

Byron Mattingly, PhD, MPH, MBA

<http://www.linkedin.com/in/byronmattingly>

ASQ: CBA, CMQ/OE, CQA, CRE, CQE, CSQE, CSSBB
ANSI-ASQ ISO 17025 Certified Lead Assessor
HL7 Certified Control Specialist
HIMSS CPHIMS, PMP, PMI-ACP

ASQ Software Division: Examining and Awards Chair
Newsletter Chair and Editor

The views expressed in this presentation are my own and do not reflect the views of my employer or other organizations with which I am or have been affiliated.

All cited Trademarks are the property of their respective owners.

Abstract

This session focuses on auditing software vendors that utilize Agile methodologies (e.g., “test-first,” “loose coupling,” etc.) and some of the newer quality tools and technologies in their software development processes:

- test-driven development
- continuous integration
- continuous verification using automated testing

In this session . . .

- ✓ Which areas should be investigated?
- ✓ How do you audit such software, including the “hidden” software tools behind it?
- ✓ What are some of the most important questions to ask?

According to John W. Helgeson:

“[t]he purpose of software quality audits is to monitor software development, the development process, and to help management obtain an independent view of the software development status.”

—*The Software Audit Guide*, ASQ Press 2010, p. xv

An audit program of vendor software developed using agile methodologies thus raises particular challenges because of the reliance of such methodologies on “hidden” software tools that sustain the validation environment.

Software Vendor Audits



Software supplier audits are often a weak spot in a validation and a HUGE hole in purchasing controls and supplier management.

Image source (3/24/2013): http://www.nasa.gov/mission_pages/chandra/news/overweight_hole.html

Auditing Software . . .

- Looks at the whole organization because *System*, *Process* and *Product* audits are all applicable
- Reduces risks
- Nothing to obviously measure or compare against, often created *de novo*
- Examines how software is created
- Different types of artifacts depending on the SDLC employed

Apples and Androids



← OR →

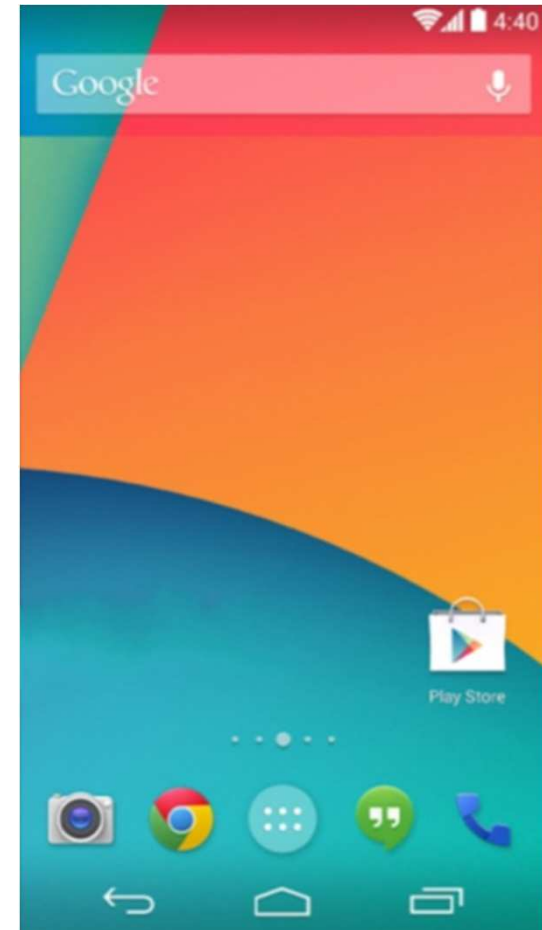


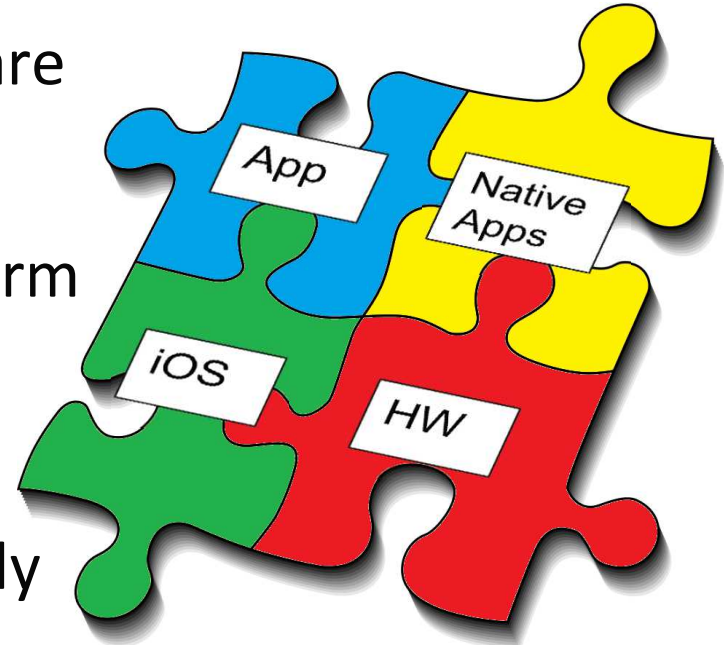
Image sources:

[http://upload.wikimedia.org/wikipedia/commons/e/e0/Steve Jobs with the Apple iPad no logo %28cropped%29.jpg](http://upload.wikimedia.org/wikipedia/commons/e/e0/Steve_Jobs_with_the_Apple_iPad_no_logo_%28cropped%29.jpg)

[http://en.wikipedia.org/wiki/Android_\(operating_system\)](http://en.wikipedia.org/wiki/Android_(operating_system))

Risks, paradigms and the Apple iPad

- ✓ The hardware, OS and native apps are inseparable
- ✓ The iPad is a high risk mobile platform
 - consumer grade device
 - few management controls
- ✓ Software is complex, but superficially easy to change



Jigsaw diagram based on: http://en.wikipedia.org/wiki/Jigsaw_puzzle

By the way . . .

By end of 2012, 50% of all doctors were using tablets

Source:

<http://www.computerworlduk.com/news/public-sector/3319356/tablets-increasingly-used-by-us-doctors-to-treat-patients/>

Why a traditional validation model does not work



Image Source: http://en.wikipedia.org/wiki/Ford_Model_T

Agile Manifesto:

<http://www.agilemanifesto.org>

“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.”

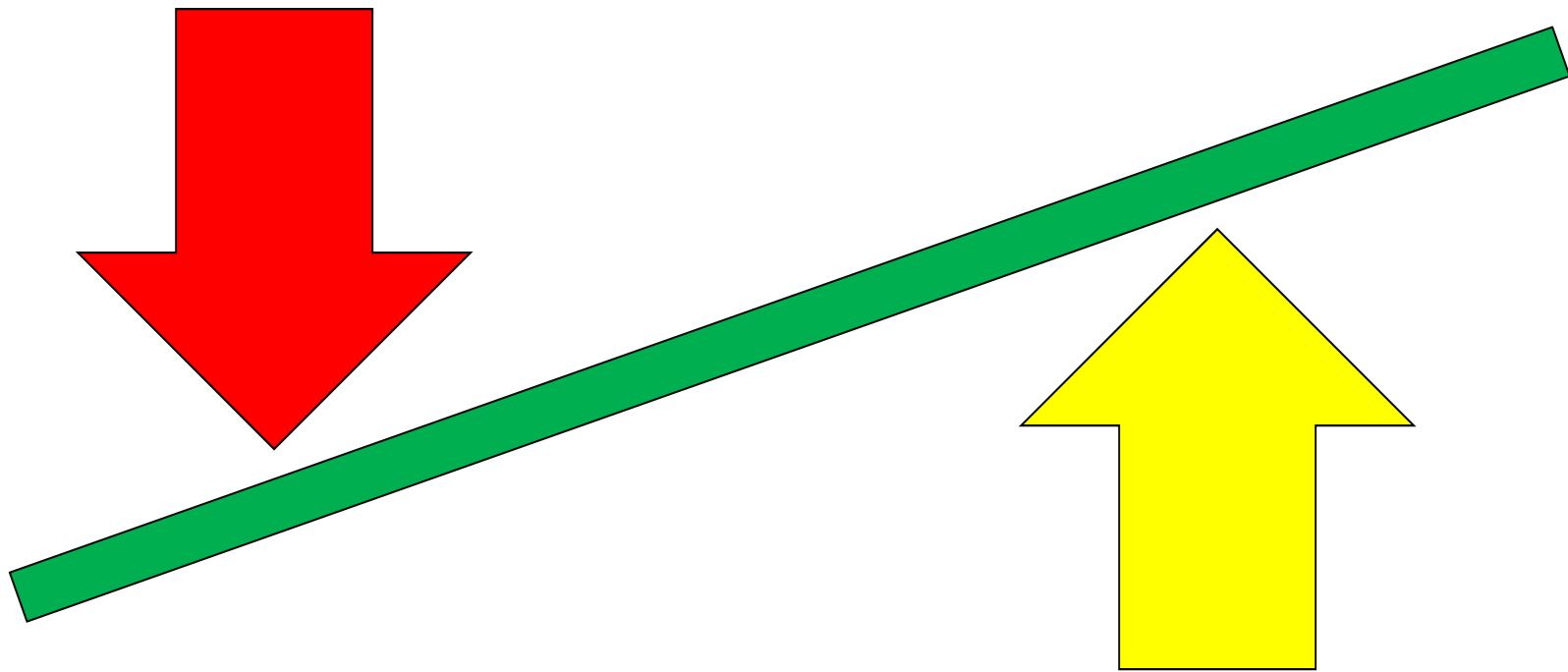
The Agile Manifesto may appear to be contrary to the values of a quality management system

Image source (4/20/2013): http://en.wikipedia.org/wiki/Ultralight_aviation



What are the tradeoffs?

Individuals and
interactions



Processes and tools

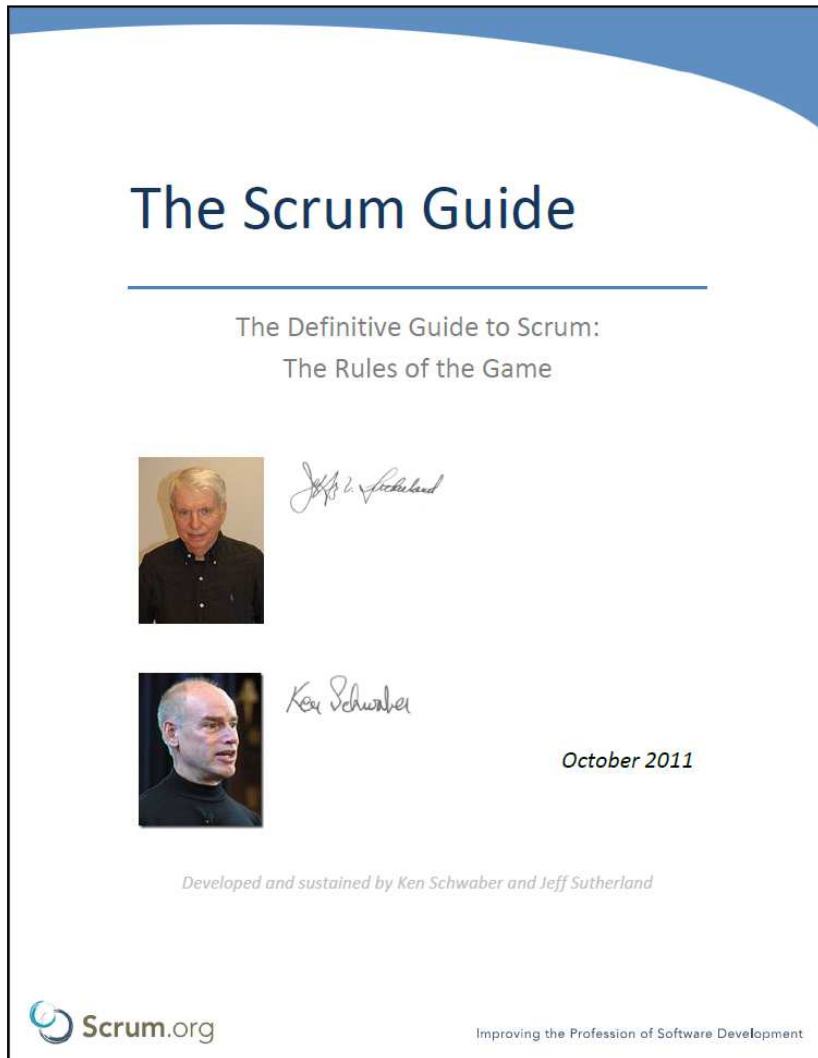
Agile Characteristics

- Cross-functional teams work in short iterations and take an incremental approach
- Focus on business priorities and customer value
- Emphasis on continuous improvement

Agile Methodologies and Frameworks

- **XP** (pair programming, simple designs, refactoring)
- **Lean**
(defer decisions, deliver fast, eliminate waste, build quality in, optimize the whole)
- **Crystal Methodologies**
(clear: 3-6, yellow: 6-20, orange: 20-40, red: 40-80)
- **Feature Driven Development**
(subject areas, feature sets, features)
- **Dynamic Systems Development Method**
(fitness of product for its intended business purpose)
- **Agile Unified Process**
(phases: inception, elaboration, construction, transition)
- **and Scrum . . .**

The Definitive Scrum Guide



Scrum Overview

Scrum (n): A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value. Scrum is:

- Lightweight
- Simple to understand
- Extremely difficult to master

Scrum is a process framework that has been used to manage complex product development since the early 1990s. Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.

—*Scrum Guide*, p. 3

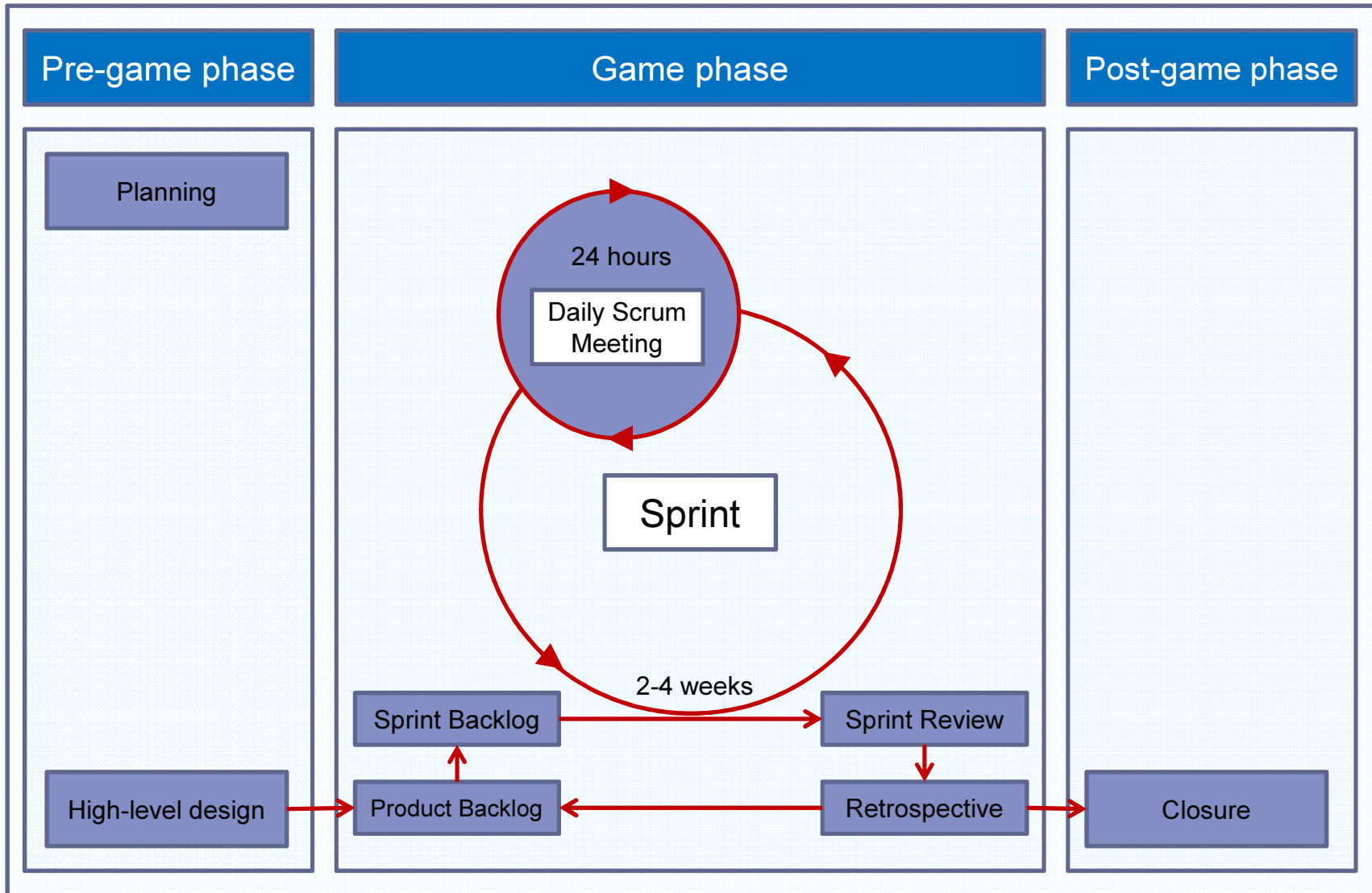
Source: <http://www.scrum.org/Scrum-Guides>

Scrum in 100 words

- Scrum is an agile ~~process~~ methodology that allows us to focus on delivering the highest business value in the shortest time.
- It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month).
- The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features.
- Every two weeks to a month anyone can see real working software and decide to release it as is or continue to enhance it for another sprint.

Source (4/20/2013): www.mountaingoatsoftware.com/scrum

Scrum is an Agile Framework



Meir “Manny” Lehman’s Law

“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, 1980

“The computing scientist’s main challenge is not to get confused by the complexities of his own making.”

—E. W. Dijkstra

Source: http://en.wikipedia.org/wiki/Technical_debt

What is Technical Debt?

Decisions made to defer necessary work may result in *technical debt*

- “Technical Debt includes those *internal* things that you choose not to do now, but which will impede future development if left undone. This includes deferred refactoring.
- Technical Debt doesn’t include deferred functionality, except possibly in edge cases where delivered functionality is ‘good enough’ for the customer, but doesn’t satisfy some standard.”

—Ward Cunningham

Source: <http://c2.com/cgi/wiki?TechnicalDebt>

What is re-factoring?

Code refactoring is a “disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior,” undertaken in order to improve some of the nonfunctional attributes of the software.

Source: http://en.wikipedia.org/wiki/Code_refactoring

What is Regulatory Debt? /1

Decisions made to defer necessary risk management and control throughout a software development lifecycle may result in *regulatory debt*

*What is Regulatory Debt?*_{/2}

How to Pay *Down* the Debt:

- Technical Debt → Refactor
- Regulatory Debt → Risk Control
(esp. Refactor into “System of Systems”)

“Controlling complexity is the essence of computer programming.”
—*Brian Kernighan*

Sources of Regulatory Debt_{/1}

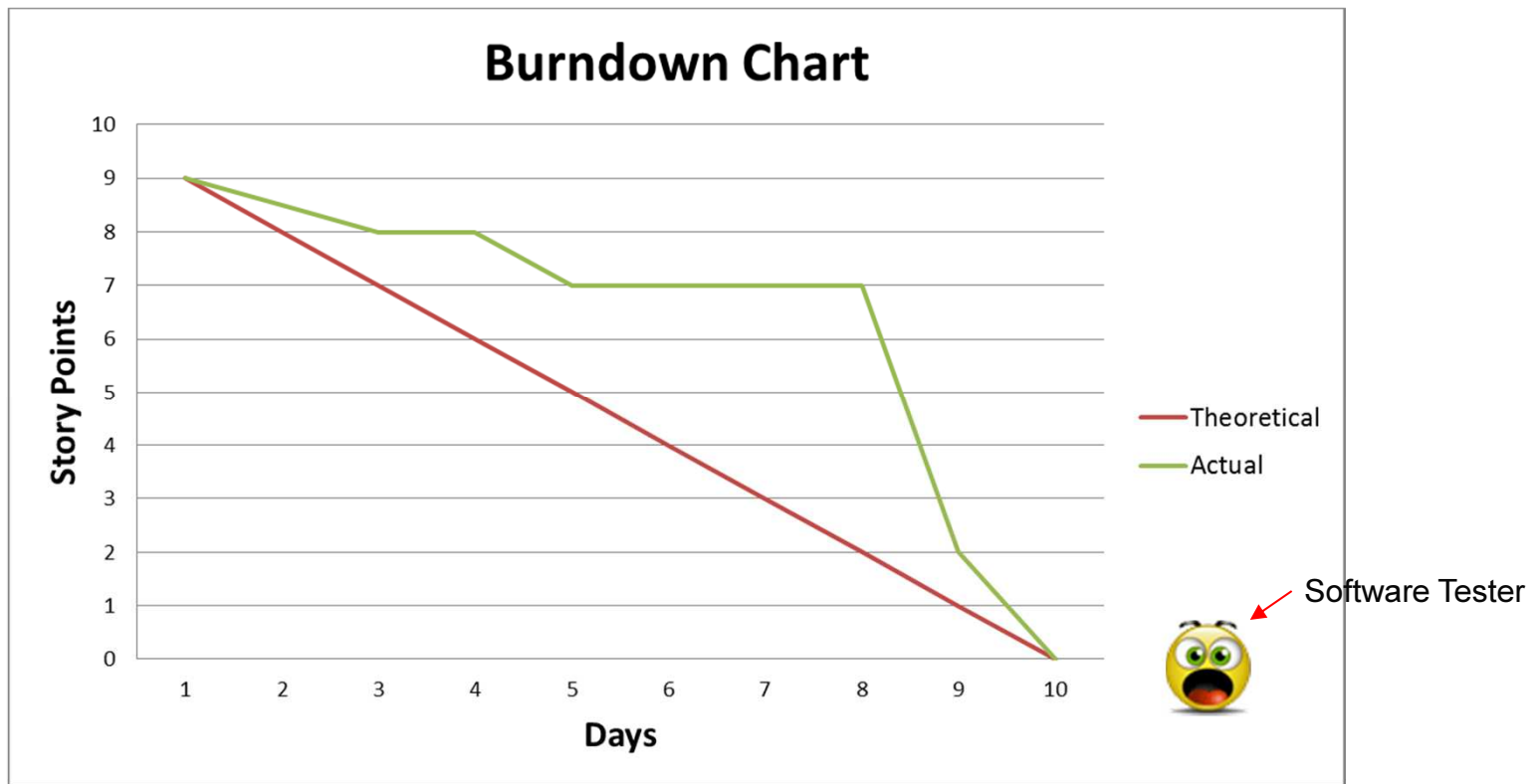
1. DoD: Done vs Done-Done
2. Just-in-time design
3. Lack of spikes for understanding what's being developed
4. Overly large / complicated user stories
5. Bad estimations

Sources of Regulatory Debt_{/2}

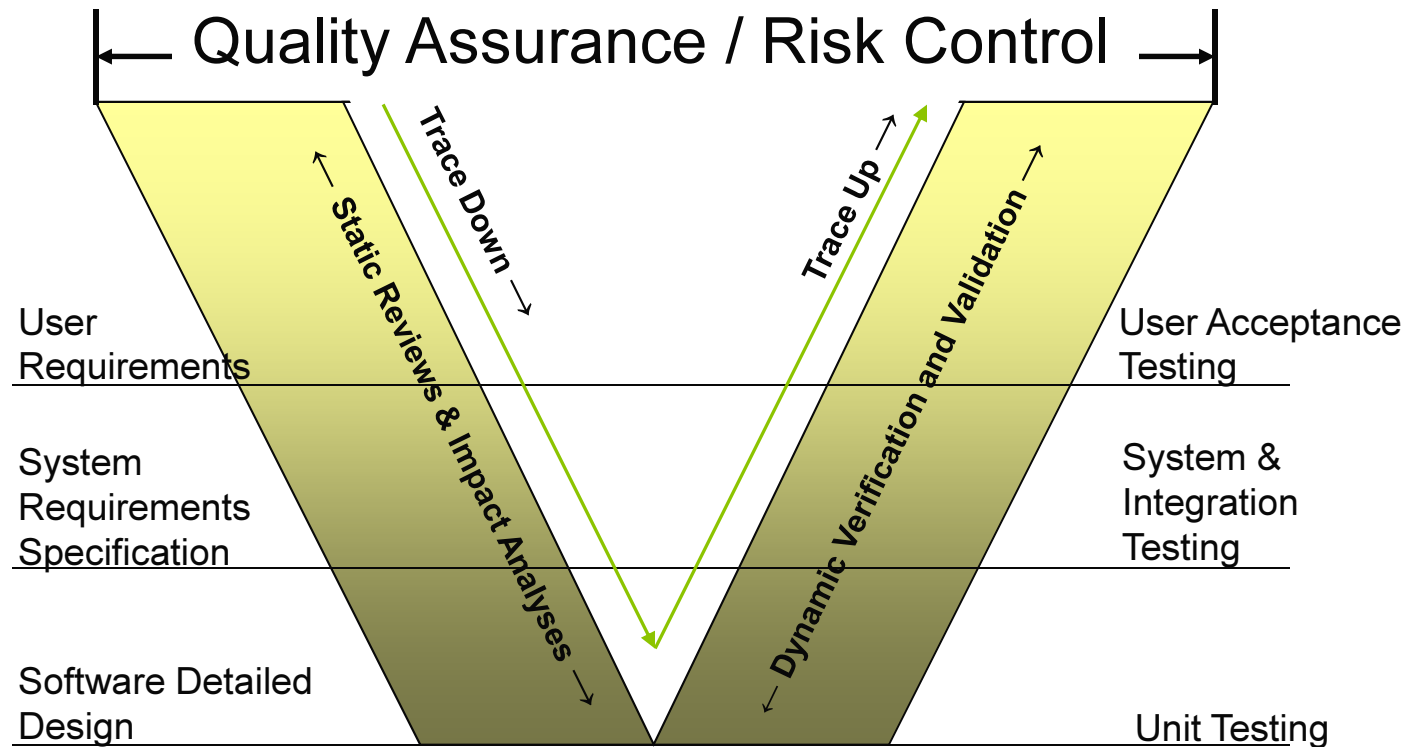
6. Pressure to get it done
7. Rushed sprints that compromise good software development
8. Lack of multiple perspectives on risk
9. Weak ScrumMaster / overbearing PO
10. Nature of complex systems!
11. “Scrum—*but*”

Sources of Regulatory Debt_{/3}

12. Agile “Falls” (hand-offs, “test” sprints, etc.)

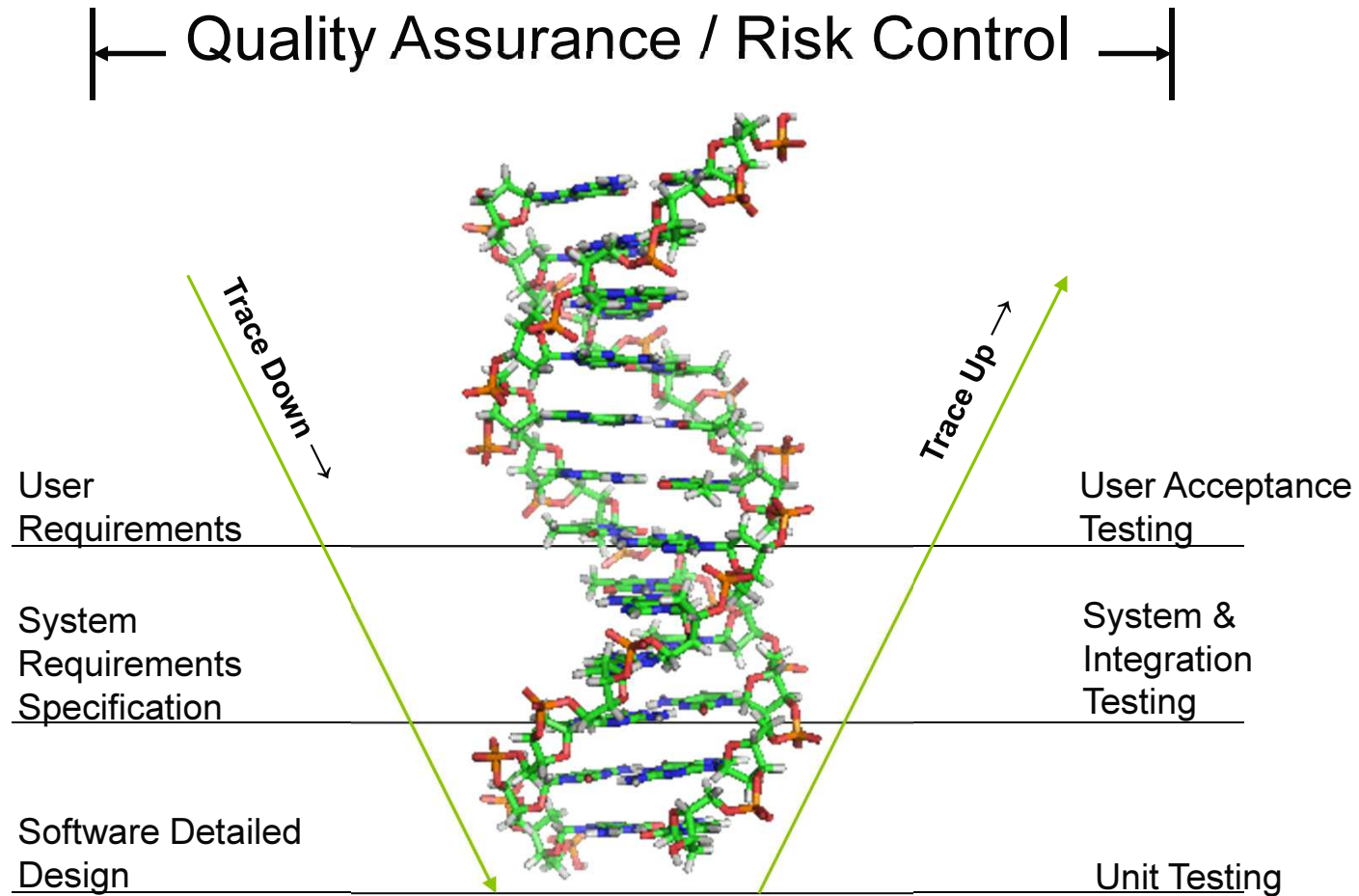


V-model: “Bent” Waterfall or Agile?



- Relationship between artifacts are time-independent
- Use “hardening sprints” for regulatory compliance (and to pay down accumulated regulatory / architectural debt)
- Finish to finish parallelism (vs. start to finish)
- Design Inputs ↔ Design Outputs
- Synchronize Approvals

V(ortex)-Model_{/1}



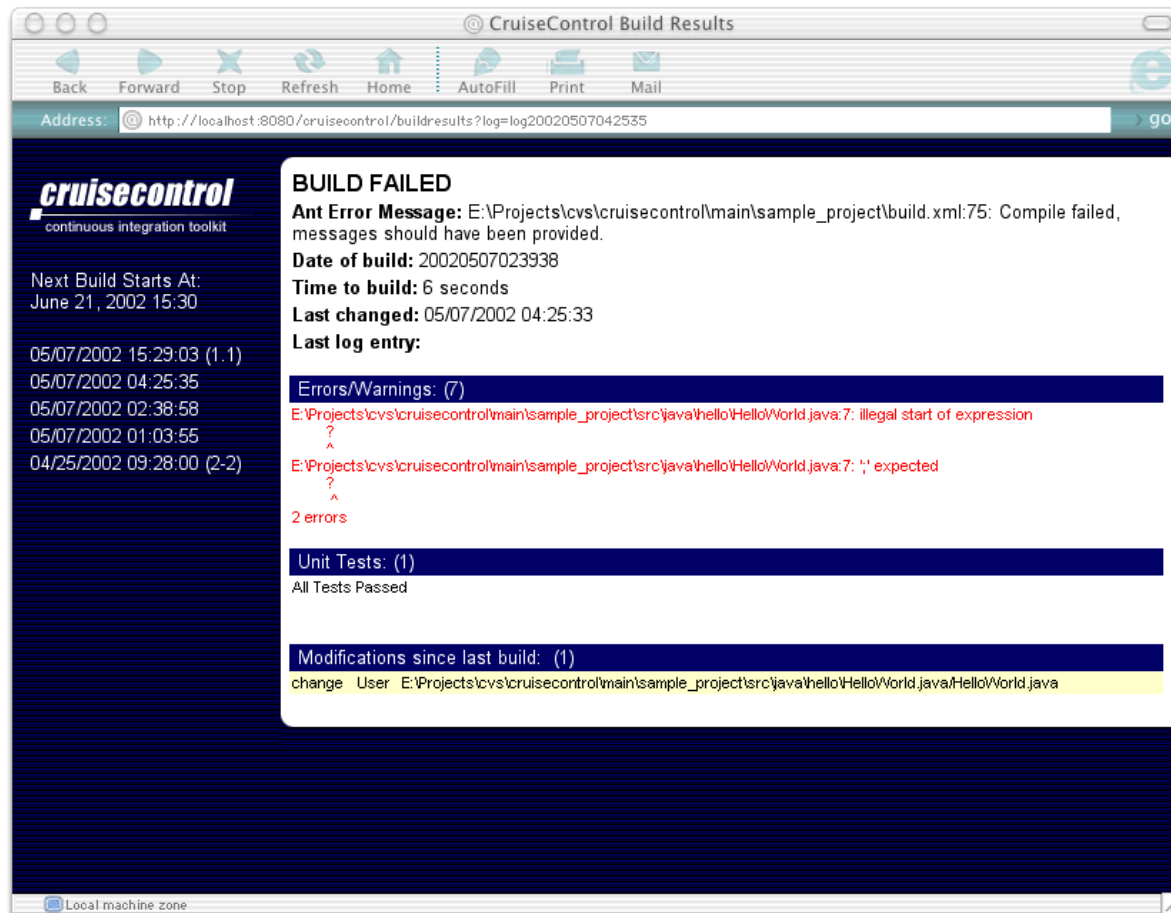
*V(ortex)-Model*_{/2}

- Relationship between artifacts are time-independent
- Use “hardening sprints” for regulatory compliance (and to pay down accumulated regulatory / architectural debt)
- Finish to finish parallelism (vs. start to finish)
- Design Inputs ↔ Design Outputs
- Synchronize Approvals

Test early and often . . .

- “Forces” projects to develop testable requirements early
- Examples of dramatic compressions of software development lifecycle timelines when used in conjunction with continuous integration
- Can’t prove “correctness” but can improve robustness

Example JUnit Continuous Integration Testing System: Cruise Control



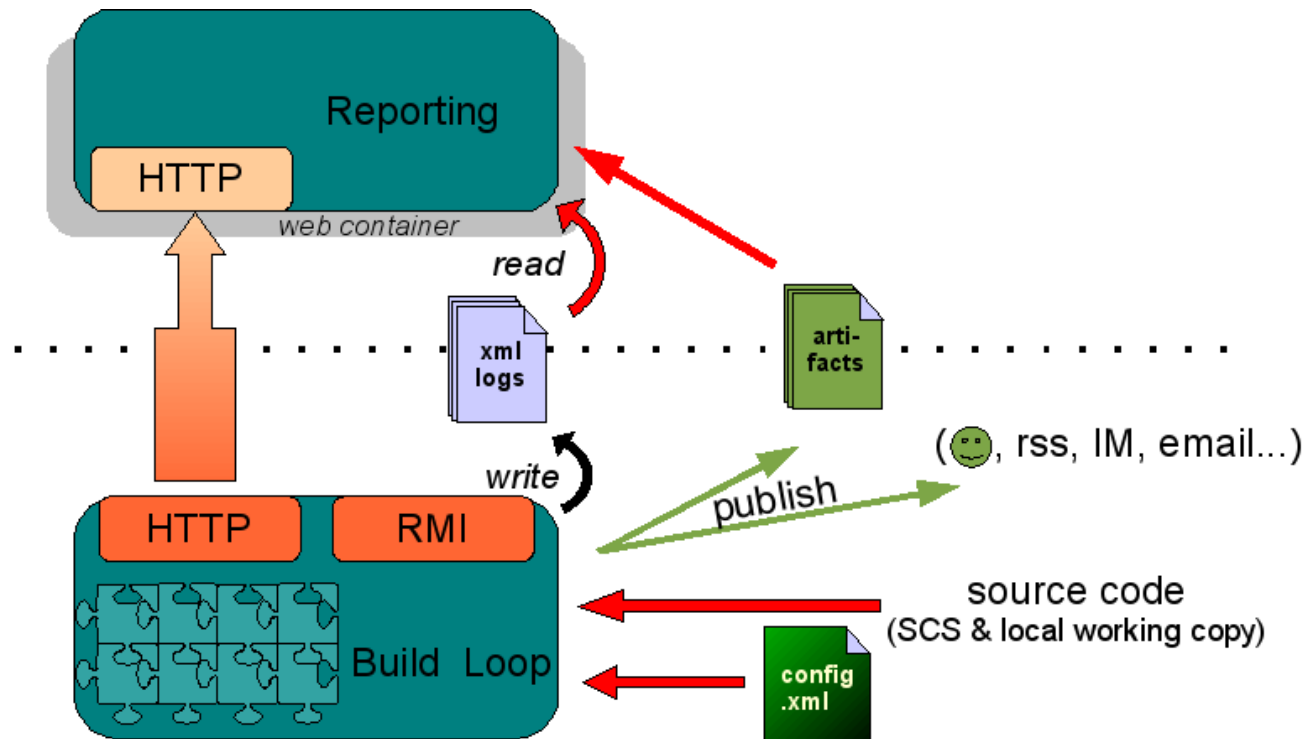
The screenshot shows a web browser window titled "CruiseControl Build Results". The address bar contains the URL: `http://localhost:8080/cruisecontrol/buildresults?log=log20020507042535`. The page content is as follows:

- cruisecontrol** continuous integration toolkit
- Next Build Starts At: June 21, 2002 15:30
- Build history:
 - 05/07/2002 15:29:03 (1.1)
 - 05/07/2002 04:25:35
 - 05/07/2002 02:38:58
 - 05/07/2002 01:03:55
 - 04/25/2002 09:28:00 (2-2)
- BUILD FAILED**
- Ant Error Message:** E:\Projects\cvcs\cruisecontrol\main\sample_project\build.xml:75: Compile failed, messages should have been provided.
- Date of build:** 20020507023938
- Time to build:** 6 seconds
- Last changed:** 05/07/2002 04:25:33
- Last log entry:**
- Errors/Warnings: (7)**
- Errors:
 - E:\Projects\cvcs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java:7: illegal start of expression
 - E:\Projects\cvcs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java:7: ';' expected
- Unit Tests: (1)**
- All Tests Passed
- Modifications since last build: (1)**
- change User: E:\Projects\cvcs\cruisecontrol\main\sample_project\src\java\hello\HelloWorld.java\HelloWorld.java

Source (3/24/2013): <http://cruisecontrol.sourceforge.net/reporting/jsp/index.html>

See also *Hudson, Jenkins, et al.*

Example Continuous Integration Testing System: Architecture

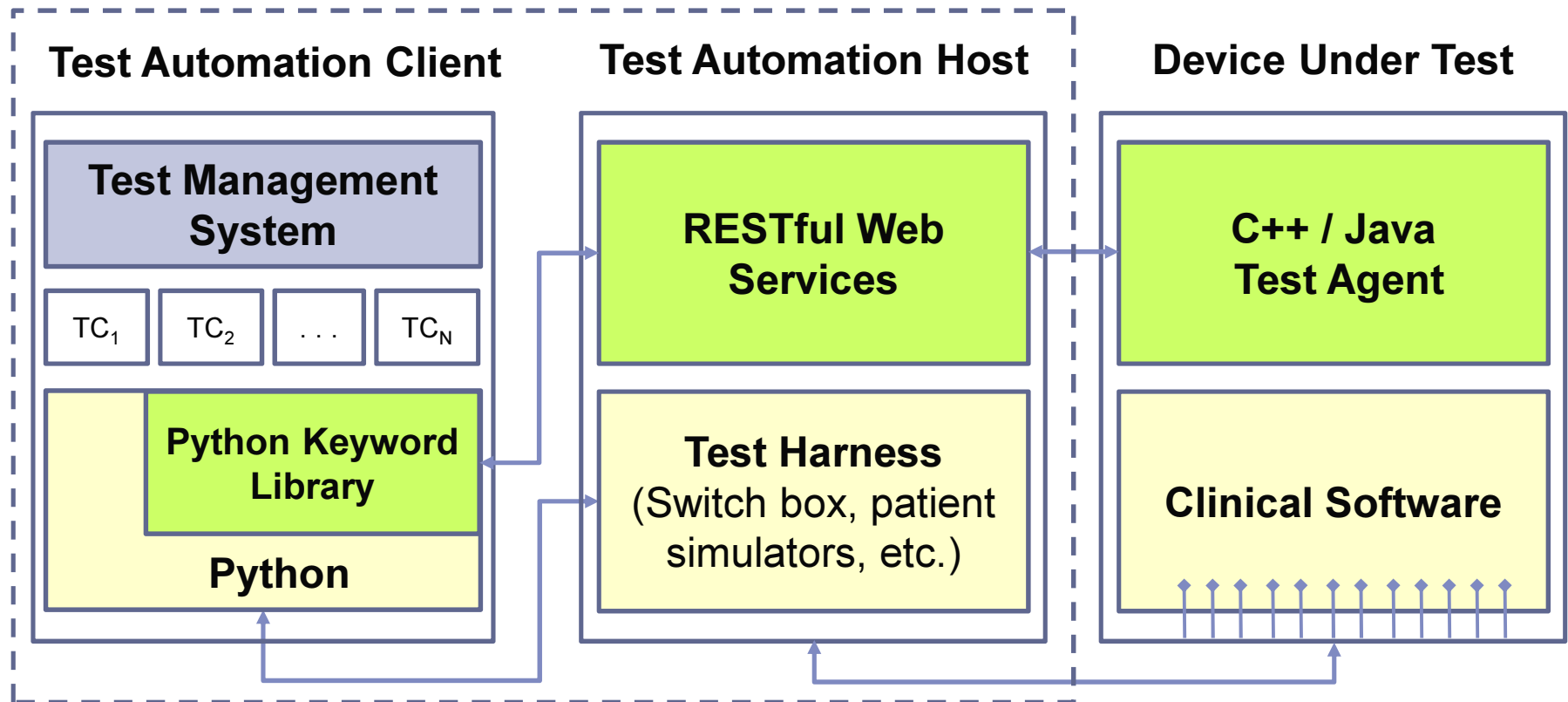


Source (3/24/2013): <http://cruisecontrol.sourceforge.net/overview.html>

Example Automated Verification Testing System High Level Design

TC = Test Case
Goal: 90-95% TCs Automated

Test Controller: PC or Linux SBC



What is Continuous Agile Testing?

- Establishes test environment and configures testing tools; tests as code becomes (relatively?!) stable
- Combines manual and automated testing approaches
- Is performed at unit, integration *and* acceptance levels. (Can be extended post-implementation by design.)

Continuous Agile Testing: Challenges

- QA teams are often distributed and not co-located.
- Agile development teams do not consider testing their responsibility; “Quality” is the responsibility of QA.
- Sprint Backlog and Planning often fail to include test planning, execution and reporting. Testing resources typically added at later stages of development.
- Test automation can take time to implement and maintain.
- Verification activities for regulated industries can be rigidly defined by the manufacturer’s QMS and burdensome to document.

Continuous Agile Testing: Best Practices / Strategies

- Automate throughout the SDLC. Testing is not a phase; *everybody* tests!
- Embrace concurrent testing—manual, semi-automated and automated. Promote *ad hoc* and independent testing that explores corner cases, patient flows and scenarios, and foreseeable misuse.
- Test loosely coupled systems partitioned by risk; allocate testing assets and perform functional validations of “System of Systems” according to risk level.

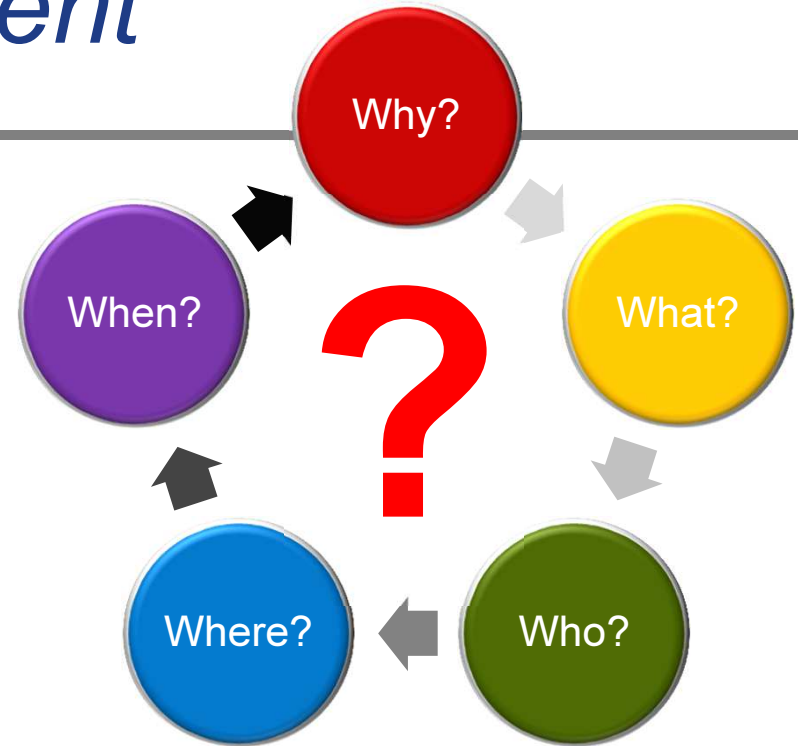
Intended Use Statement

System Defining Questions

- **Why** does this system exist?
- **What** is it supposed to do?
- **Who** is supposed to use it?
- **Where** will it to be used?
- **When** will it to be used?

Boundaries of the system

- What parts of the overall process does this system **automate**?
- What are the interfaces between this system and other **processes**?
- What are the interfaces between this system and other **systems**?



Example “Detailed” User / System Requirements

2.c When you tap the “Animations” category, 3 animations can be viewed and emailed.

For the following 3 animations, each:

- Plays when tapped but without sound
- Can be paused and re-started
- Can be maximized to full screen when the <maximize arrows> are tapped with a finger; can be minimized to normal size when the <minimize arrows> are tapped
- Has an email icon enabled that when tapped will open an Outlook message with the animation already attached.
 1. Mickey animation
 2. Pluto animation
 3. Donald animation

Risk Management (Control)

The estimation of risk for a given hazard is a function of the relative likelihood of its occurrence and the severity of harm resulting from its consequences.

Risk = Severity x Probability of hazard

Following the estimations of risk, risk management focuses on controlling or mitigating the risks.

Medical Device Use-Safety: Incorporating Human Factors Engineering into Risk Management (July 18, 2000), p. 8

Example Risk Rating

	Severity		
Frequency	Business	Compliance	Hazard to People
Regularly occurs: > 1 in 100 to 1 in 10	Major	Major	Major
May occur, with unknown regularity: > 1 in 1,000 to 1 in 100	Moderate	Moderate	Major
Highly unlikely: > 1 in 1,000,000 to 1 in 1,000	Minor	Minor	Moderate

Severity

- **Business:** A business problem or inconvenience results unrelated to compliance or a personal injury
- **Compliance:** Compliance is compromised as a result (e.g. exposure to a regulatory finding)
- **Hazard to People:** A person is injured or harmed in any way

Risk Level

- **Major:** Likely to occur with serious consequences
- **Moderate:** May occur with potentially serious consequences
- **Minor:** Unlikely to occur

Example Risk Register / Mitigation Plan

ID	Risk	Impact	Initial Risk Level	Mitigation	Residual Risk Level
1			Major, Moderate or Minor		Major, Moderate or Minor
2			Major, Moderate or Minor		Major, Moderate or Minor
...			Major, Moderate or Minor		Major, Moderate or Minor
N			Major, Moderate or Minor		Major, Moderate or Minor

Preliminary Hazard Analysis (PHA)

A Preliminary Hazard Analysis (PHA) identifies hazards, consequences, level of risk (risk index), and potential mitigations (usually implemented in the design) in the early stages of a project's life cycle

Preliminary Hazard Analysis Table

Hazard ID	Hazard	Consequence (what, where, when)	Probable Cause(s)	Risk Category	Risk Index (1,2,3)	Risk Mitigation	Verification	Residual Risk

But: Should keep re-visiting PHA as requirements, design, and conditions evolve!

FTA: Intended Use vs. Undesired Effects

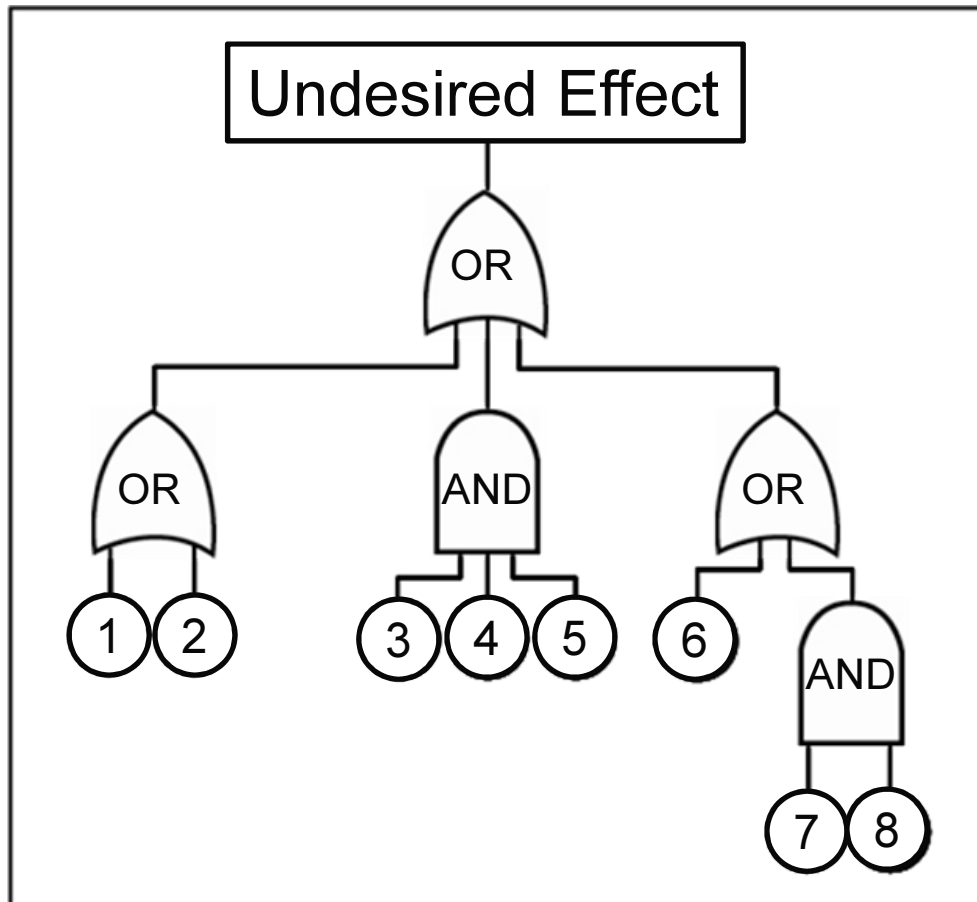


Diagram based on: https://en.wikipedia.org/wiki/Fault_tree_analysis

See also: <http://www.hq.nasa.gov/office/codeq/risk/docs/ftacourse.pdf>
<http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf>

- Analyze effects to deductively find how they are caused by combinations of other failures.
- Requires thorough understanding of system and its intended use(s)
- Shows robustness—i.e. resistance of complex systems to external events.
- *Not* good at examining multiple system-wide failures but good at examining multiple interrelated *causes* of failure.

Risk-Based Validation Model

Software Type %Projects	Vendor Assessment	User Reqts	UAT	System Reqts	System / Integration Testing	Software Design	Unit Testing
1. OTSS ~10%	Dark Blue	Dark Blue	Dark Blue	White	White	White	White
2. Configured ~55%	Light Blue	Dark Blue	Dark Blue	Dark Blue	Dark Blue	White	White
3. Customized / Developed ~35%	Light Blue	Dark Blue	Dark Blue	Dark Blue	Dark Blue	Dark Blue	Dark Blue

OTSS = "Off The Shelf Software"

FDA: Risk-based approach to allocate validation efforts

Antikythera mechanism: *Intended Use?*



Prime Numbers: 19, 53, 127, 223

Image source: http://en.wikipedia.org/wiki/Antikythera_mechanism

A Tightly Coupled System: 0.112579655

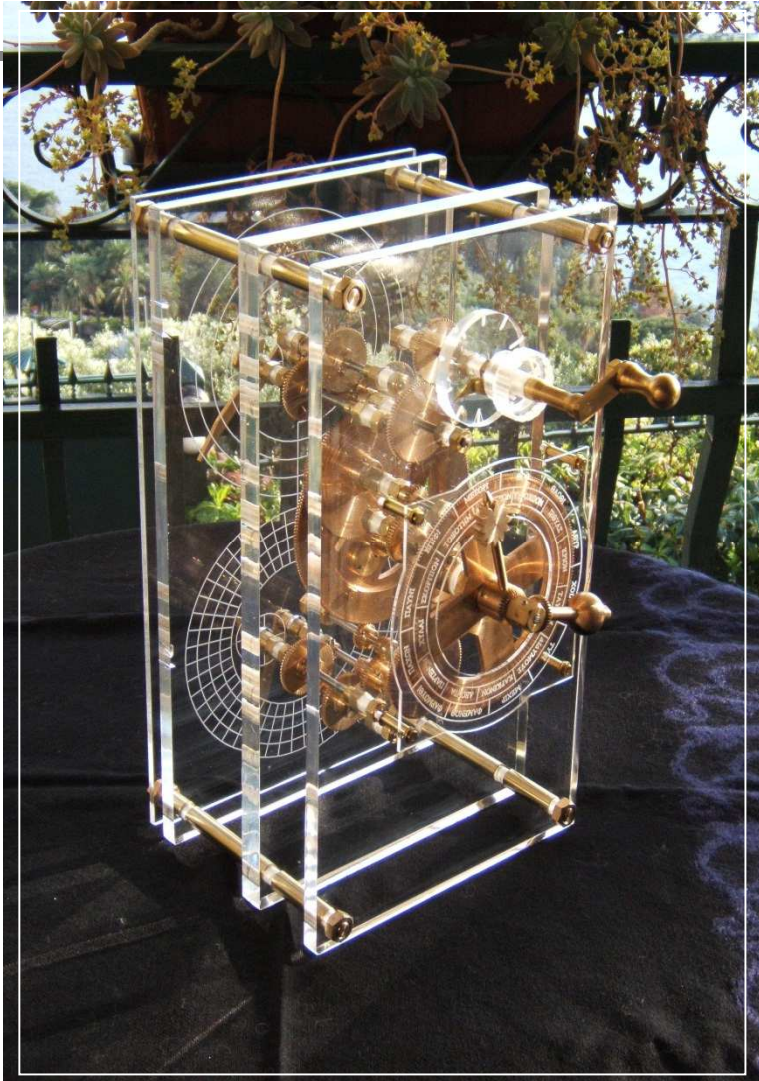
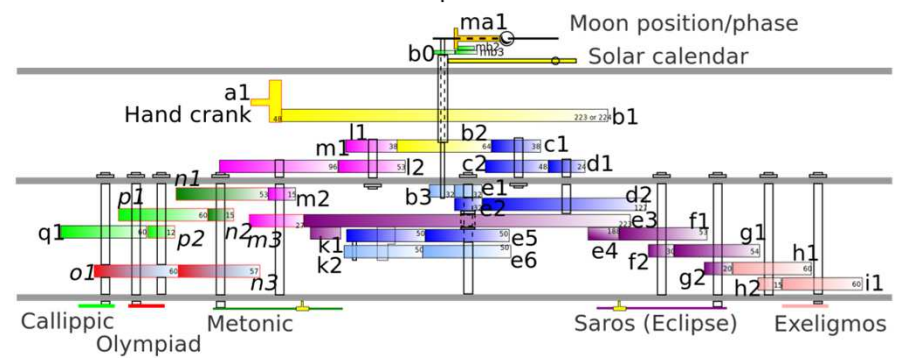
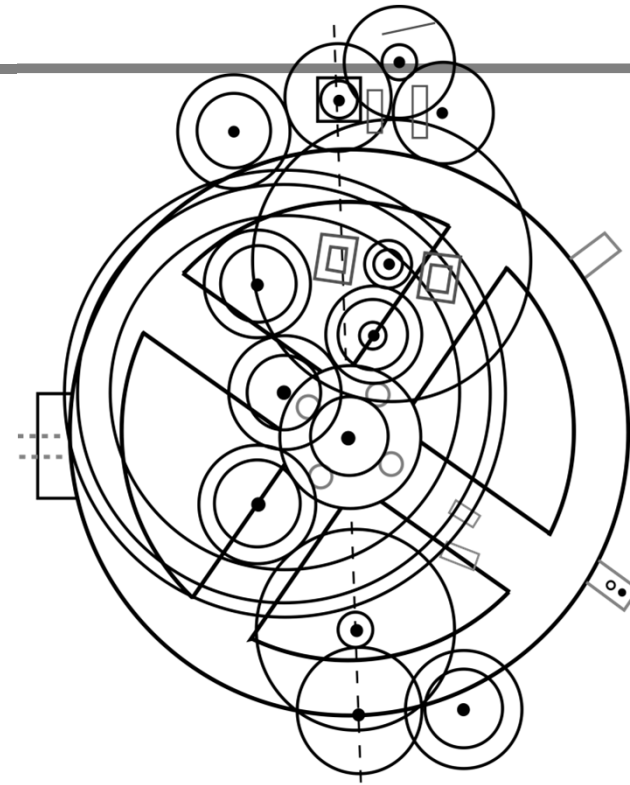
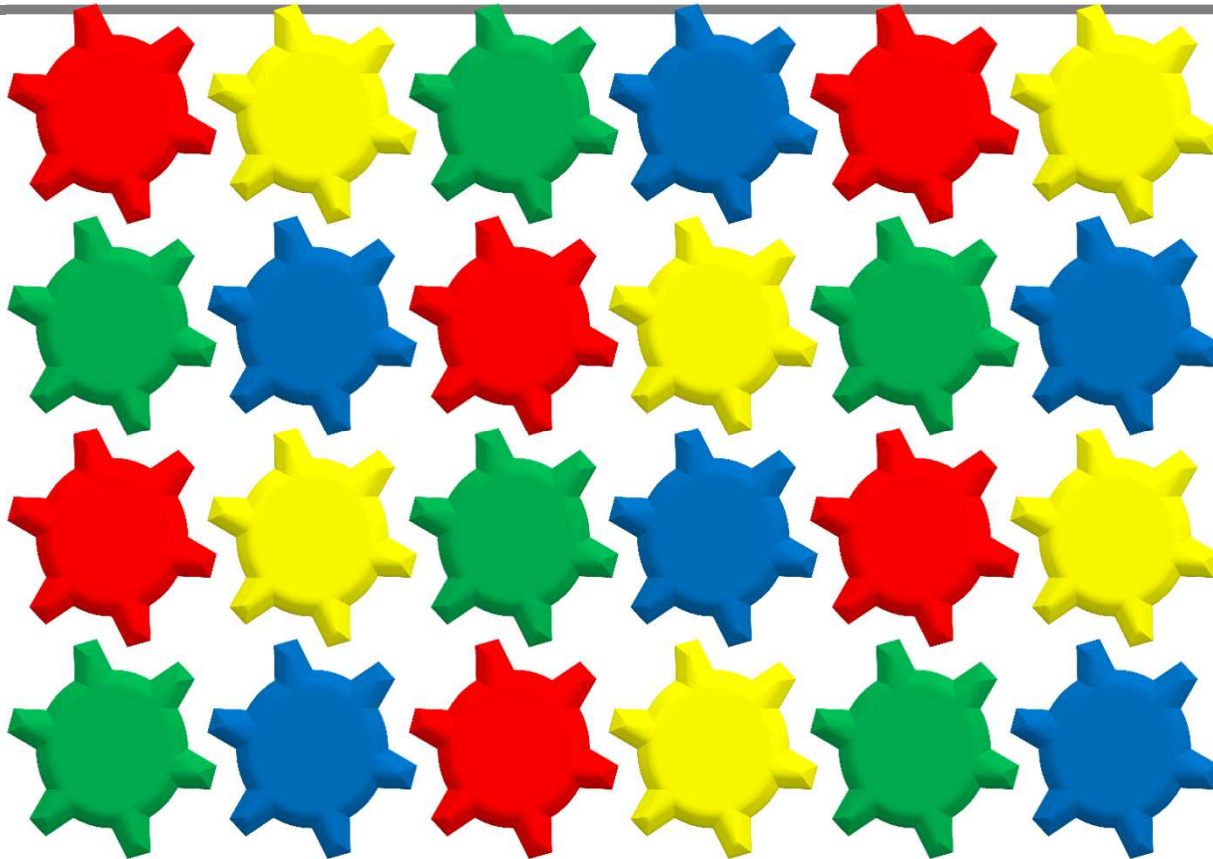


Image source: http://en.wikipedia.org/wiki/Antikythera_mechanism



Partition by Risk into “System of Systems”



Risk-based Agile Deployment means partitioning an overall system by identifying intended use(s) and by using risk as a scaling factor into a “System of Systems” that are loosely coupled (coherent *and* cohesive) and that can be validated according to risk level.

Decompose “System of Systems” into separate functional validations

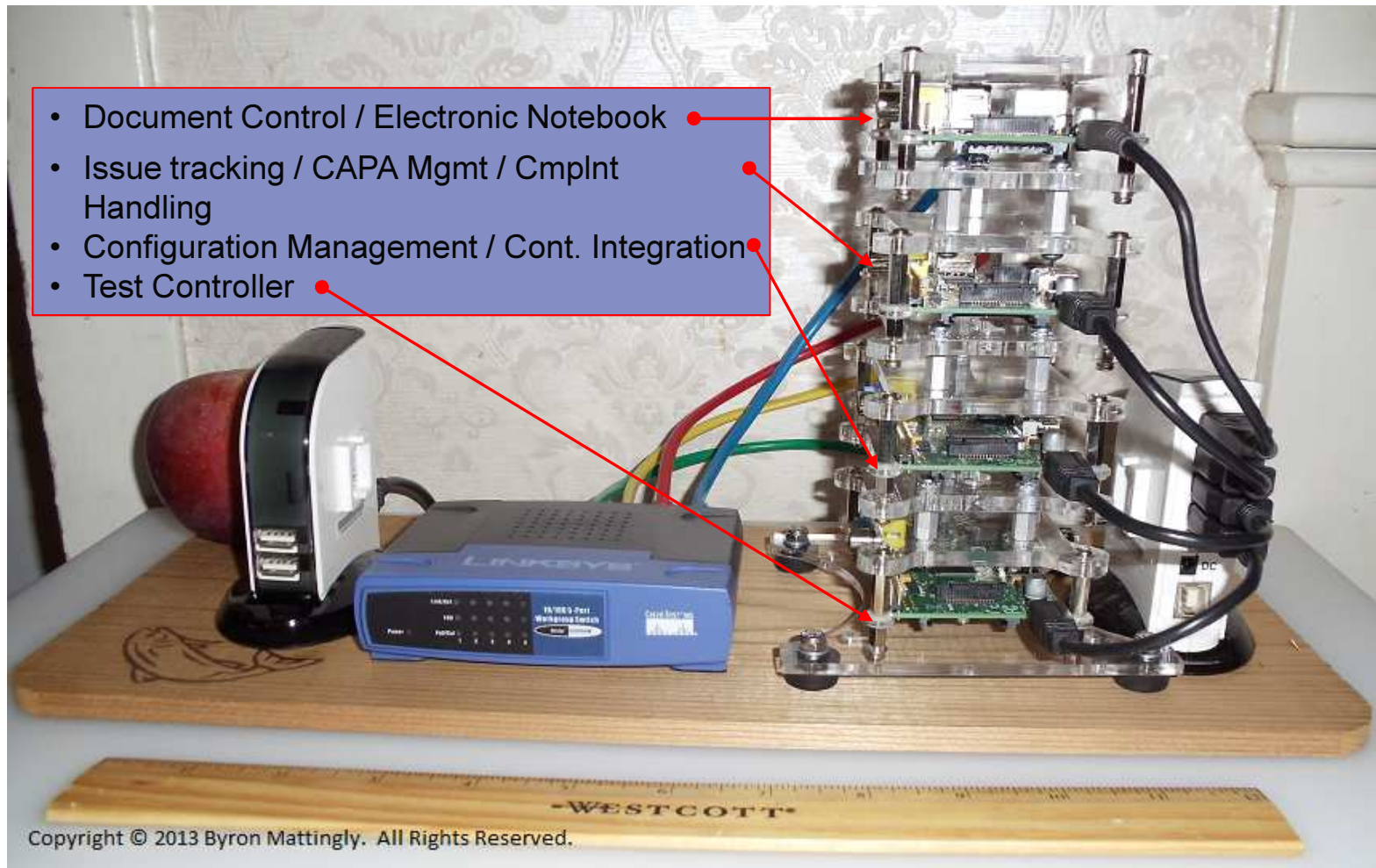


This simplifies the validation effort and reduces the risk of “vertical cascading” that can push an overall system to a tipping point and into catastrophic failure.

Key Questions

- How does automation support development?
- What about other types of manual testing, e.g. ad hoc, corner case, foreseeable misuse?
- How often and when does “paying down” technical debt / compliance debt by refactoring get recorded in document control?

Thank you!



ByronMattingly2013@yahoo.com