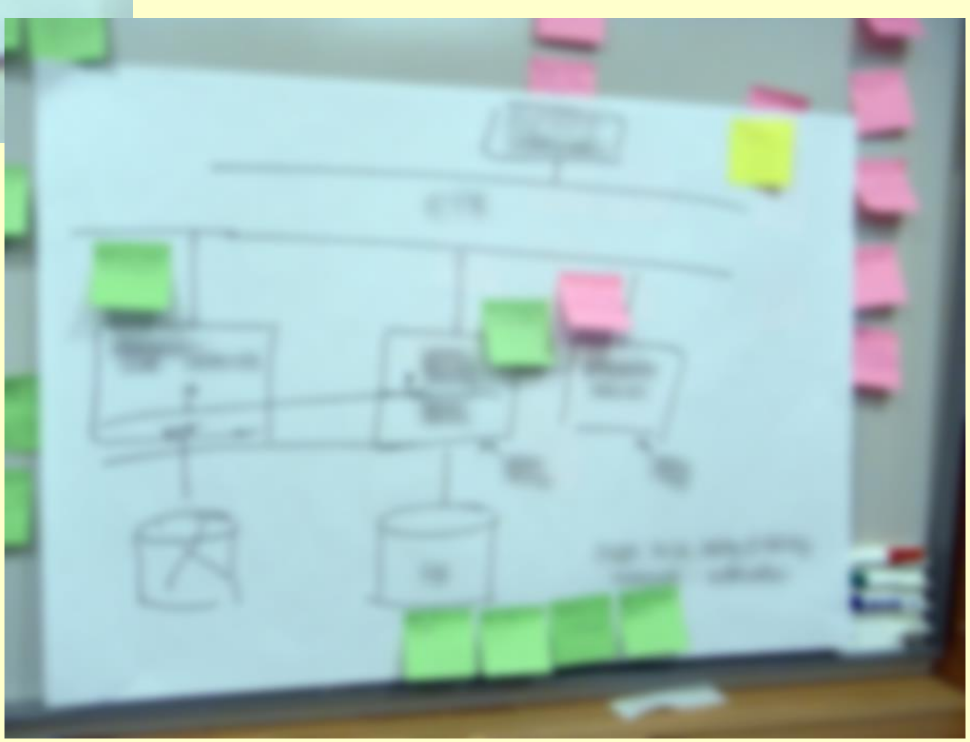


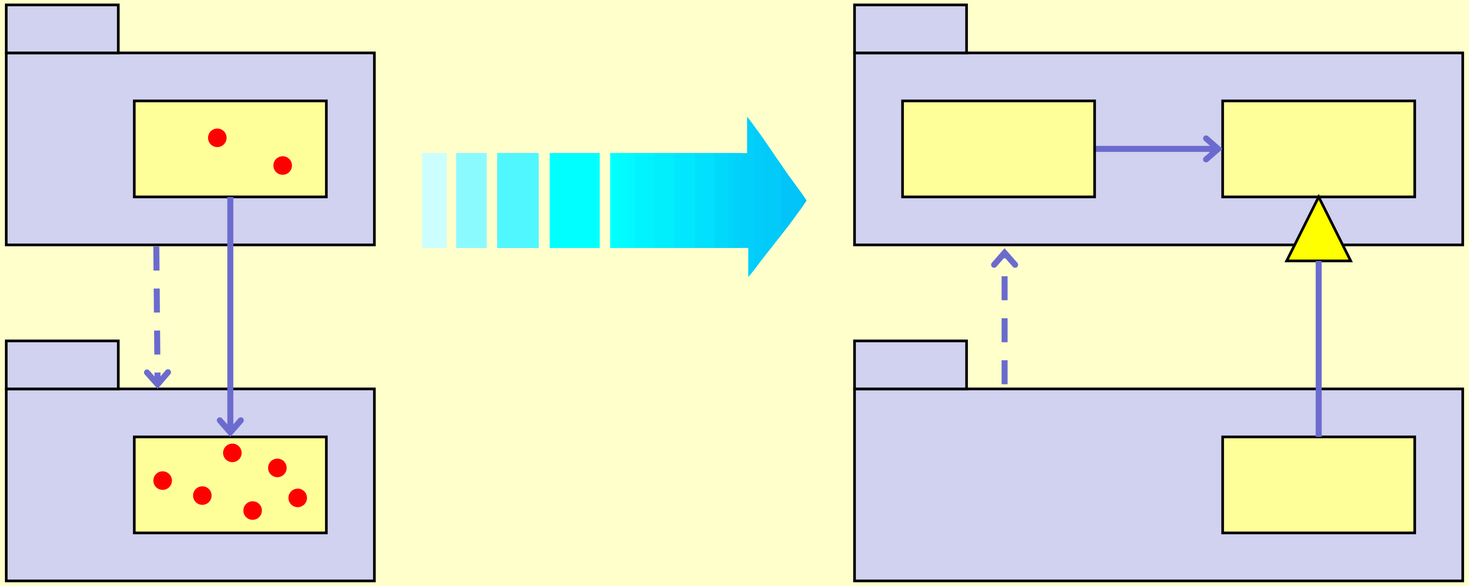
Rate of change



Prediction is very difficult, especially about the future.

Niels Bohr





Scenario buffering by dot-voting possible changes and then readjusting dependencies

- Change is often the only constant
- Use past change to forecast future change — look for the hot spots and at defect density
- Use speculation and future requirements to decide between design alternatives
- Do not use speculation to add extra complexity to the architecture
- Structure the system with respect to rate of change and (un)certainty

What is technical debt and how can it be managed?

101 Things I Learned in Architecture School
Matthew Frederick

No design system is or should be perfect.

As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

Meir Manny Lehman

spaghetti

mess

software entropy

code decay

code rot

technical debt

big ball of mud

software erosion

code smell

Technical Debt is a wonderful metaphor developed by Ward Cunningham to help us think about this problem. In this metaphor, doing things the quick and dirty way sets us up with a technical debt, which is similar to a financial debt.

Like a financial debt, the technical debt incurs interest payments, which come in the form of the extra effort that we have to do in future development because of the quick and dirty design choice. We can choose to continue paying the interest, or we can pay down the principal by refactoring the quick and dirty design into the better design.

The metaphor also explains why it may be sensible to do the quick and dirty approach.

Martin Fowler
http://martinfowler.com/bliki/TechnicalDebt.html

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

http://c2.com/doc/oopsla92.html

**A mess is not a technical debt.
A mess is just a mess.**

Robert Martin

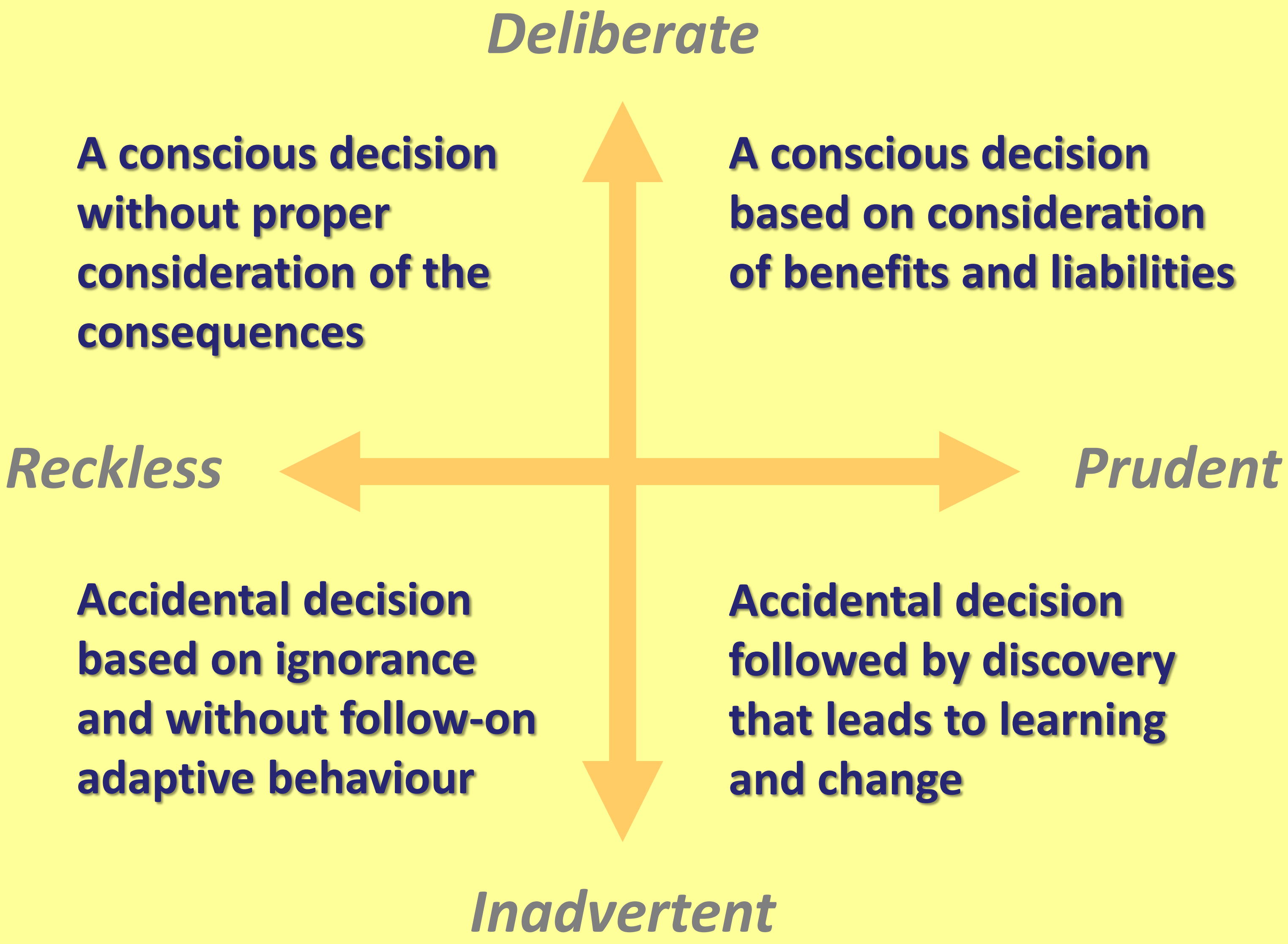
http://blog.objectmentor.com/articles/2009/09/22/a-mess-is-not-a-technical-debt

The useful distinction isn't between debt or non-debt, but between prudent and reckless debt.

Not just is there a difference between prudent and reckless debt, there's also a difference between deliberate and inadvertent debt.

Dividing debt into reckless/prudent and deliberate/inadvertent implies a quadrant.

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



Deliberate

A conscious decision without proper consideration of the consequences

A conscious decision based on consideration of benefits and liabilities

Reckless

Prudent

Accidental decision based on ignorance and without follow-on adaptive behaviour

Accidental decision followed by discovery that leads to learning and change

Inadvertent

amortise

restructure

repayment

write off

balance

asset

interest

credit rating

default

liability

principal

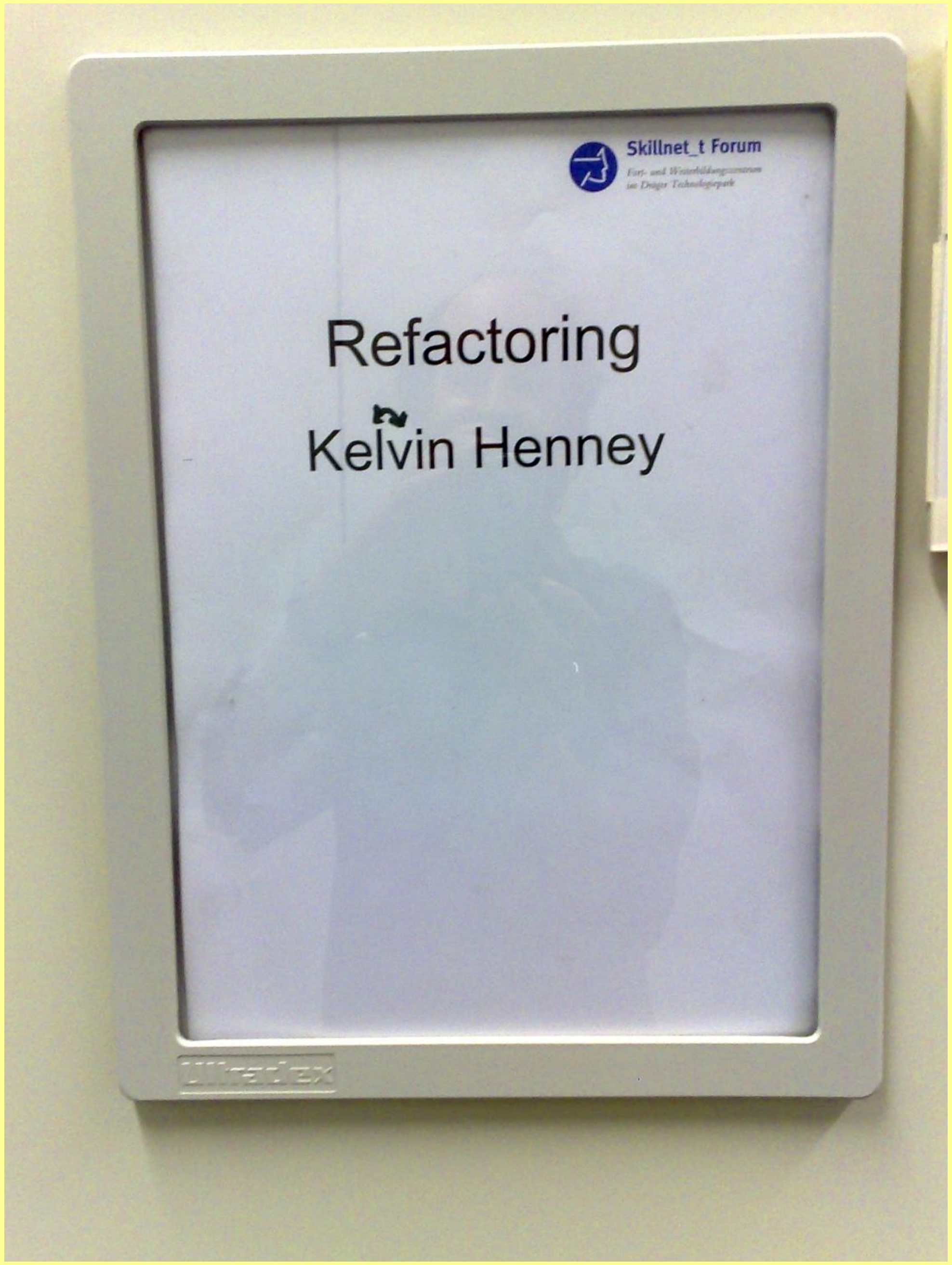
runaway

spiralling

loan

value

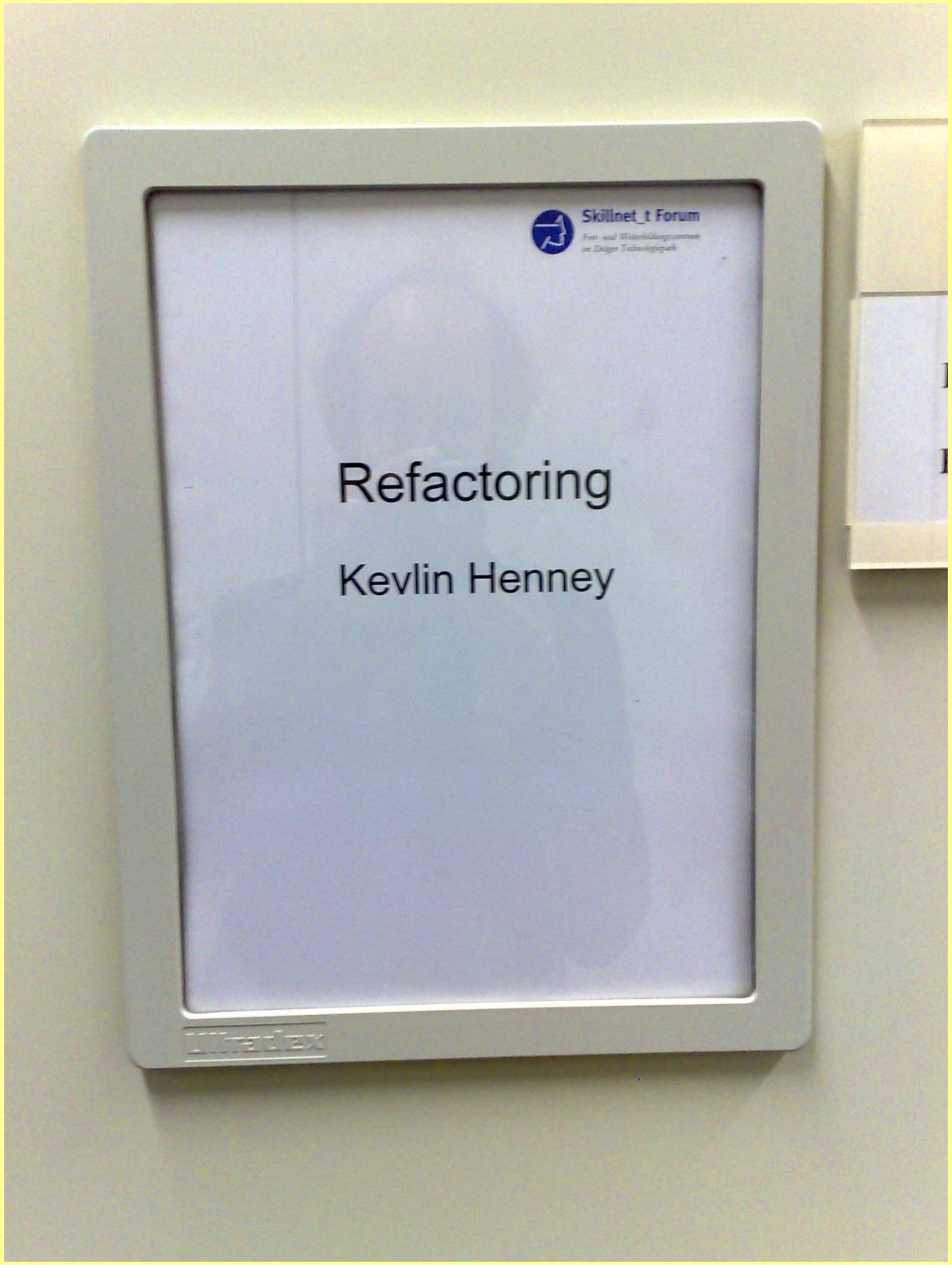
consolidation



Skillnet_t Forum
Fort- und Weiterbildungszentrum im Dräger Technologiepark

Refactoring

Kelvin Henney



Refactoring

Kevlin Henney

Refactoring (noun): ***a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.***

Refactor (verb): ***to restructure software by applying a series of refactorings without changing the observable behavior of the software.***

Martin Fowler
Refactoring

repair

refactoring

re-evaluation

remembering

revision

re-engineering

rewriting

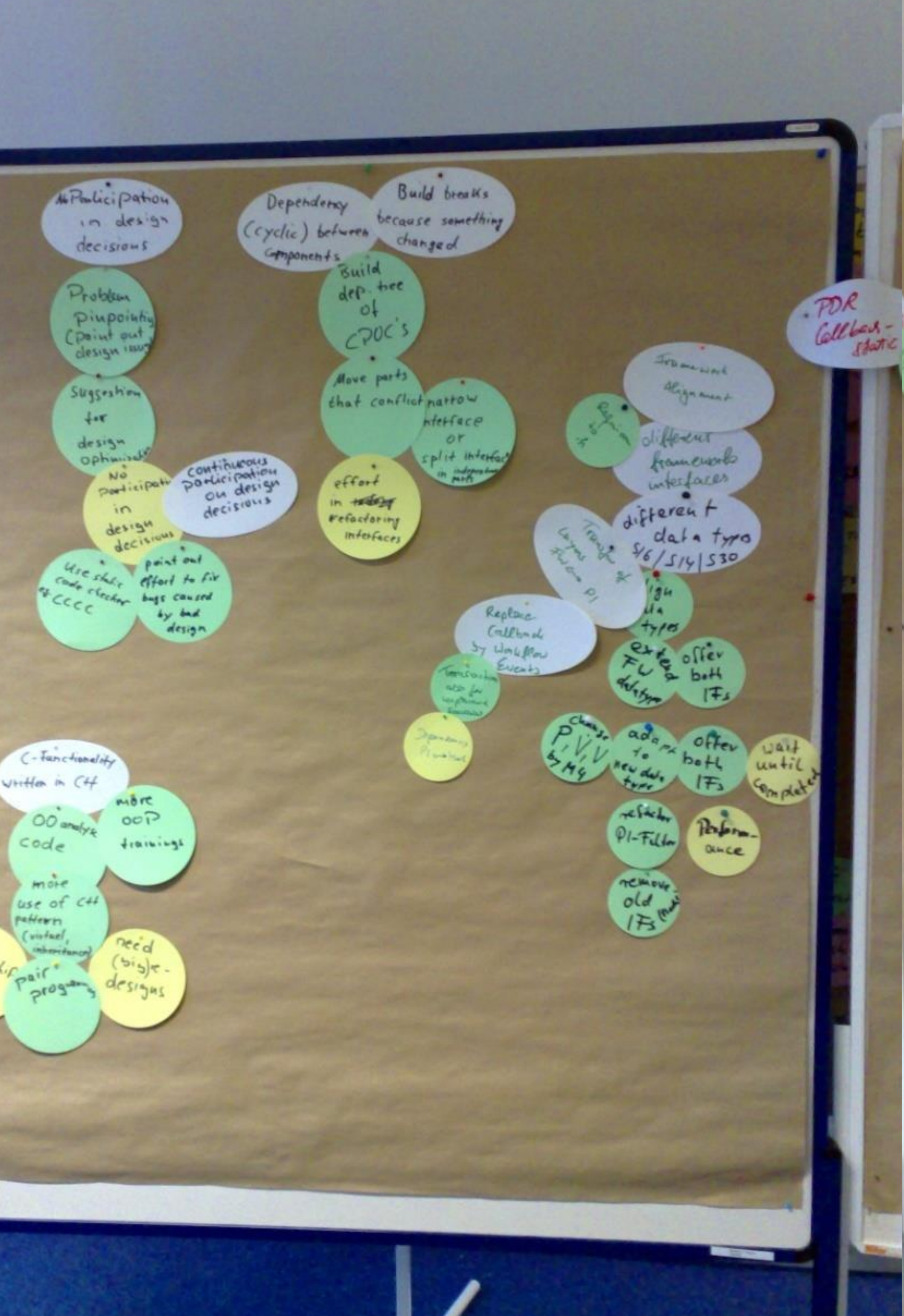
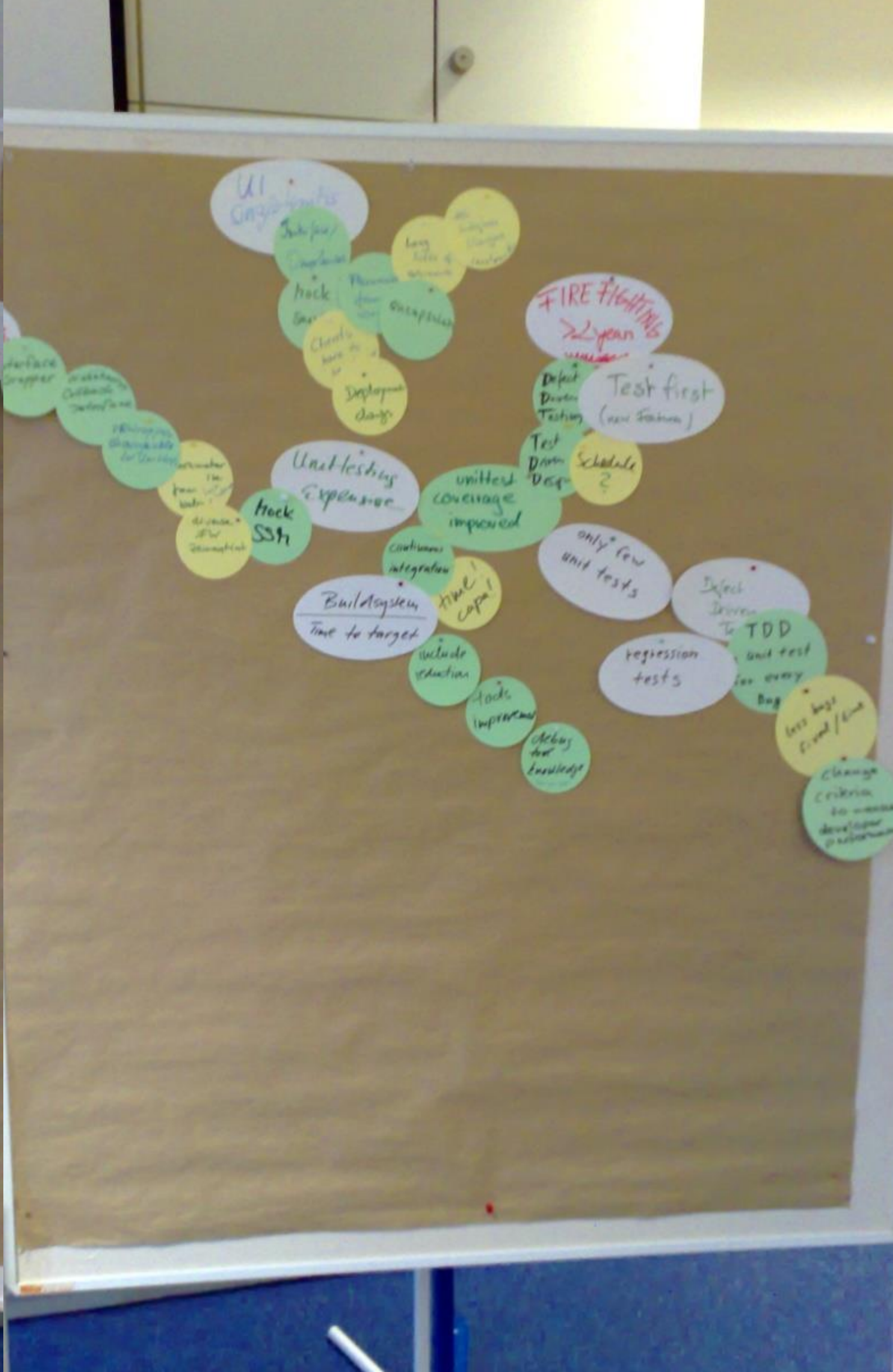
retrospection

reduction

reaction

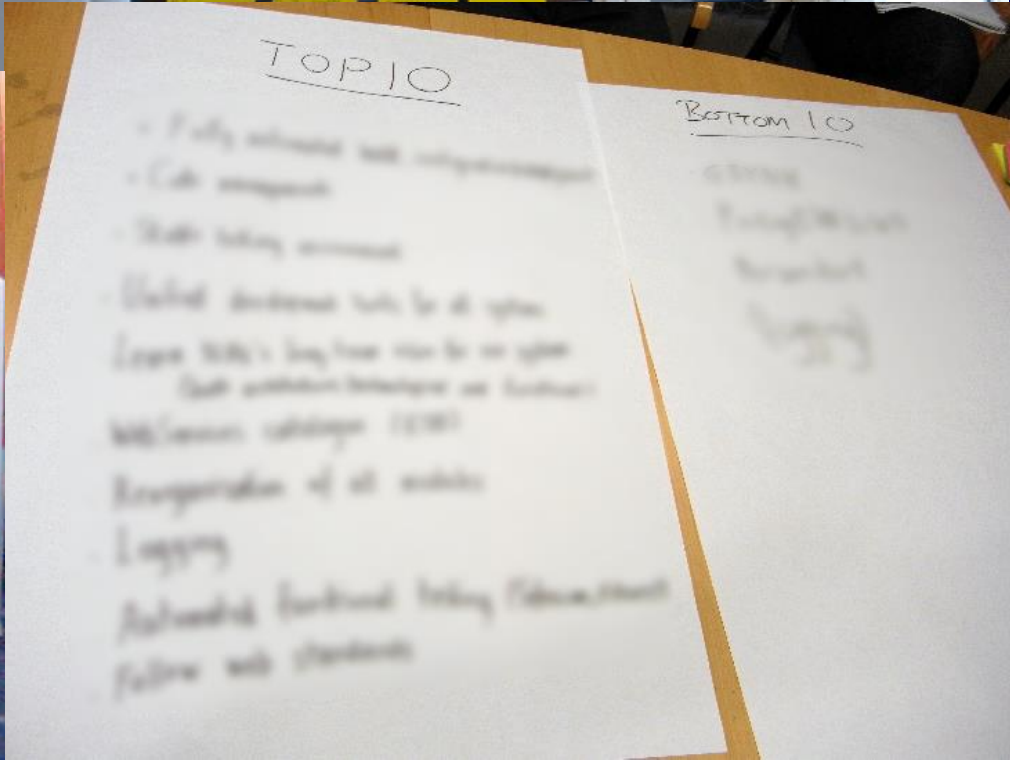
recovery

reuse


No Participation in design decisions
Dependency (cyclic) between components
Build breaks because something changed
Problem pinpointing (point out design issue)
Build dep. tree of CPOC's
PDR Callback-static
Suggestion for design optimization
Move parts that conflict
narrow interface or split interface
Framework alignment
different frameworks interfaces
different data types
No participation in design decisions
continuous participation on design decisions
effort in refactoring interfaces
Use static code checker e.g. CCCC
point out effort to fix bugs caused by bad design
Replace Callback by Workflow Events
extend FW datatype
offer both IFs
Dependency Pl
C-Functionality written in C++
more OOP trainings
OO analyze code
more use of C++ pattern (virtual, inheritance)
need (bigger) designs
pair programming
Change P, V, V by MG
adapt to new data type
offer both IFs
wait until completed
refactor PI-Filter
Performance
remove old IFs

UI Components
Mock
FIRE FIGHTING
Test first (new features)
Defect Driven Testing
Test Driven Dev
Schedule?
Unittesting expensive
unittest coverage improved
Mock SSH
Continuous integration
time / capa!
only few unit tests
Defect Driven T
Buildsystem
Time to target
include education
tools improvement
TDD unit test for every bug
regression tests
less bugs fixed / time
Change criteria to measure developer performance

Testlanguage for Ventilation
DSL ?
USE ANOTHER LANGUAGE
reduce amount of queue's
Message parser
too much implementation detail
Less detail in UML models
EARLIER CODE REVIEWS
Interface Definition
Documentation of Interfaces
Architektur/ Dependencies
Release management
Writing Requirements
tool chain
Clear Case
UCM config management
Traceability TEST
Testsystem (Unittests)
framework

SCRUM


TOP 10
BOTTOM 10

101 Things I Learned in Architecture School
Matthew Frederick

A good designer isn't afraid to throw away a good idea.

- There is little excuse for introducing reckless debt
- Awareness of technical debt is the responsibility of all roles
- Consideration of debt must involve practice and process
- Management of technical debt must account for business value
- Perfection is not possible, but understanding the ideal is useful
- For change hotspots, habitability is always a consideration

The ability to simplify means to eliminate the unnecessary so that the necessary may speak.

Hans Hofmann