zühlke
empowering ideas

Creating An Incremental Architecture For Your System

ACCU Conference, April 2018, Bristol, UK

Giovanni Asproni
email: giovanni.asproni@zuhlke.com
gasproni@asprotunity.com
twitter: @gasproni
linkedin: http://www.linkedin.com/in/gasproni

Agenda

- What is Software Architecture
- The problems with the “Rational Model”
- What is Incremental Architecture and its advantages
- Getting started
- Growing the system
- Incremental Architecture for legacy and brownfield systems

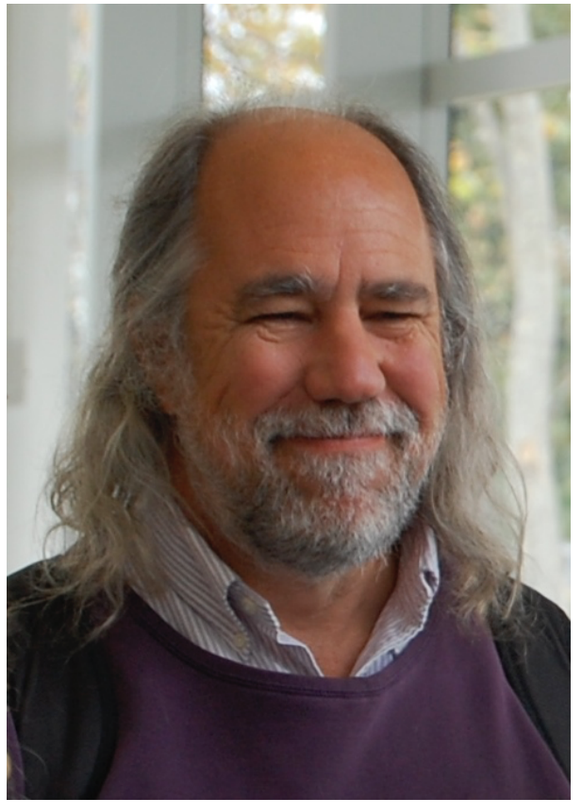


Photo from: https://commons.wikimedia.org/wiki/File:Grady_Booch,_CHM_2011_2_cropped.jpg

"Every software-intensive system has an architecture. In some cases that architecture is intentional, while in others it is accidental […] **the architecture of a system is the naming of the most significant design decisions that shape a system, where we measure significant by cost of change and by impact upon use.**"

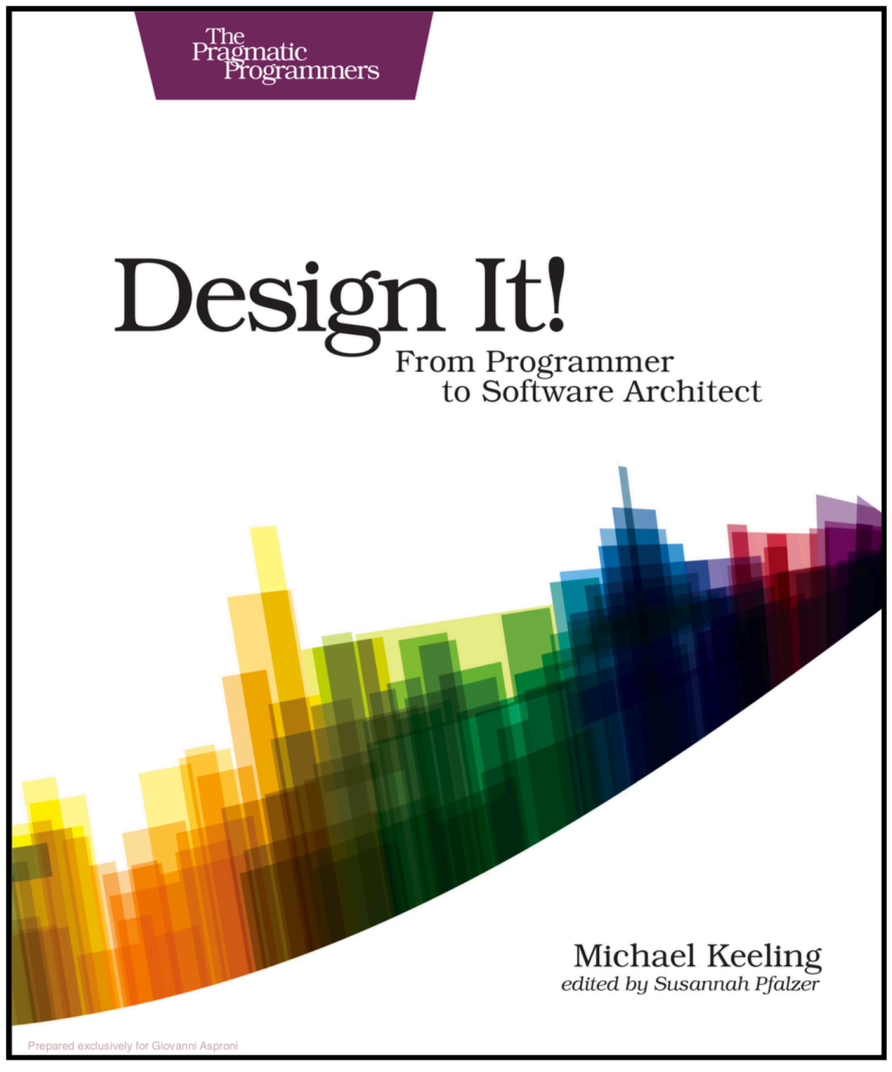
http://handbookofsoftwarearchitecture.com/?p=63

“Cost of change” is a relative value

zühlke
empowering ideas

"Impact upon use" is about how much the users of the system are affected by those decisions

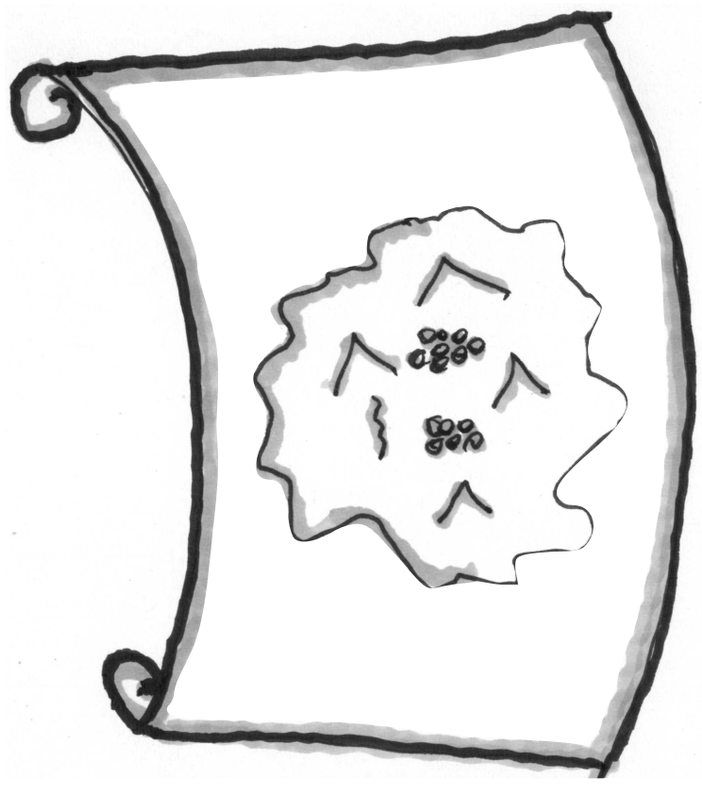
zühlke
empowering ideas



"Today's programmers make design decisions about the architecture every single day. When one line of code can tank a required quality attribute, then you are a software architect whether you identify as one or not."

Architecture Is For

zühlke
empowering ideas



- Making important structural choices
- Recording and communicating decisions
- Guiding implementation decisions
- Splitting work among teams

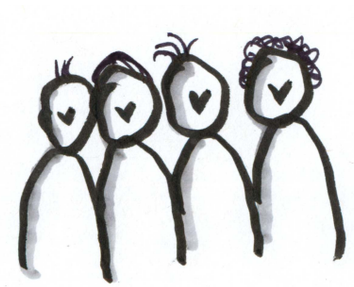
What Does Architecture Include?

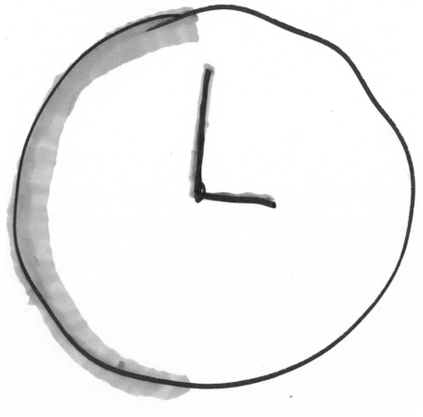
zühlke
empowering ideas

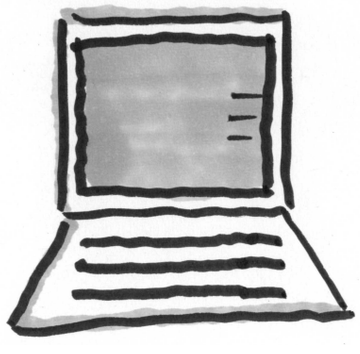
- Static and dynamic views of the system
- Sketches
- Prototypes and reference implementations
- Documentation
- Design and implementation guidelines
- Anything else necessary to understand “the shape” and the behaviour of the system

zühlke
empowering ideas

Architectural Drivers

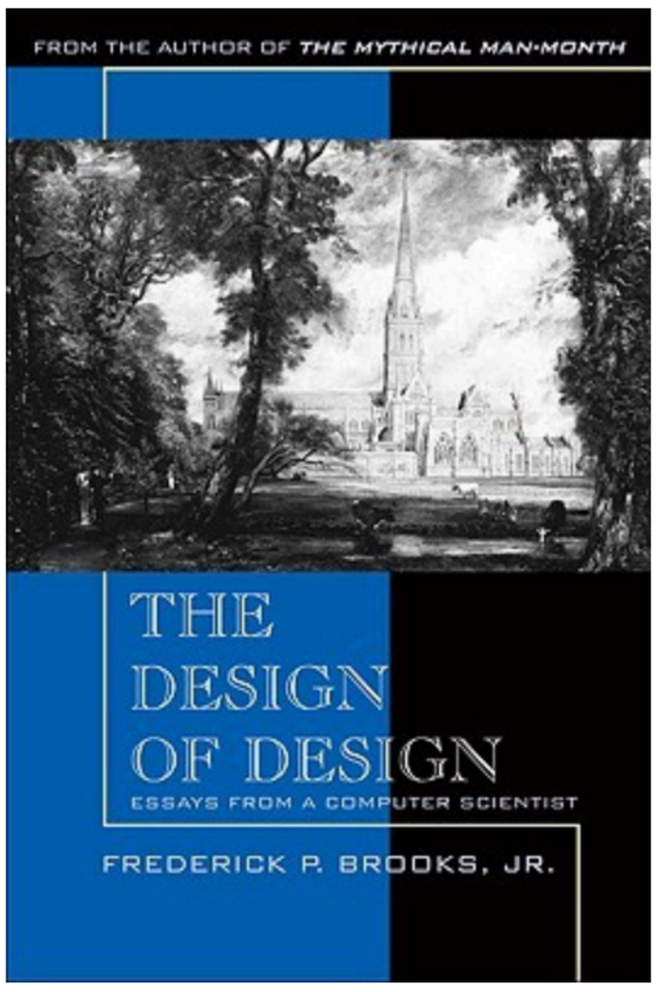






The Rational Design Process

zühlke
empowering ideas



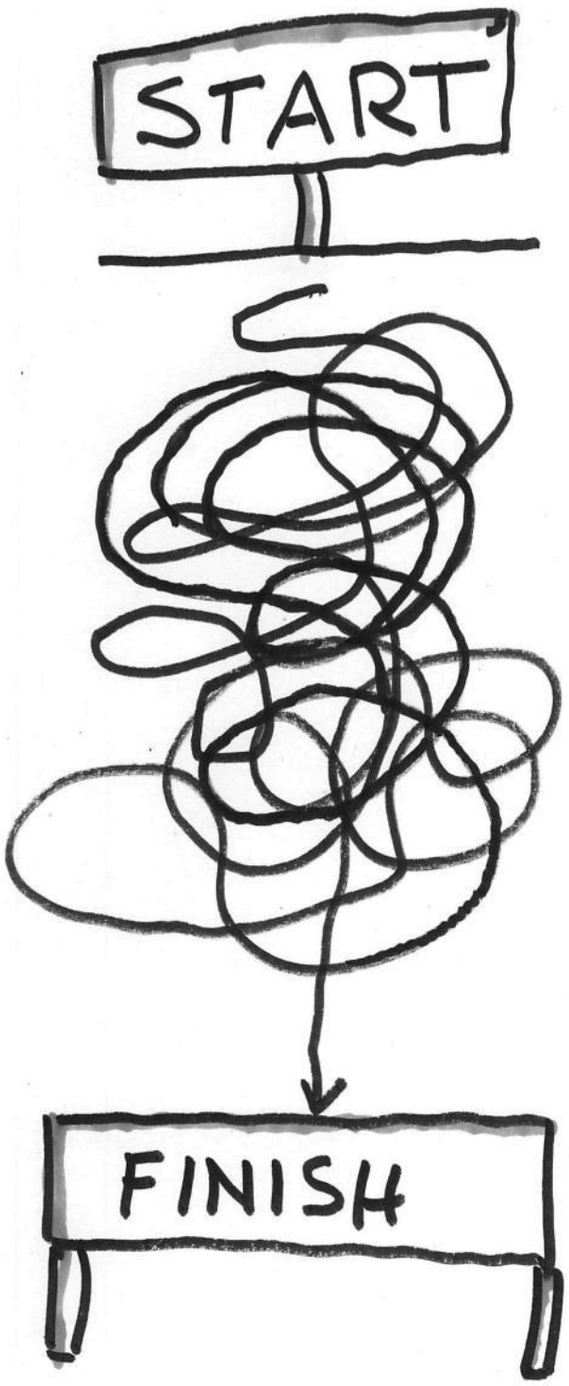
- **Goal**
- **Desiderata**
- **Utility function**
- **Constraints, especially budget (perhaps not $ cost)**
- **Design tree of decisions**

UNTIL ("good enough") or (time runs out)
DO another design (to improve utility function)
UNTIL design is complete
WHILE design remains feasible,
make another design decision
END WHILE
Backtrack up design tree
Explore a path not searched before
END UNTIL
END DO
Take best design
END UNTIL

From: The Design of Design, Fred Brooks

The Rational Design Process Doesn't Work!

zühlke
empowering ideas



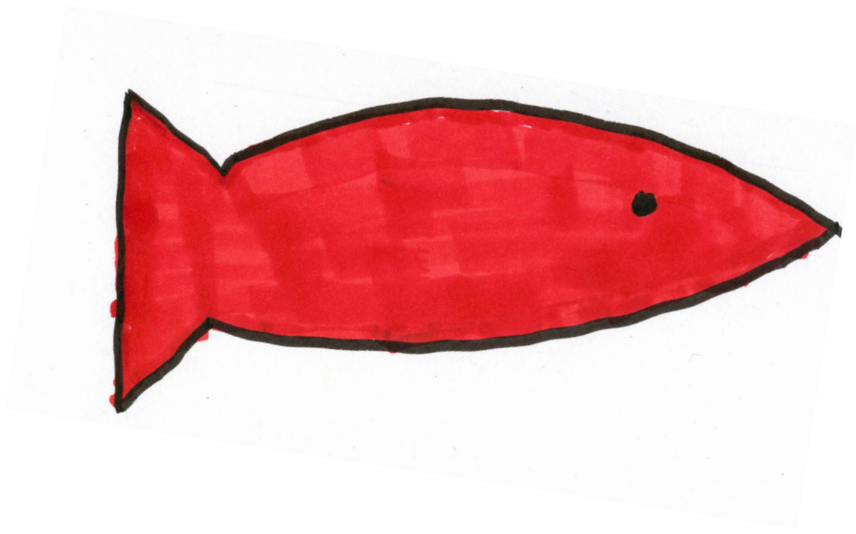
- Design Isn't Just to Satisfy Requirements, but Also to Uncover them
- Design Isn't Simply Selecting from Alternatives, but Also Realizing Their Existence
- Often the domain is not well understood even by the domain experts!

zühlke
empowering ideas

The chances of success of a big upfront design are inversely proportional to the size of the system being designed

BUFD Is A Red Herring

zühlke
empowering ideas

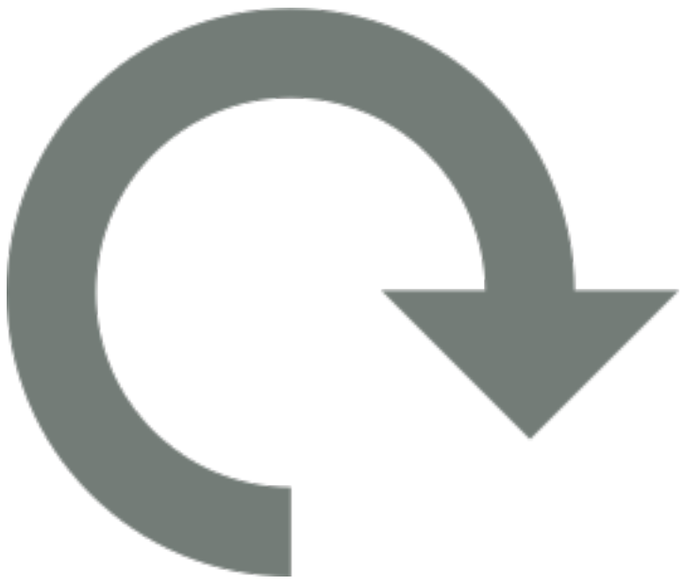


- Most projects have:
  - Speculative Upfront Design (SUFD)
  - Absence of refactoring

“incremental design (also called evolutionary design) allows you to build technical infrastructure (such as domain models and persistence frameworks) incrementally, in small pieces, as you deliver stories”

Incremental Architecture

zühlke
empowering ideas



- One small (set of) feature(s) at a time
  - Design
  - Implement
  - Test
  - Deploy

One More Thing…

zühlke
empowering ideas

Adding increments is not enough, you need to iterate as well!

An Incremental Approach Can Alleviate / Remove Constraints

zühlke
empowering ideas

- Early production => early revenue => fewer budget restrictions

- A technology can be learnt while creating and deploying the system

- Often, important deadlines can be met with just some basic functionality

zühlke
empowering ideas

An incremental approach allows to create the right domain model for the application

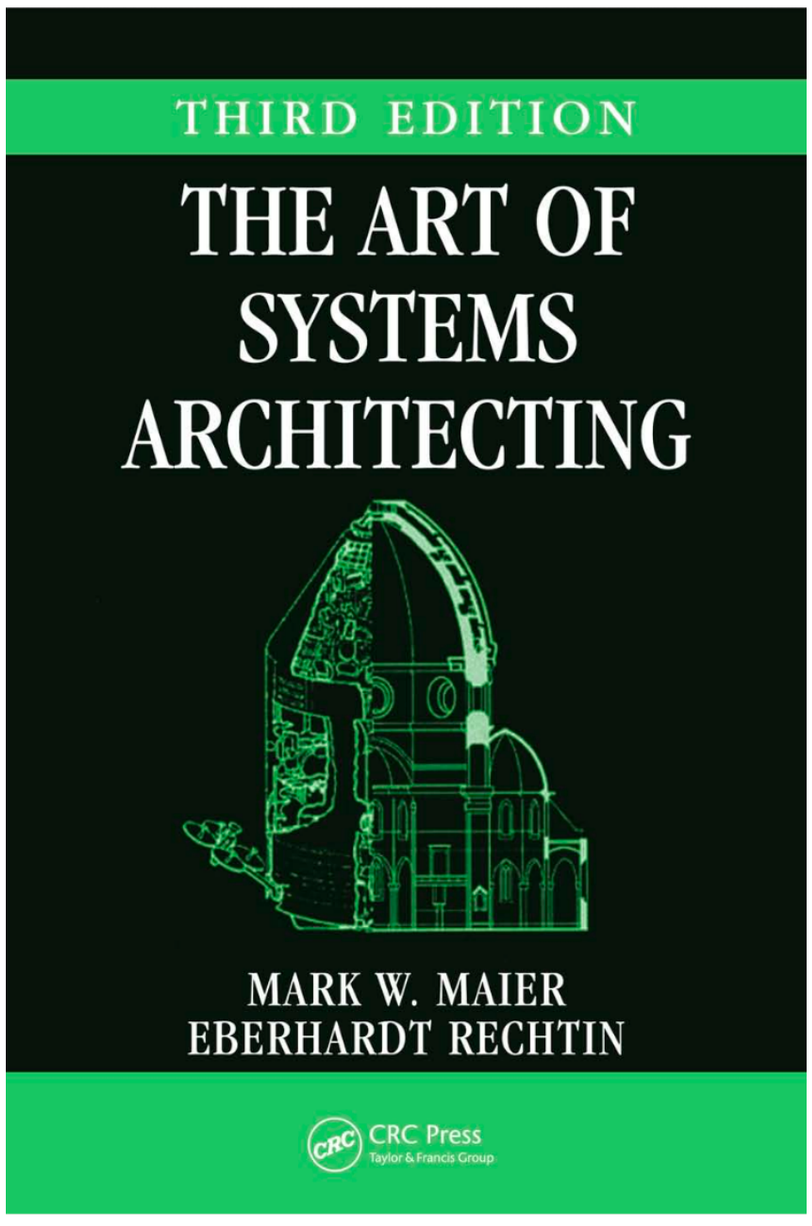
Qualities: Incremental Approach

zühlke
empowering ideas

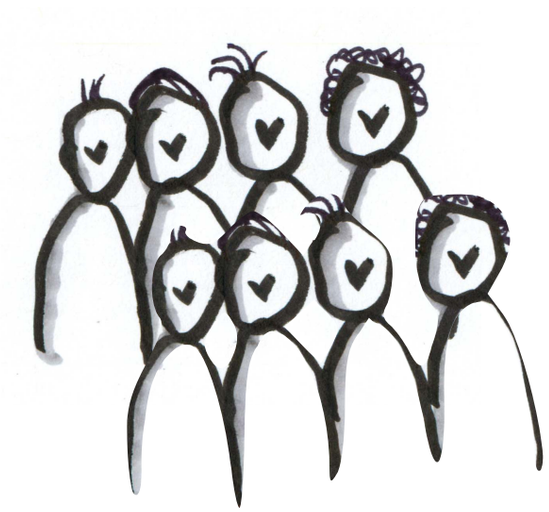
- Architectural Patterns
- Hoisting
- Prototyping

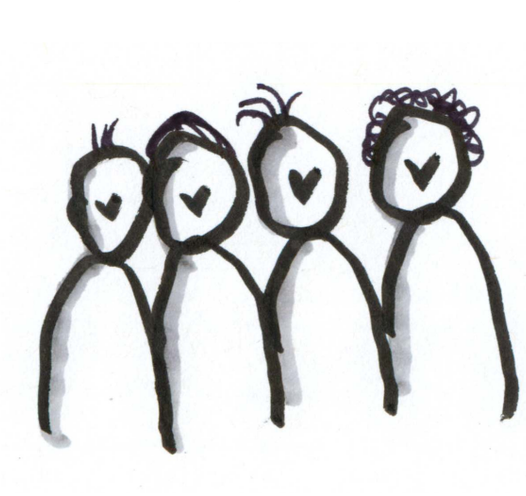
zühlke
empowering ideas

Getting started



"If the politics don't fly, the system never will"

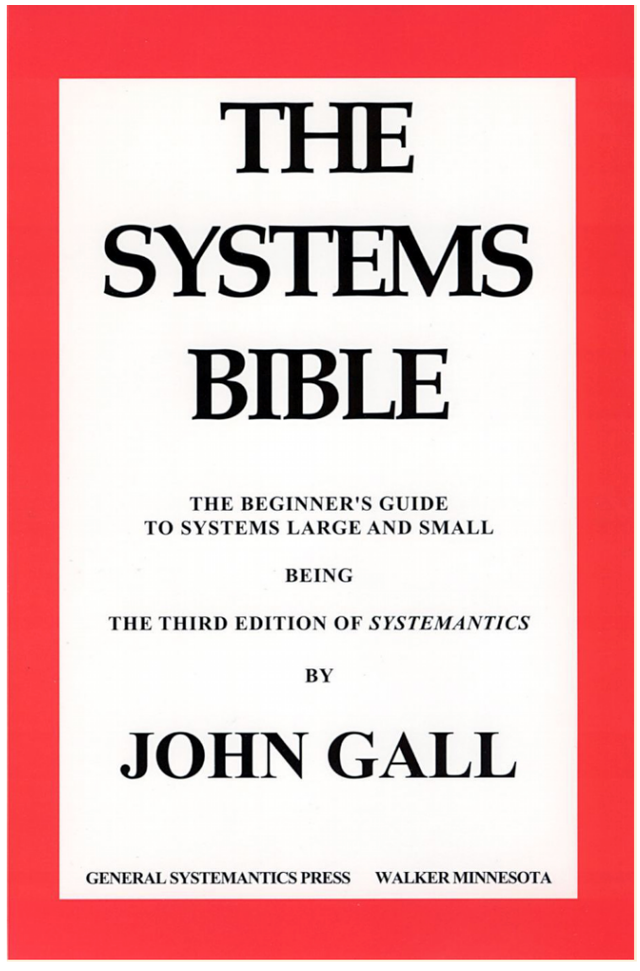




Engage with stakeholders and the development team from the start!

Start Small!

zühlke
empowering ideas

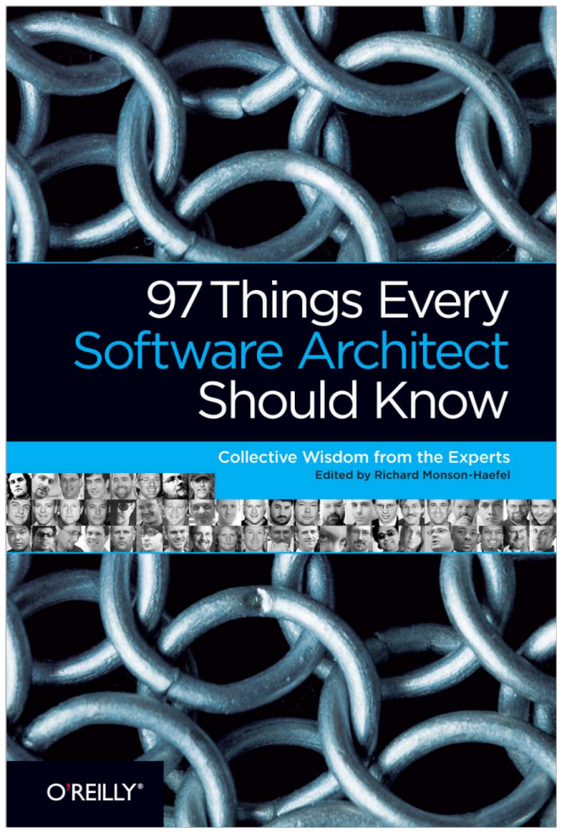


"A complex system that works is invariably found to have evolved from a simple system that worked."

Focus On What You Know

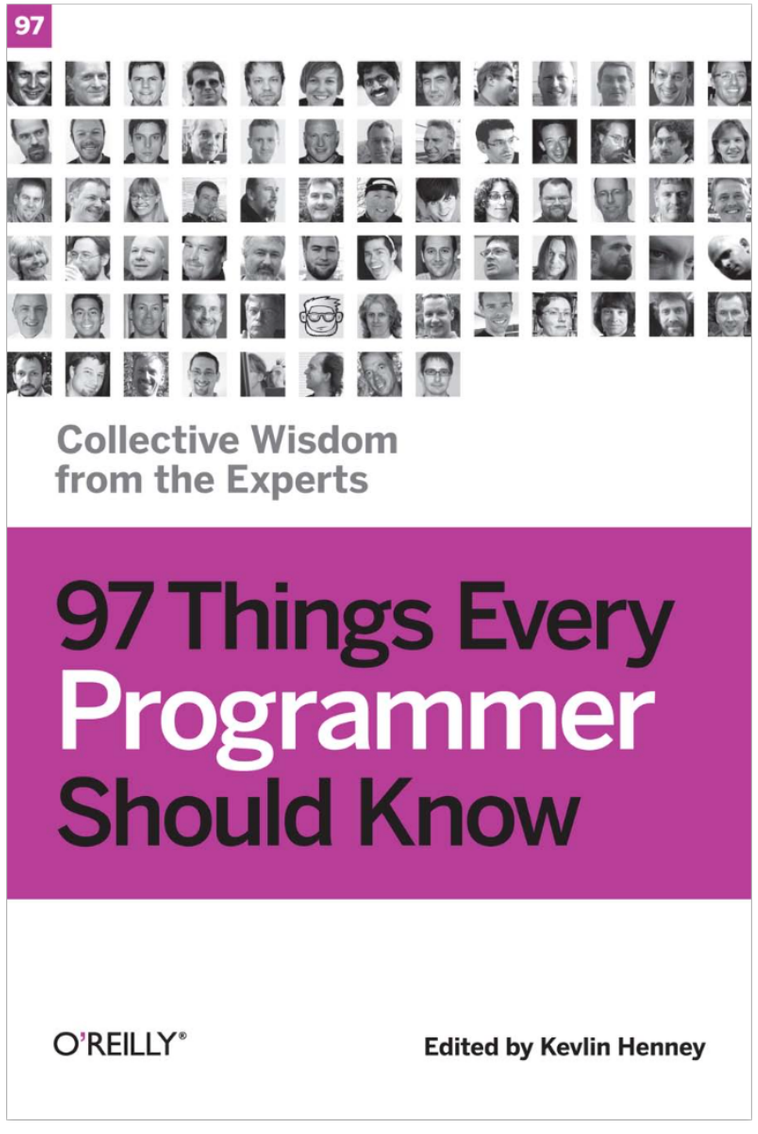
zühlke
empowering ideas

- No SUFD
- Sufficient design
- Remove “reusable” and “flexible” from your dictionary
- Address major risks



"Favoring simplicity before generality acts as a tiebreaker between otherwise equally viable design alternatives. When there are two possible solutions, favor the one that is simpler and based on concrete need rather than the more intricate one that boasts of generality."

Kevlin Henney



"Choose Your Tools With Care"

Giovanni Asproni

zühlke
empowering ideas

You may have an idea of the final shape of the system.

However, that’s just as an hypothesis

Test Your Hypothesis

zühlke
empowering ideas

"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 Reeves, "What Is Software Design?"

http://www.developerdotstar.com/mag/articles/reeves_design.html

Don’t Fall in love with your design!

Walking Skeleton

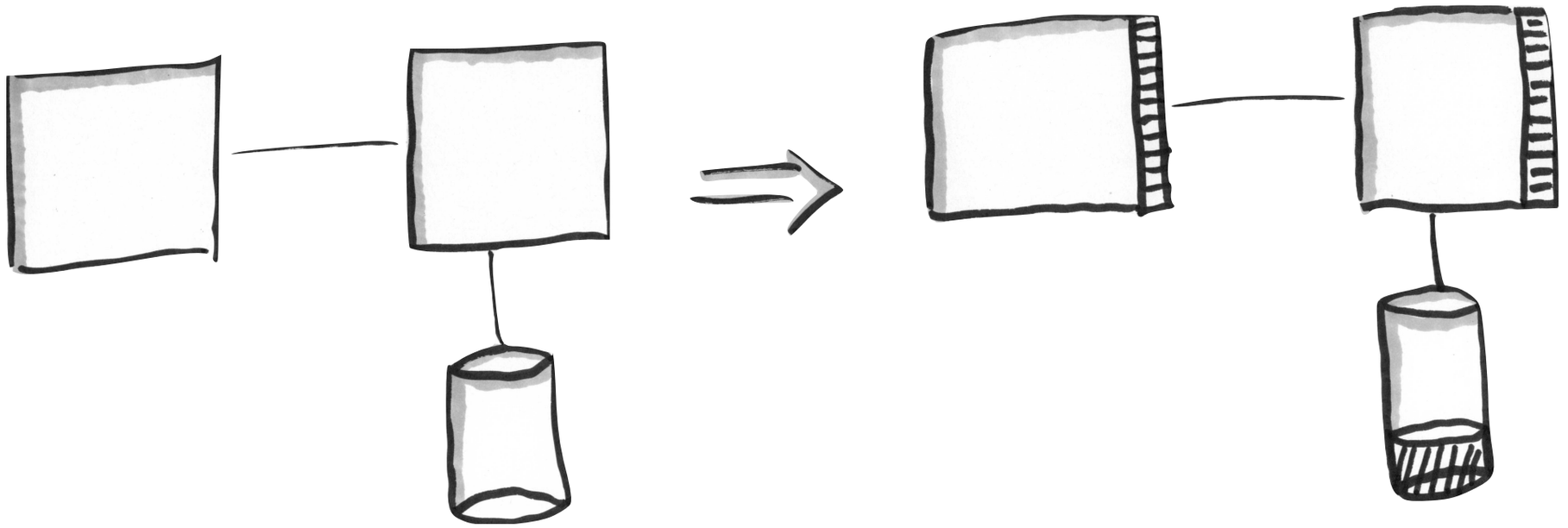
zühlke
empowering ideas

"is a tiny implementation of the system that performs a small end-to-end function. It need not use the final architecture, but it should link together the main architectural components. The architecture and the functionality can then evolve in parallel."

From: http://alistair.cockburn.us/Walking+skeleton

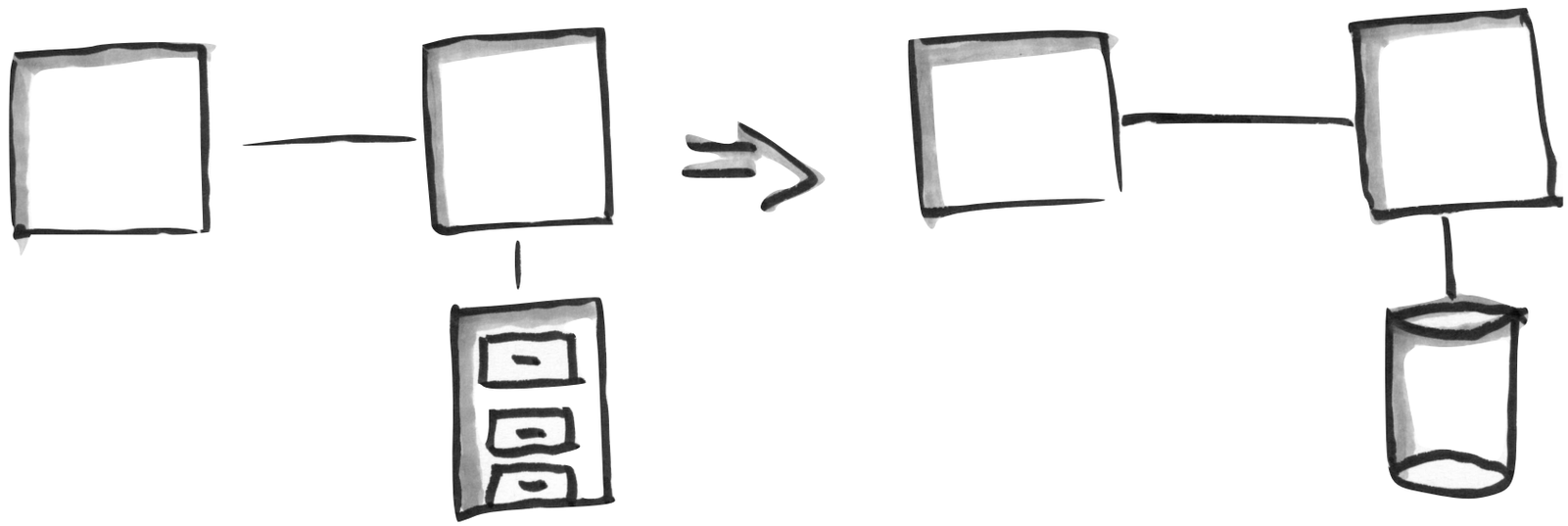
Walking Skeleton

zühlke
empowering ideas



Skeleton On Crutches

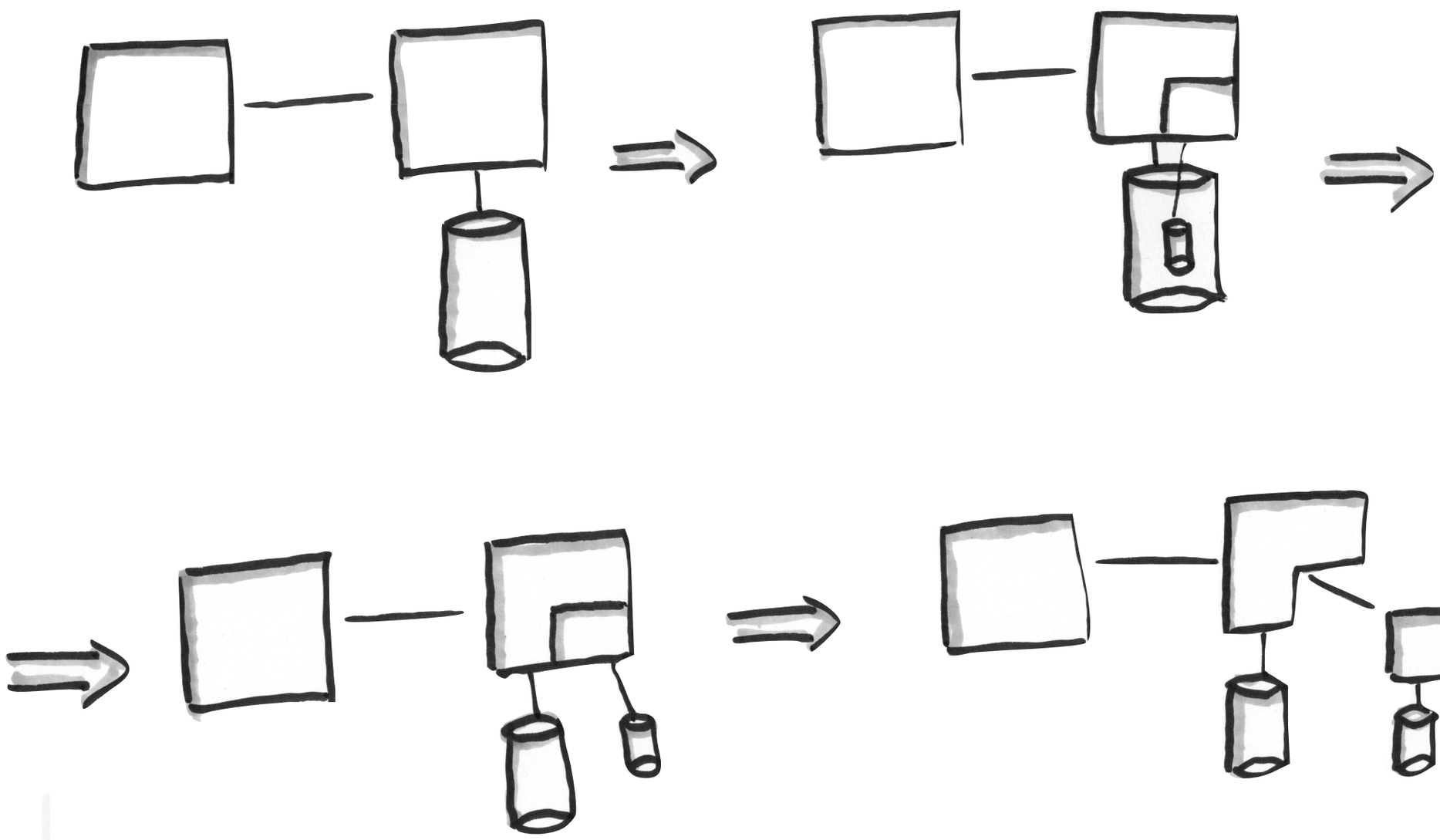
zühlke
empowering ideas



https://gojko.net/2014/06/09/forget-the-walking-skeleton-put-it-on-crutches/

Extracting A Service

zühlke
empowering ideas



zühlke
empowering ideas

Design for testability

zühlke
empowering ideas

The main purpose of automated tests is to make the system maintainable and evolvable

zühlke
empowering ideas

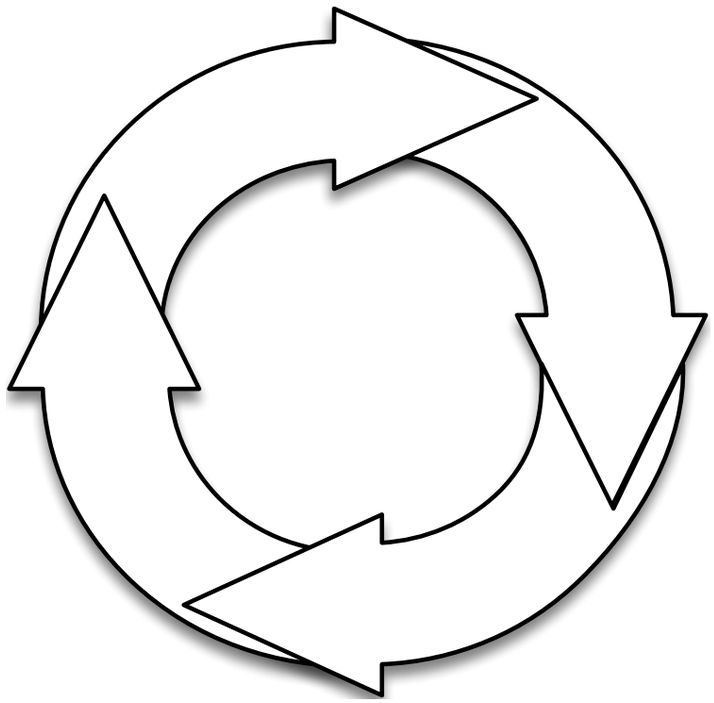
"When testing is done right, it is almost unreasonably effective at uncovering defects that matter."

John Regehr

http://blog.regehr.org/archives/820

Create Feedback Loops

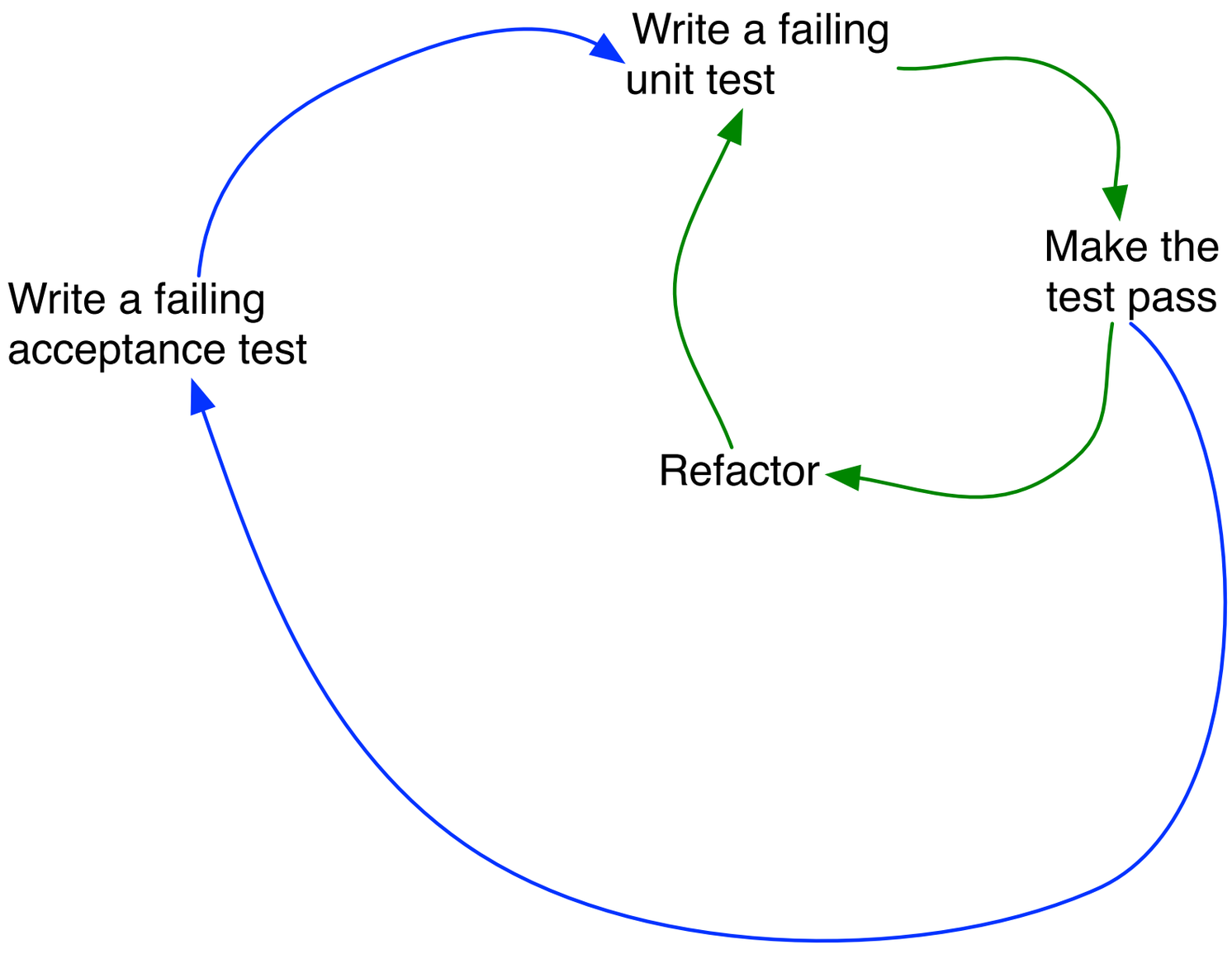
zühlke
empowering ideas



- Continuous Integration
- Continuous Deployment
- Continuous Delivery

TDD at all levels helps

zühlke
empowering ideas

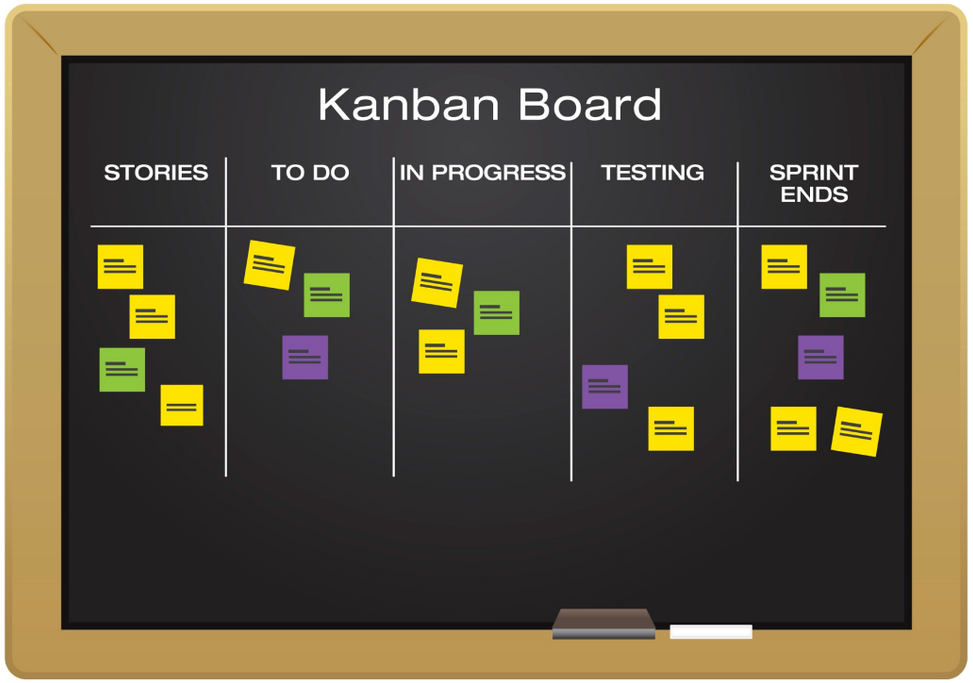


Often Forgotten Things

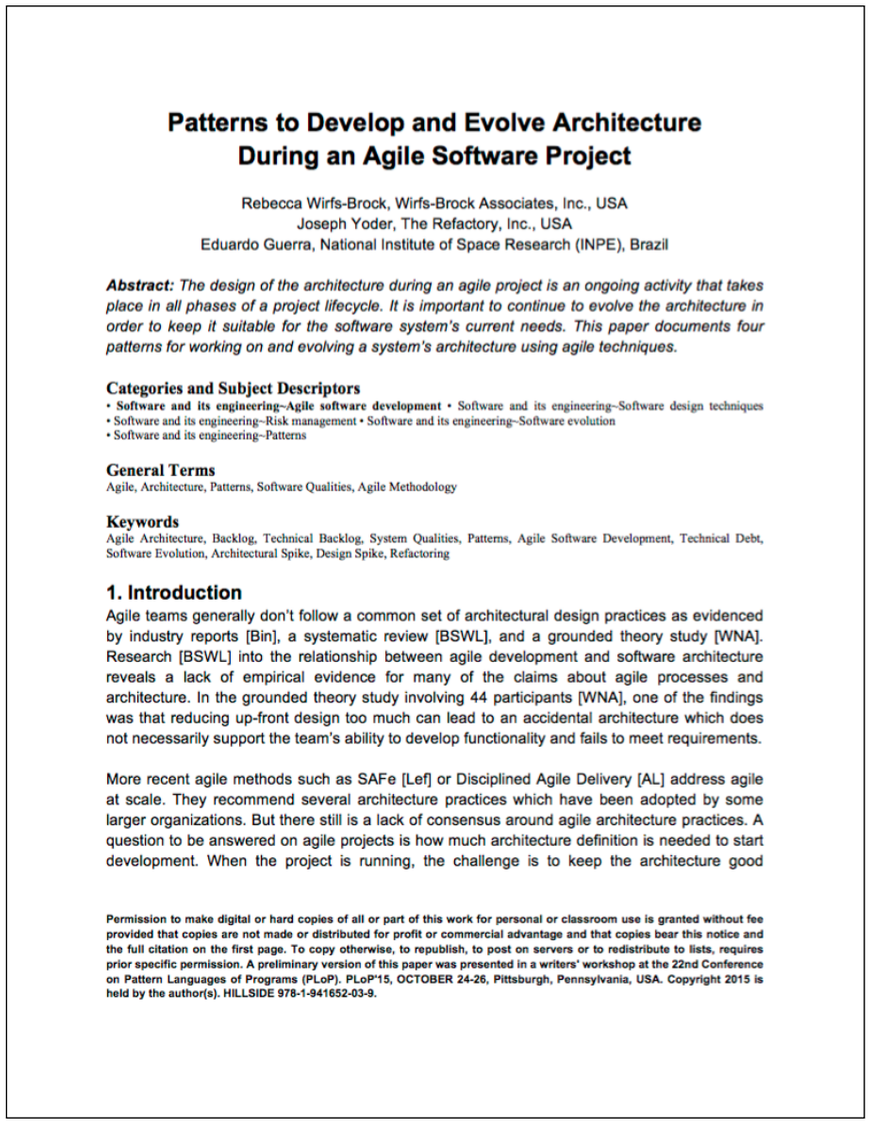
zühlke
empowering ideas

- Buildability
- Deployability
- Configurability
- Versioning strategies

zühlke
empowering ideas



How can I include architectural tasks in my workflow?



Patterns to Develop and Evolve Architecture During an Agile Software Project

Rebecca Wirfs-Brock, Wirfs-Brock Associates, Inc., USA
Joseph Yoder, The Refactory, Inc., USA
Eduardo Guerra, National Institute of Space Research (INPE), Brazil

Abstract: *The design of the architecture during an agile project is an ongoing activity that takes place in all phases of a project lifecycle. It is important to continue to evolve the architecture in order to keep it suitable for the software system's current needs. This paper documents four patterns for working on and evolving a system's architecture using agile techniques.*

Categories and Subject Descriptors
• **Software and its engineering~Agile software development** • Software and its engineering~Software design techniques • Software and its engineering~Risk management • Software and its engineering~Software evolution • Software and its engineering~Patterns

General Terms
Agile, Architecture, Patterns, Software Qualities, Agile Methodology

Keywords
Agile Architecture, Backlog, Technical Backlog, System Qualities, Patterns, Agile Software Development, Technical Debt, Software Evolution, Architectural Spike, Design Spike, Refactoring

1. Introduction

Agile teams generally don't follow a common set of architectural design practices as evidenced by industry reports [Bin], a systematic review [BSWL], and a grounded theory study [WNA]. Research [BSWL] into the relationship between agile development and software architecture reveals a lack of empirical evidence for many of the claims about agile processes and architecture. In the grounded theory study involving 44 participants [WNA], one of the findings was that reducing up-front design too much can lead to an accidental architecture which does not necessarily support the team's ability to develop functionality and fails to meet requirements.

More recent agile methods such as SAFe [Lef] or Disciplined Agile Delivery [AL] address agile at scale. They recommend several architecture practices which have been adopted by some larger organizations. But there still is a lack of consensus around agile architecture practices. A question to be answered on agile projects is how much architecture definition is needed to start development. When the project is running, the challenge is to keep the architecture good

Patterns

- Architecture in the Backlog
- Architectural Trigger
- Architectural Spike
- Technical Debt Management

Documenting The Design

A Rational Design Process: How and Why to Fake It

DAVID LORGE PARNAS AND PAUL C. CLEMENTS

Abstract—**Many have sought a software design process that allows a program to be derived systematically from a precise statement of requirements. This paper proposes that, although we will not succeed in designing a real product in that way, we can produce documentation that makes it appear that the software was designed by such a process. We first describe the ideal process, and the documentation that it requires. We then explain why one should attempt to design according to the ideal process and why one should produce the documentation that would have been produced by that process. We describe the contents of each of the required documents.**

Index Terms—**Programming methods, software design, software documentation, software engineering.**

I. The Search for the Philosopher's Stone: Why Do We Want a Rational Design Process?

A PERFECTLY rational person is one who always has a good reason for what he does. Each step taken can be shown to be the best way to get to a well defined goal. Most of us like to think of ourselves as rational professionals. However, to many observers, the usual process of designing software appears quite irrational. Programmers start without a clear statement of desired behavior and implementation constraints. They make a long sequence of design decisions with no clear statement of why they do things the way they do. Their rationale is rarely explained.

Many of us are not satisfied with such a design process. That is why there is research in software design, programming methods, structured programming, and related topics. Ideally, we would like to derive our programs from a statement of requirements in the same sense that theorems are derived from axioms in a published proof. All of the methodologies that can be considered "top down" are the result of our desire to have a rational, systematic way of designing software.

This paper brings a message with both bad news and good news. The bad news is that, in our opinion, we will never find the philosopher's stone. We will never find a process that allows us to design software in a perfectly rational way. The good news is that we can fake it. We can present our system to others as if we had been rational designers and it pays to pretend do so during development and maintenance.

Manuscript received March 18, 1985. This work was supported by the U.S. Navy and by the National Science and Engineering Research Council (NSERC) of Canada.

D. L. Parnas is with the Department of Computer Science, University of Victoria, Victoria, B. C. V8W 2Y2, Canada, and the Computer Science and Systems Branch, Naval Research Laboratory, Washington, DC 20375.

P. C. Clements is with the Computer Science and Systems Branch, Naval Research Laboratory, Washington, DC 20375.

IEEE Log Number 8405736.

II. Why Will a Software Design "Process" Always Be an Idealization?

We will never see a software project that proceeds in the "rational" way. Some of the reasons are listed below:

1) In most cases the people who commission the building of a software system do not know exactly what they want and are unable to tell us all that they know.

2) Even if we knew the requirements, there are many other facts that we need to know to design the software. Many of the details only become known to us as we progress in the implementation. Some of the things that we learn invalidate our design and we must backtrack. Because we try to minimize lost work, the resulting design may be one that would not result from a rational design process.

3) Even if we knew all of the relevant facts before we started, experience shows that human beings are unable to comprehend fully the plethora of details that must be taken into account in order to design and build a correct system. The process of designing the software is one in which we attempt to separate concerns so that we are working with a manageable amount of information. However, until we have separated the concerns, we are bound to make errors.

4) Even if we could master all of the detail needed, all but the most trivial projects are subject to change for external reasons. Some of those changes may invalidate previous design decisions. The resulting design is not one that would have been produced by a rational design process.

5) Human errors can only be avoided if one can avoid the use of humans. Even after the concerns are separated, errors will be made.

6) We are often burdened by preconceived design ideas, ideas that we invented, acquired on related projects, or heard about in a class. Sometimes we undertake a project in order to try out or use a favorite idea. Such ideas may not be derived from our requirements by a rational process.

7) Often we are encouraged, for economic reasons, to use software that was developed for some other project. In other situations, we may be encouraged to share our software with another ongoing project. The resulting software may not be the ideal software for either project, i.e., not the software that we would develop based on its requirements alone, but it is good enough and will save effort.

For all of these reasons, the picture of the software designer deriving his design in a rational, error-free way from a statement of requirements is quite unrealistic. No

zühlke
empowering ideas

Growing The System

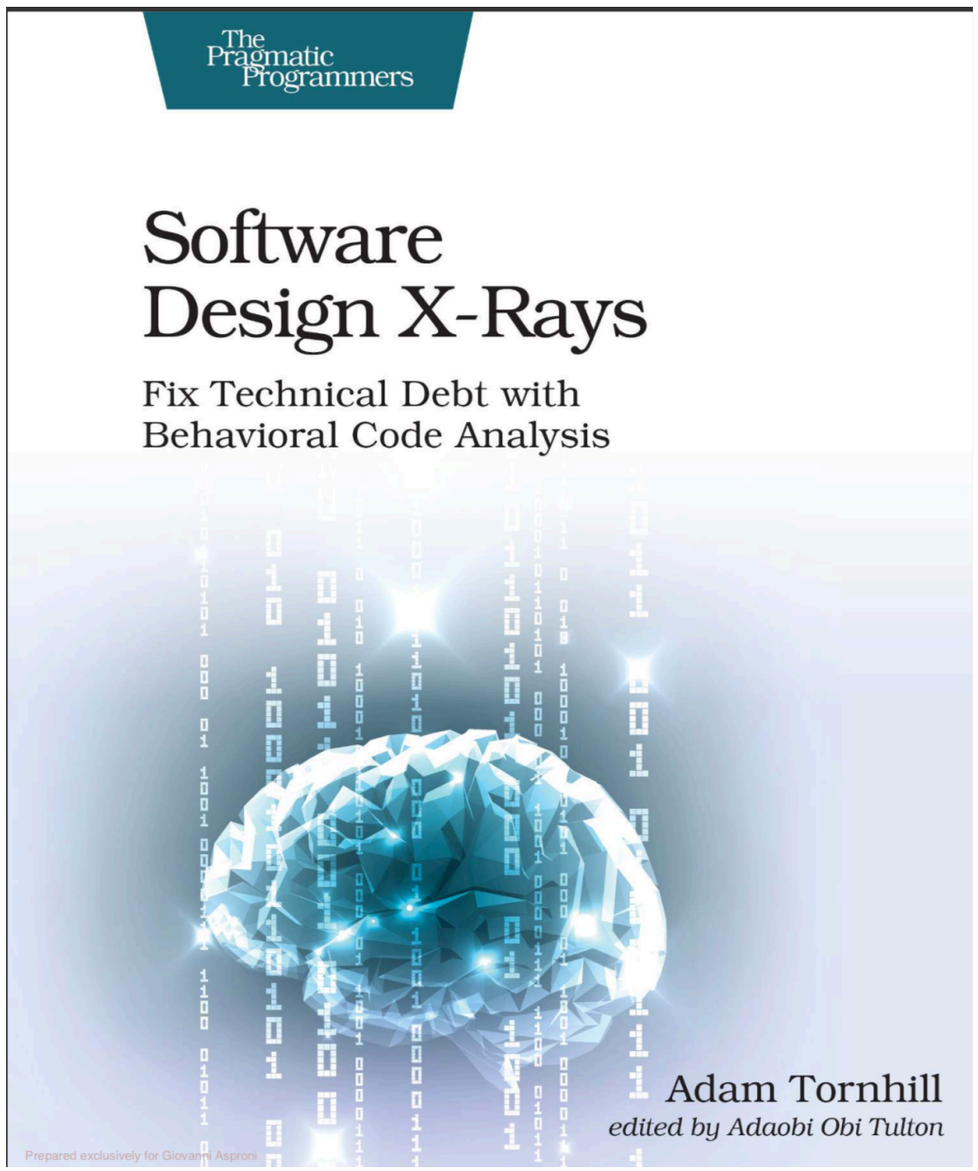
When The System Grows…

zühlke
empowering ideas

- Increasing team size
- Increasing the number of teams
- Feature teams or component teams?

Brownfield And Legacy Systems

zühlke
empowering ideas



- Rewrite from scratch rarely works
- The incremental approach still apply

zühlke
empowering ideas

Thank You!

giovanni.asproni@zuhlke.com

@gasproni

Books And References

zühlke
empowering ideas