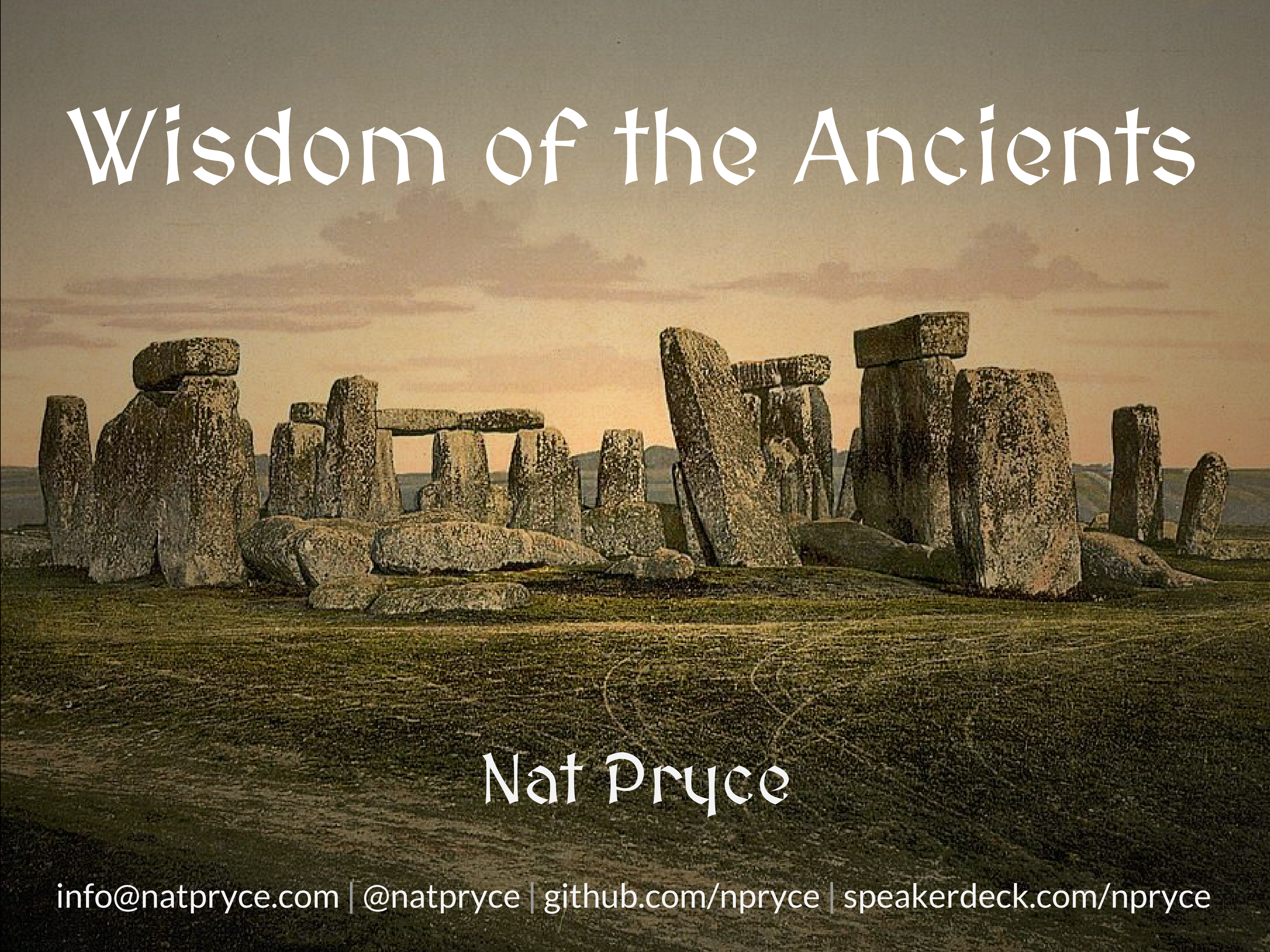
Wisdom of the Ancients



Nat Pryce

info@natpryce.com | @natpryce | github.com/npryce | speakerdeck.com/npryce

What it was like when I started

I spent very little of my time writing code

IBM SYSTEM 360

… any program is *a model of a model within a theory of a model of an abstraction of some portion of the world or of some universe of discourse.*

–Manny Lehman

Programs, Life Cycles, and Laws of Evolution. 1980

Lehman's categories of software system

S-type	formally defined by and derivable from a specification
P-type	solves a real-world problem but does not affect the world it models
E-type	embedded in the world it models; its operation changes that world

Law of Continuous Change

> Any software system used in the real-world must change or become less and less useful in that environment.

Law of Increasing Complexity

> As a system evolves, its complexity increases unless work is done to maintain or reduce it.

–Manny Lehman (1974, ...)

Evolution processes [of software systems] constitute multi level, multi loop, multi agent feedback systems

–Manny Lehman (1974, ...)

Principle of Uncertainty

The outcome, in the real world, of software system operation is inherently uncertain with the precise area of uncertainty also unknown

–Manny Lehman (1989)

… conceptual integrity is the most important consideration in system design.

It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.

–Fred Brooks

The Mythical Man Month. 1975

How can we apply Lehman's insights?

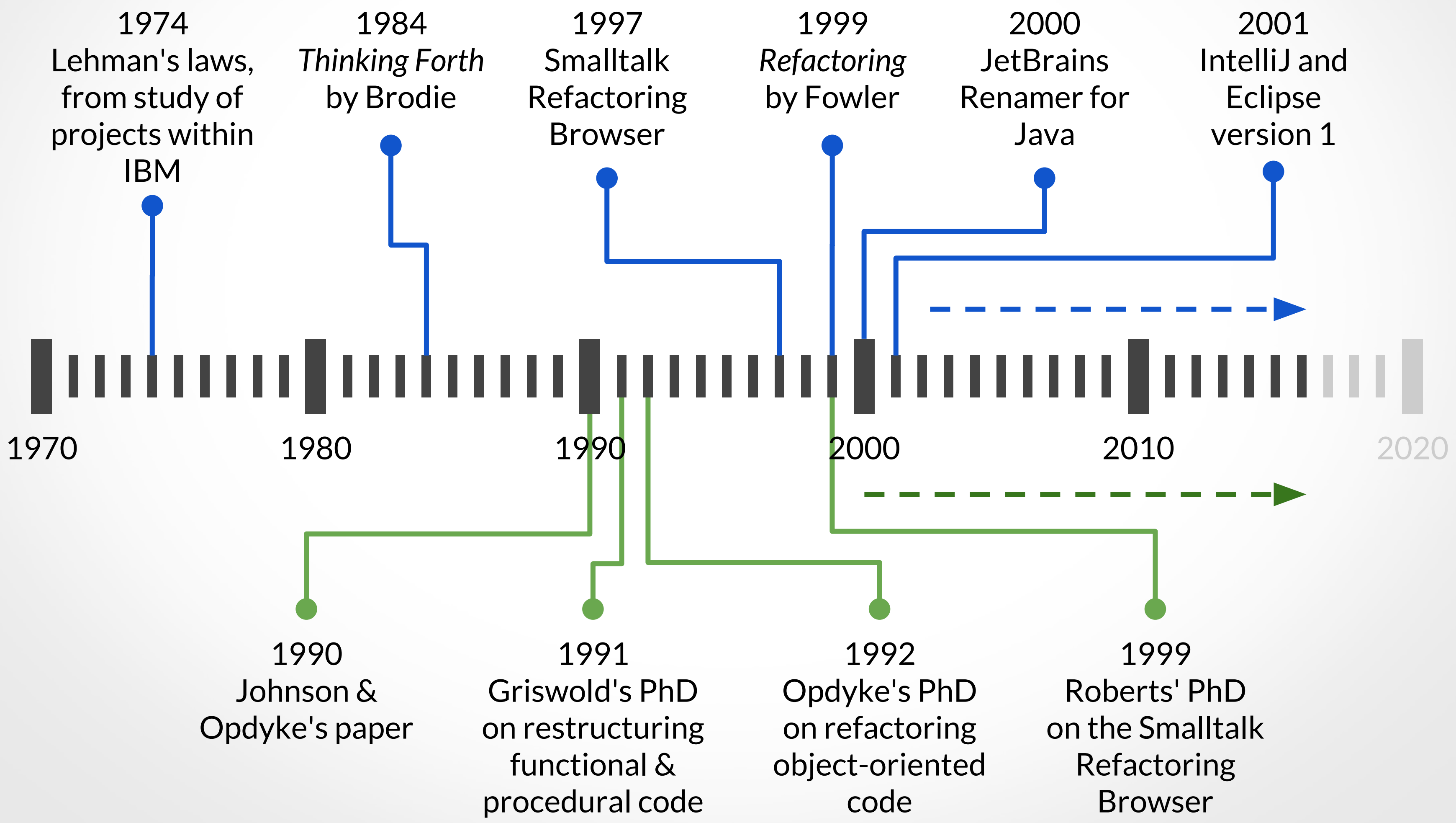
1

Consider the system type when evaluating a technique or technology

S-programs are … the programming form from which most advanced programming methodology and related techniques derive.

–Manny Lehman (1980)

Refactoring tools: a prehistory


1974
Lehman's laws, from study of projects within IBM
1984
Thinking Forth by Brodie
1997
Smalltalk Refactoring Browser
1999
Refactoring by Fowler
2000
JetBrains Renamer for Java
2001
IntelliJ and Eclipse version 1
1970
1980
1990
2000
2010
2020
1990
Johnson & Opdyke's paper
1991
Griswold's PhD on restructuring functional & procedural code
1992
Opdyke's PhD on refactoring object-oriented code
1999
Roberts' PhD on the Smalltalk Refactoring Browser

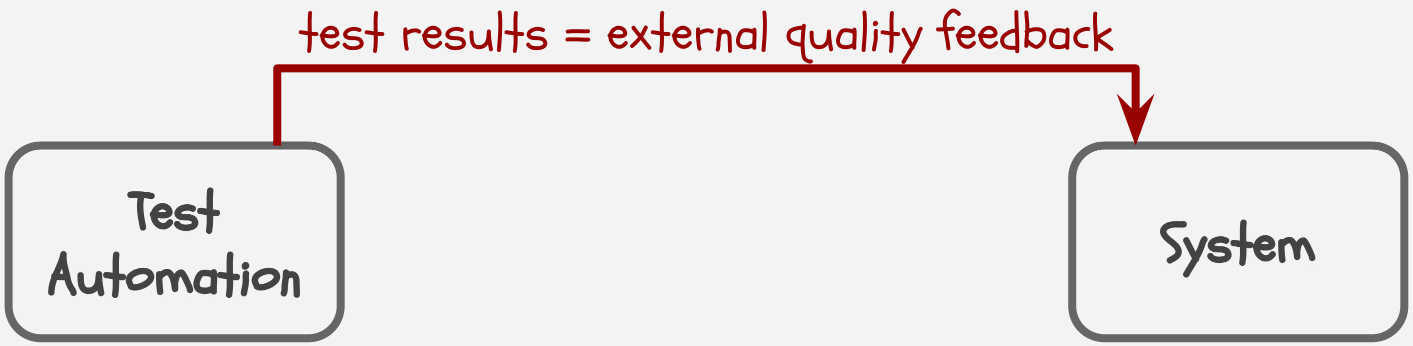
…as programming methodology evolves still further, all large programs (software systems) will be constructed as structures of S-programs.

–Manny Lehman (1980)

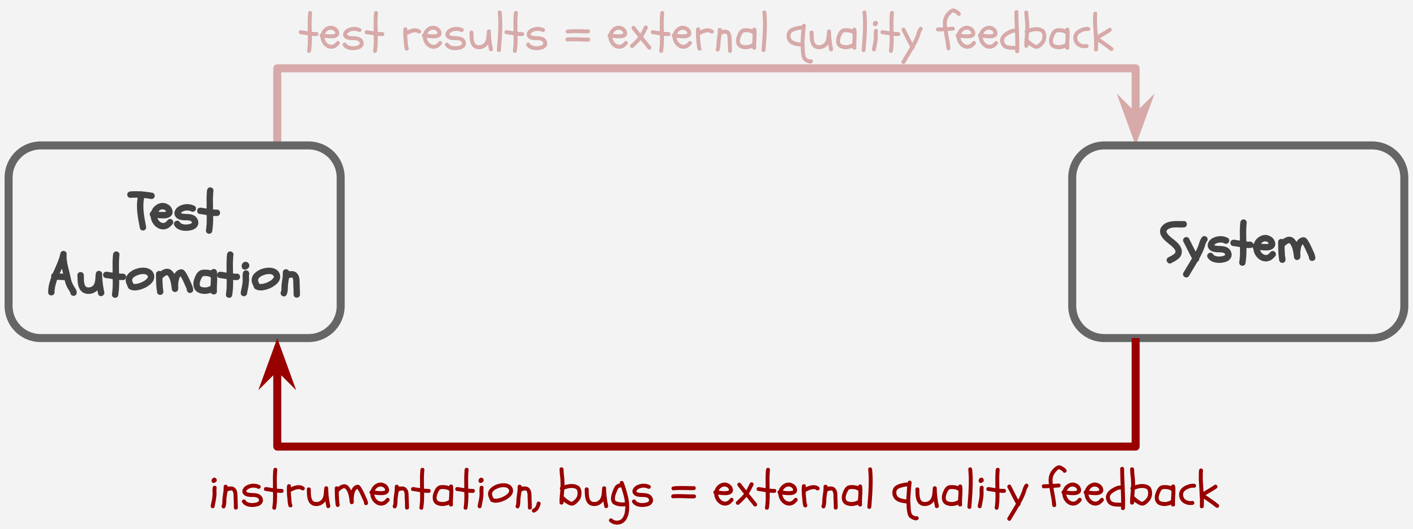
2

Nurture your feedback cycles

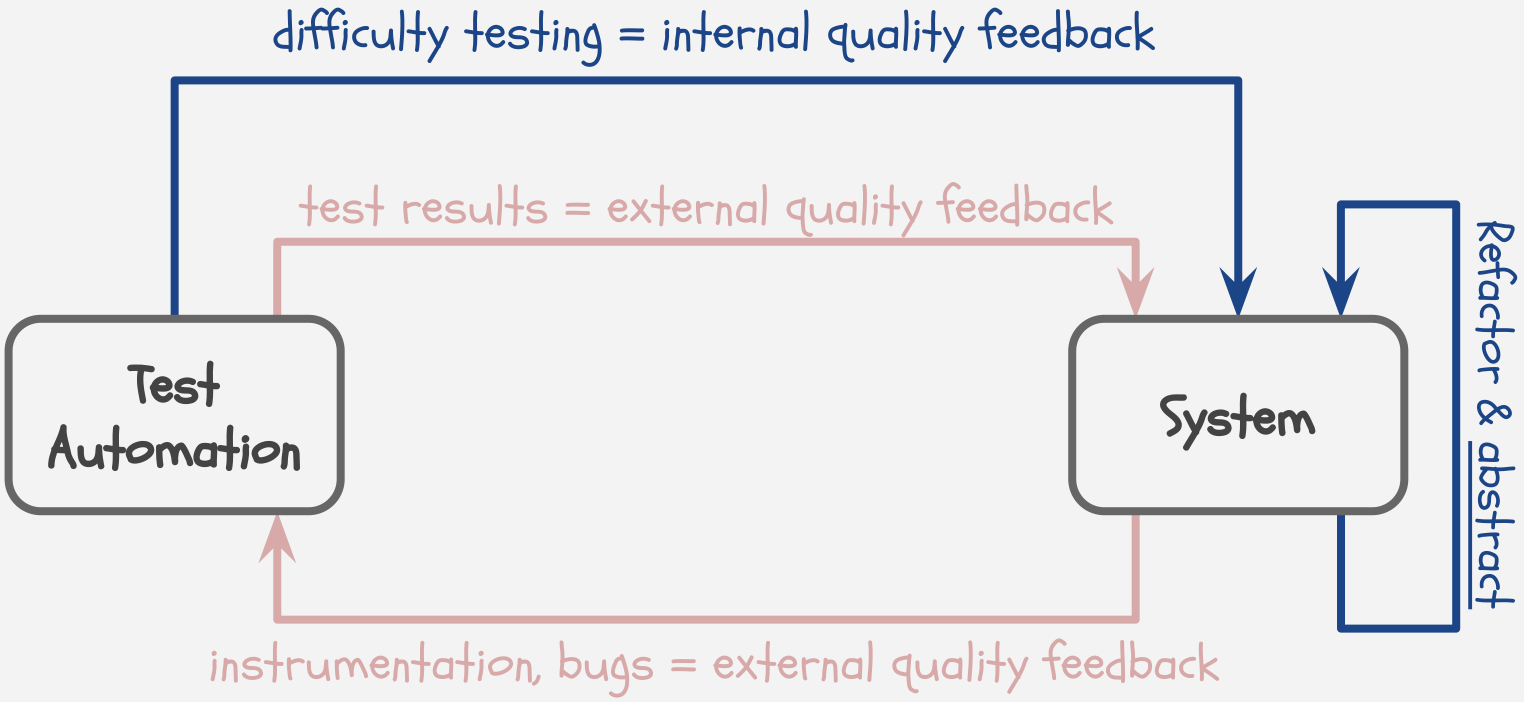
How good is the system?



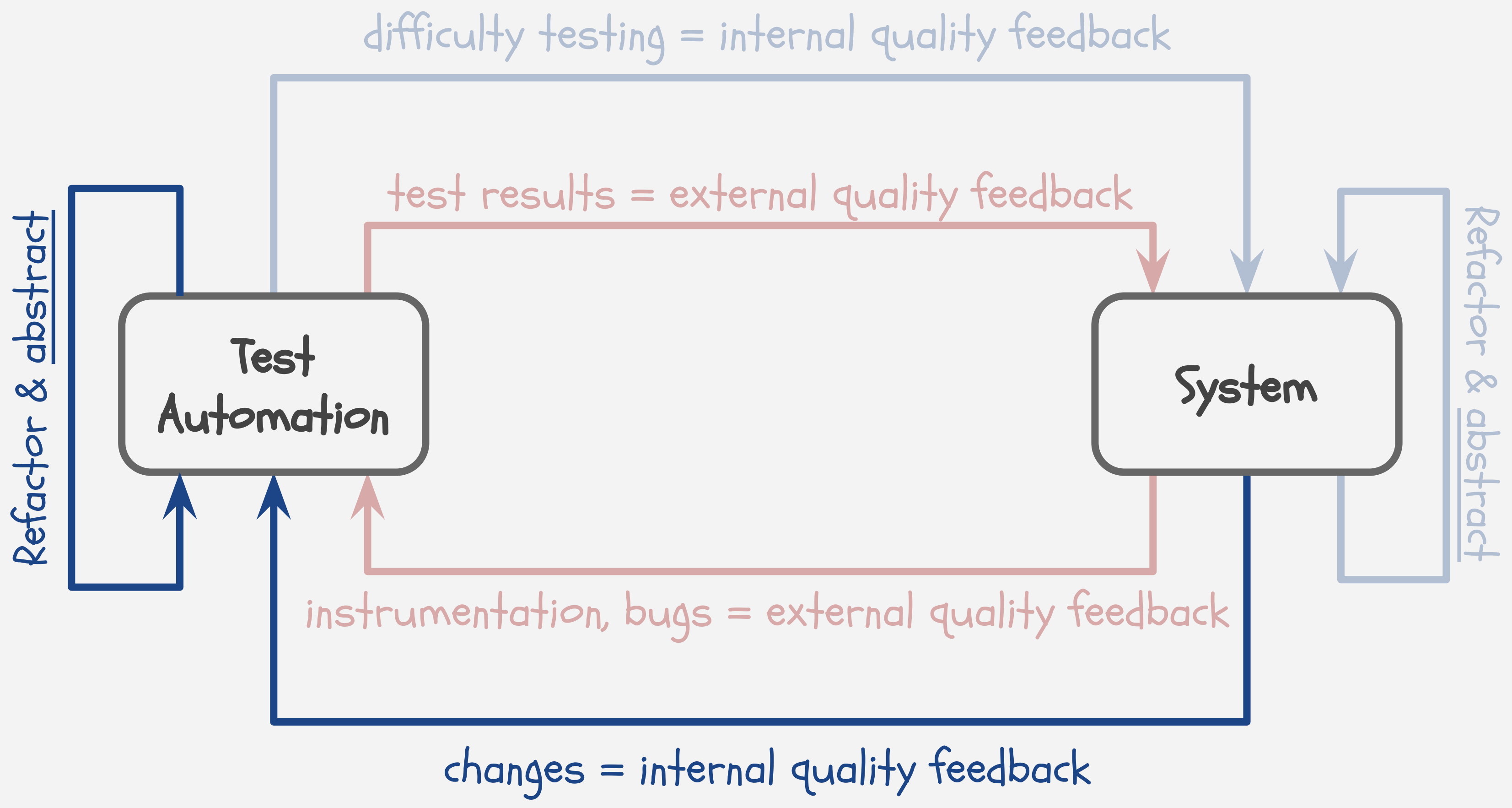
How good are the tests?


test results = external quality feedback
Test Automation
System
instrumentation, bugs = external quality feedback

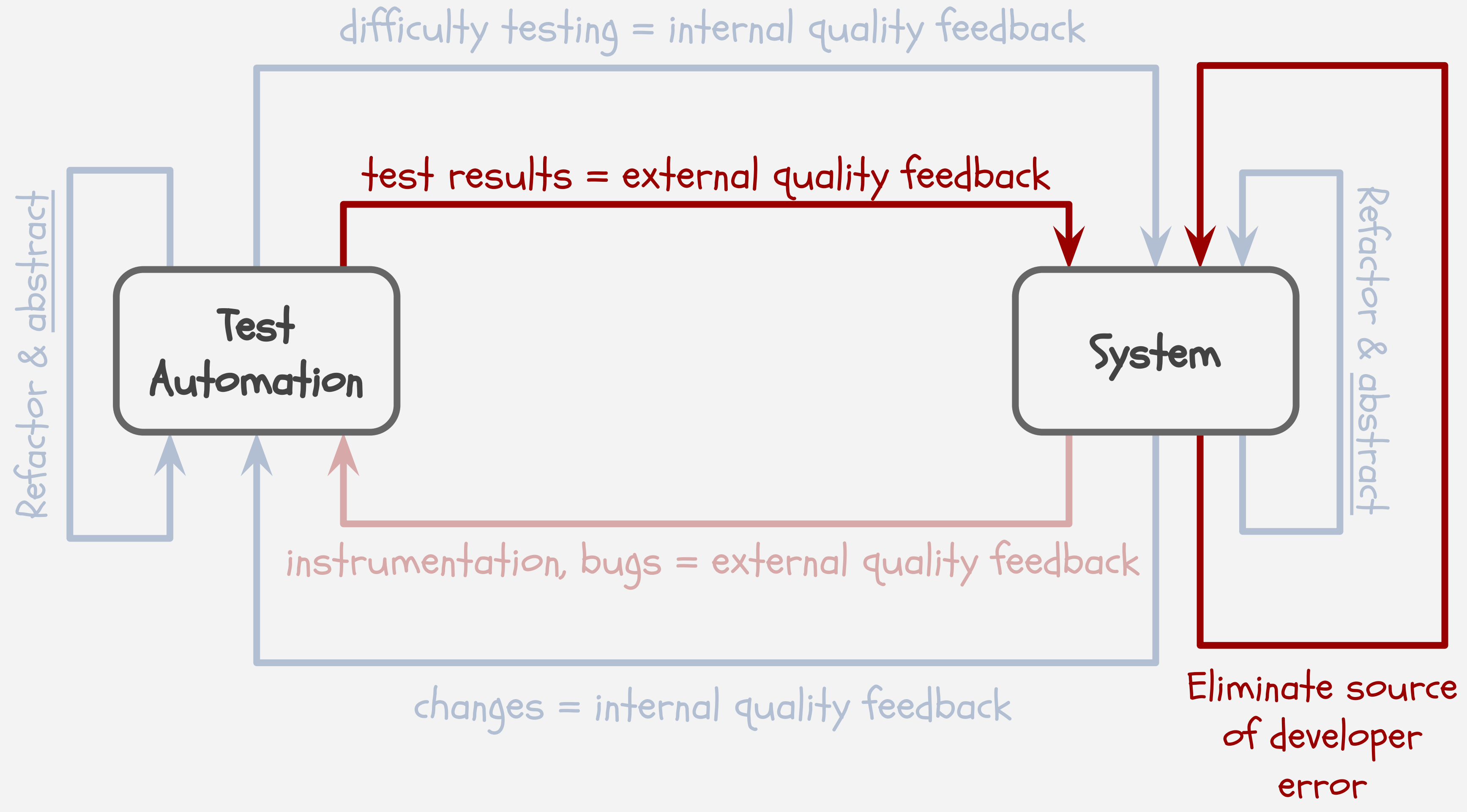
How maintainable is the software?


difficulty testing = internal quality feedback
test results = external quality feedback
Test Automation
System
Refactor & abstract
instrumentation, bugs = external quality feedback

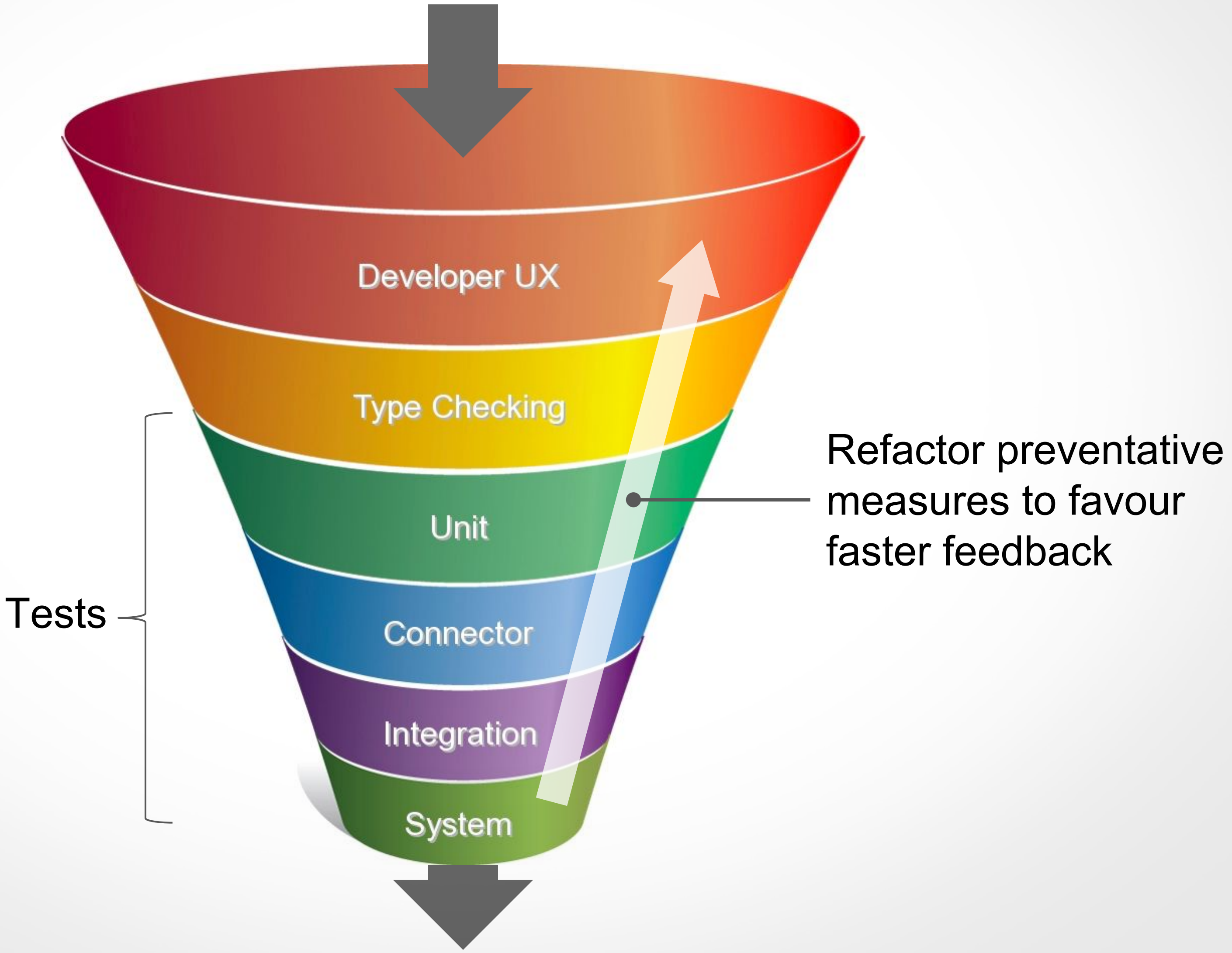
How maintainable are the tests?


difficulty testing = internal quality feedback
test results = external quality feedback
Refactor & abstract
Test Automation
System
Refactor & abstract
instrumentation, bugs = external quality feedback
changes = internal quality feedback

Can we eliminate the need for tests?


difficulty testing = internal quality feedback
test results = external quality feedback
Refactor & abstract
Test Automation
System
Refactor & abstract
instrumentation, bugs = external quality feedback
changes = internal quality feedback
Eliminate source of developer error

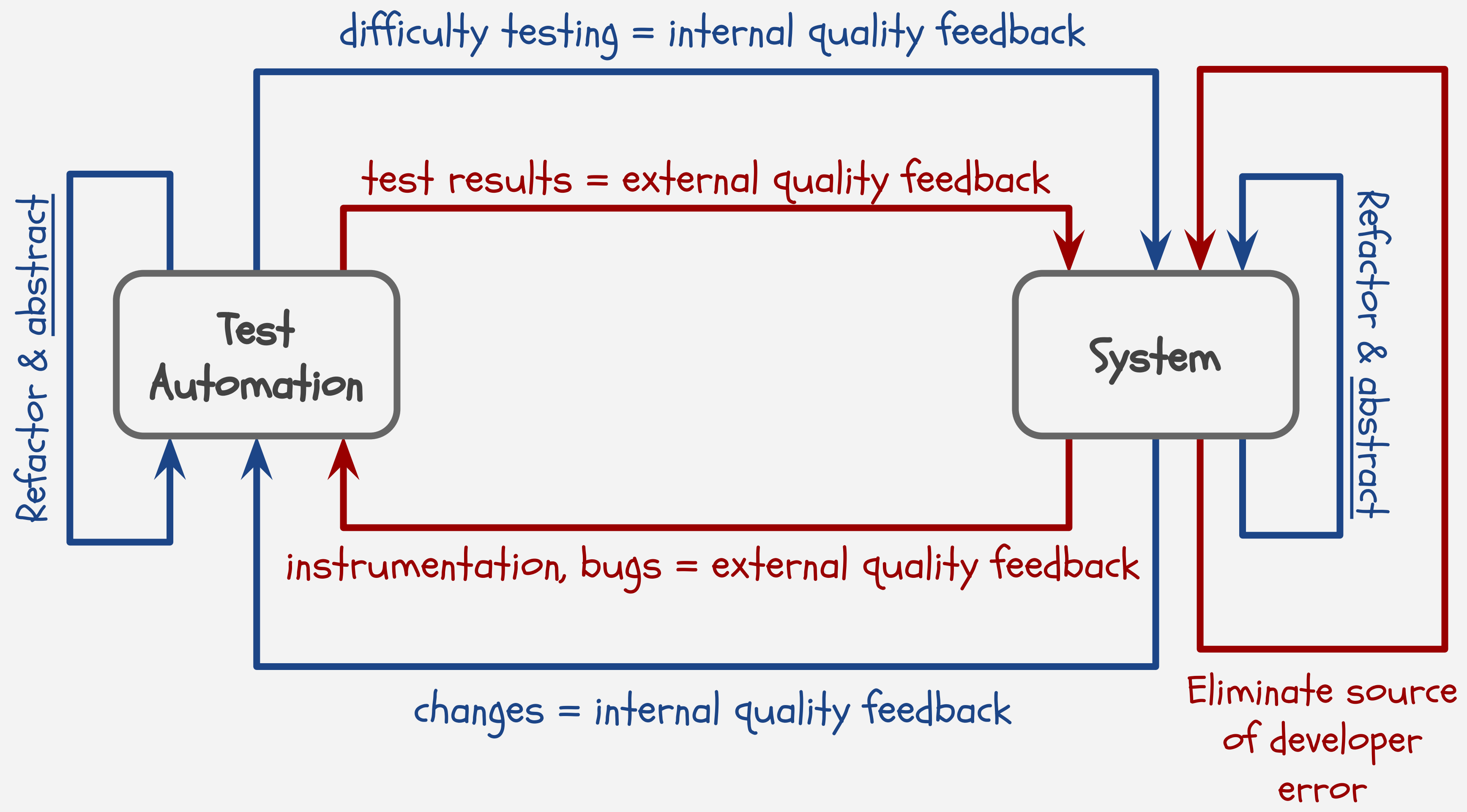
The Funnel of Feedback



Funnel diagram by http://unitydc.co.uk/funnels

Types instead of Tests?!?!

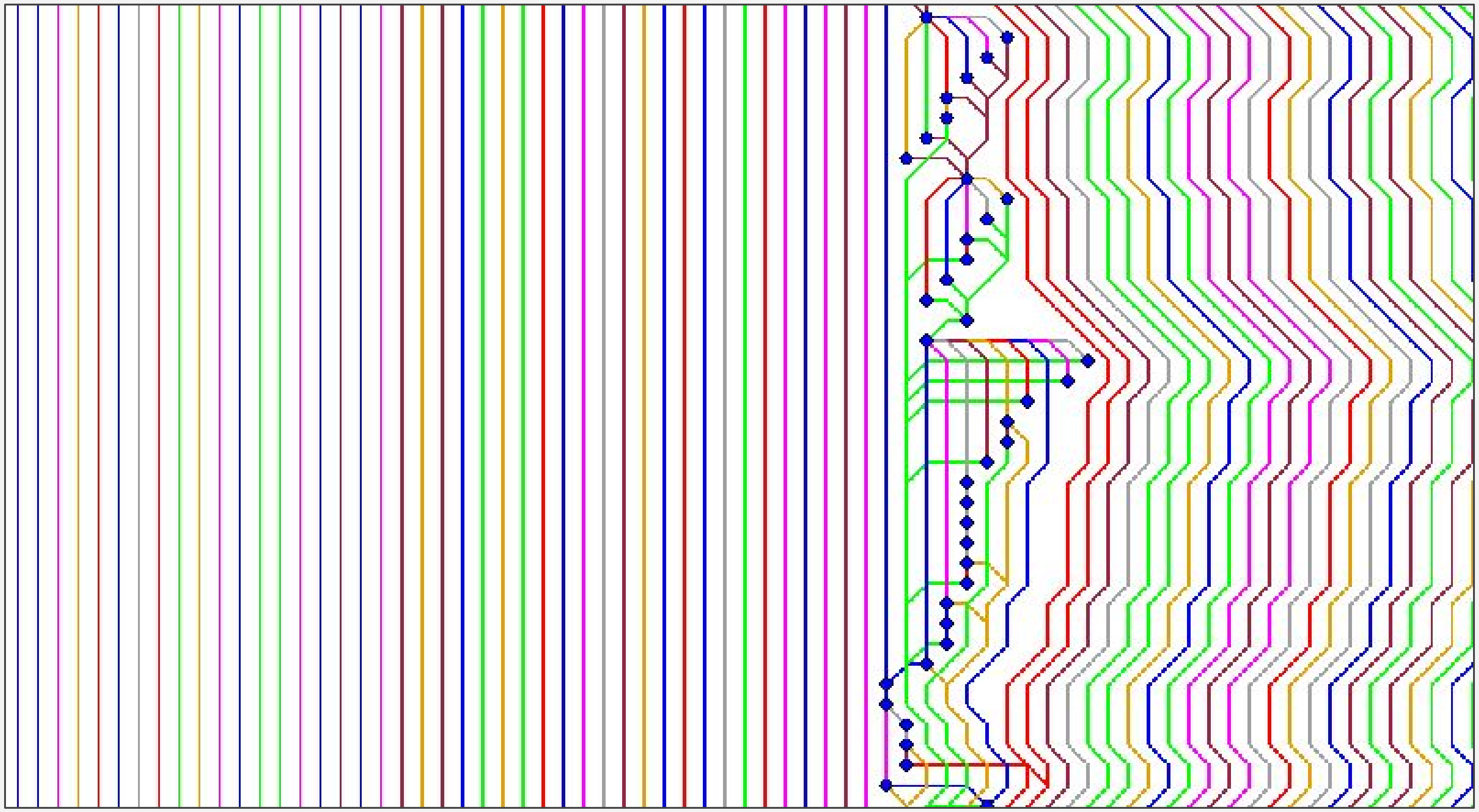
A cybernetic system


difficulty testing = internal quality feedback
test results = external quality feedback
Refactor & abstract
Test Automation
System
Refactor & abstract
instrumentation, bugs = external quality feedback
changes = internal quality feedback
Eliminate source of developer error

3

Accept uncertainty

Why are developers uncomfortable with design as continual, gradual, never-ending adaption?

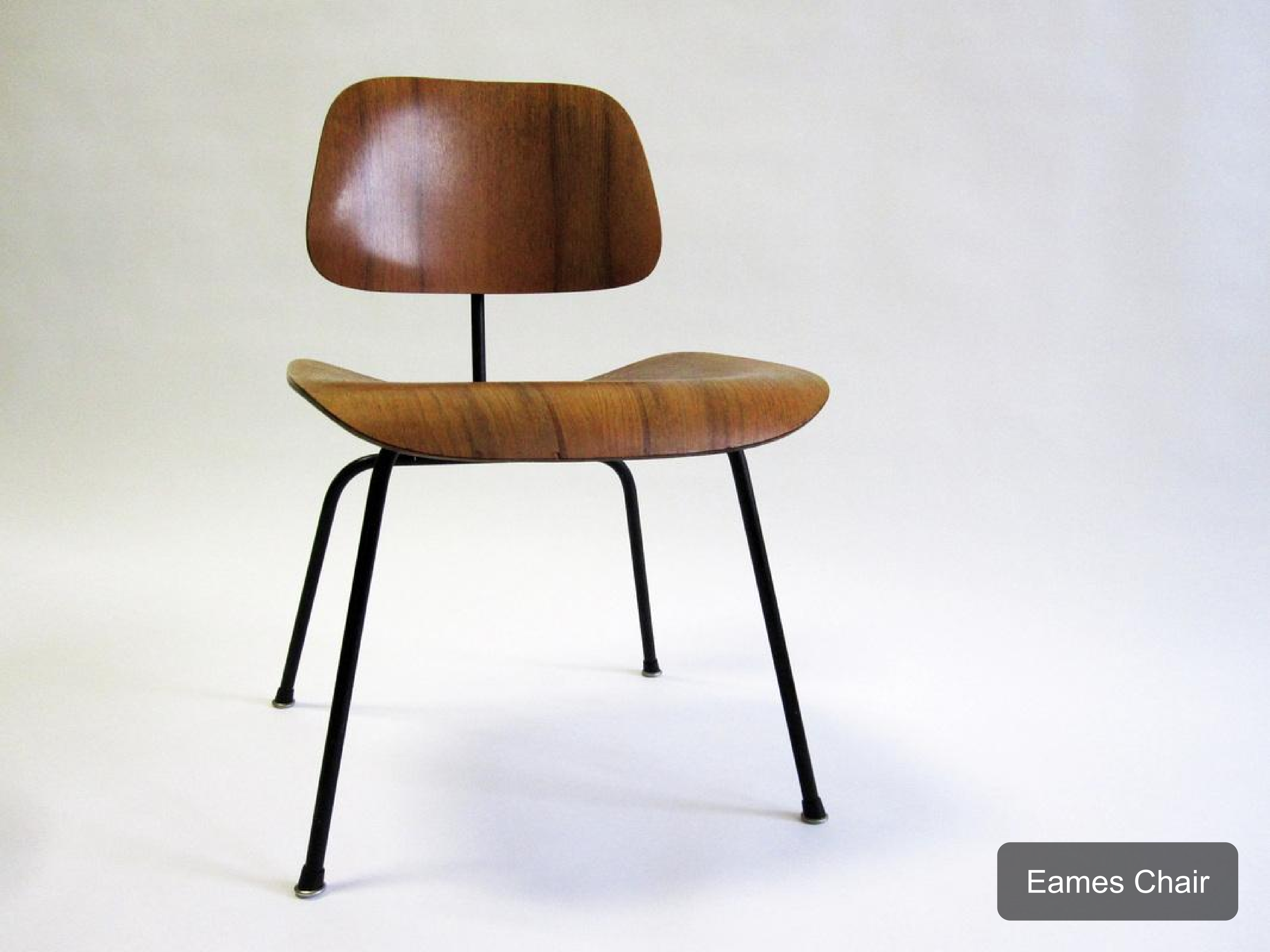


Weltanschauung

Modernism

Modernist [styles] shared certain underlying principles: a rejection of history and applied ornament; a preference for abstraction; and a belief that design and technology could transform society.

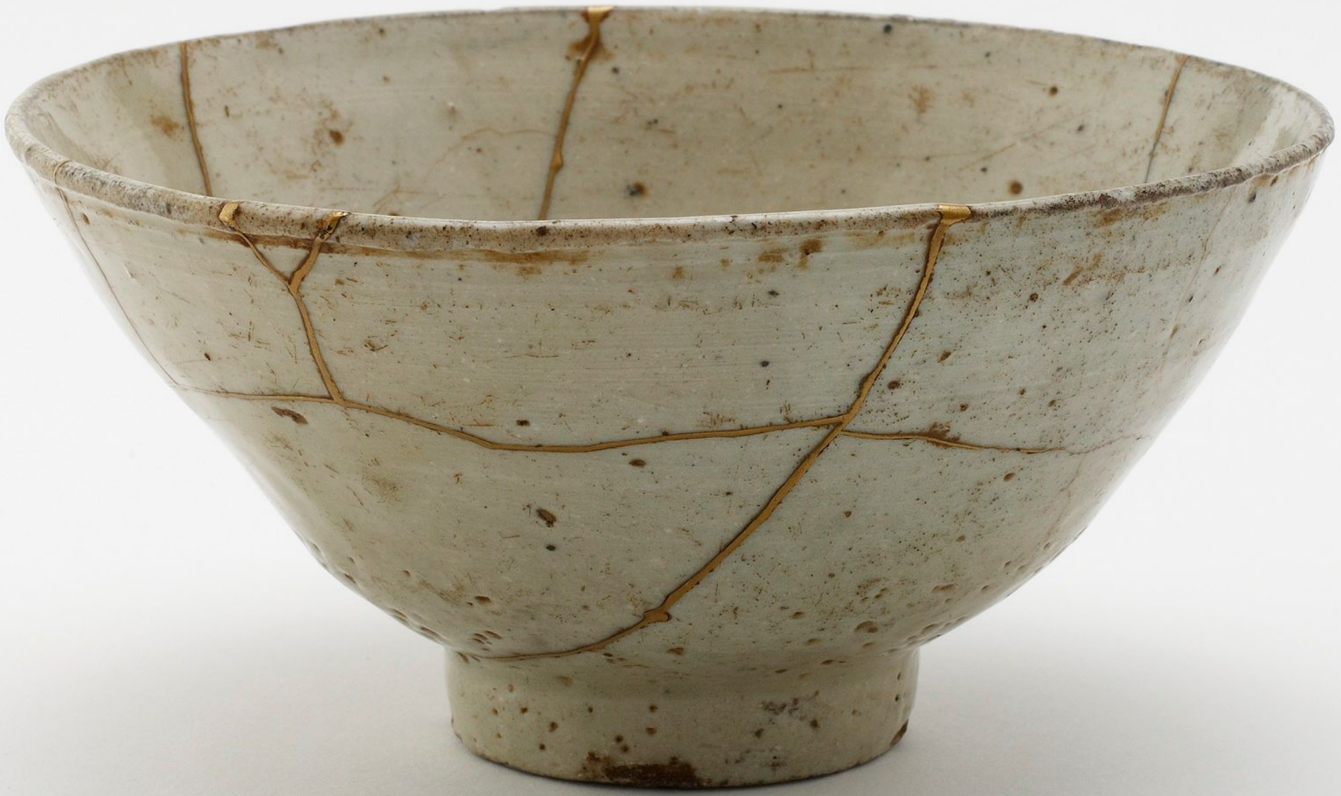
http://www.vam.ac.uk/page/m/modernism/


Eames Chair

The dynamic nature of [Taoist and Zen] philosophy laid more stress upon the process through which perfection was sought than upon perfection itself. True beauty could be discovered only by one who mentally completed the incomplete. … Uniformity of design was considered fatal to the freshness of imagination.

–Kakuzo Okakura

The Book of Tea, 1906



Wabi sabi, kintsugi bowl

N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. *Change Bursts as Defect Predictors*. 2010

"What happens if code changes again and again in some period of time? … Such change bursts have the highest predictive power for defect-prone components [and] significantly improve upon earlier predictors such as complexity metrics, code churn, or organizational structure."

1. Consider the system type
2. Nurture your feedback cycles
3. Accept uncertainty

Vielen Dank
Thank you

Thanks to

SPRINGER NATURE

(who are hiring in London & Berlin)

https://www.springernature.com/gp/group/careers

info@natpryce.com | @natpryce | github.com/npryce | speakerdeck.com/npryce