THIRTY-SECOND ANNUAL
PACIFIC NORTHWEST

# SOFTWARE QUALITY CONFERENCE

# PNSQC ™

October 20-22, 2014

World Trade Center Portland
Portland, Oregon

# PNSQC™ 2014 President's Welcome

This year's theme "Bridges to Quality" came about as we discussed all the processes and people that are involved in our development activities.  The reality is that we all own quality and we should all feel empowered to do what is needed to meet and exceed our customers' expectations.  Recently I was sitting down with a software quality leader who was rightfully frustrated that a development team was ignoring all the basic practices of software engineering. His team was relying on the software quality organization to test the product and determine its quality.  This person's experience with their group is very backwards and fitting as we look forward to this year's conference and see how we can bridge the differences we have with our teams and processes.  You can't test quality in – it must be built in from the very foundation.

This year we are pleased to have Dr. Richard Turner kick-off the technical conference.  Dr. Turner is an author of four books and enjoys research in many areas including applying software processes.  He will be providing some insight into the concepts of discipline, agility, and their relationship within the context of system engineering.  So come and join us as he goes on to address whether agile systems engineering is a meaningful concept and his suggestions for evolving a more useful paradigm for contemporary systems engineering.

On Tuesday we welcome Jon Bach as he returns to PNSQC as a keynote speaker kicking us off with a presentation on "Live Site Quality: the Bridge Between Your Silos".  Jon is well known in the testing community and will be sharing his two and half years of experience as a Quality Engineering Director at EBay where he will introduce to us a concept he calls "Live Site Quality".  His main idea is "selling" stakeholders on different perspectives and getting colleagues from different parts of the organization to buy the idea that the defects that most affect their Live Site Quality may lie between their team and another.

If you have been to our conference in the past, you know that we use the format of multiple tracks, to give you plenty of choices throughout the two-day conference.  We have also continued to maintain the 45-minute presentation format with 10 minutes between sessions to ensure that presenters and attendees have plenty of time to get settled and ready for the next topic.

Monday Night will include a reception with poster papers on the mezzanine level along with hors d'oeuvres and beverages.  We have also partnered with the Technology Association of Oregon (TAO) who will be leading an "Open Mic Night" with the theme *Blood-Curdling Tales of Testing Terror.*   On Tuesday night we will have the pleasure of collaborating with Rose City Software Process Improvement Network (SPIN) where Jon Bach will be discussing "Inciting and Inviting a Quality Culture".

As we have done in the past during the Monday and Tuesday lunch hours, we will be doing deep dive birds of the feather sessions.  This is an excellent opportunity to discuss quality topics with your peers while being moderated by an industry activist.

Our conference is full of practical, useful, and valuable information.  With everyone's help, software quality continues to move forward.  We all play an active part in building the bridge to quality!  I am glad you are here and hope to associate and network with each of you as we strive to accomplish our mission, which is to "Enable knowledge exchange to produce higher quality software".

Doug Reynolds, President, PNSQC 2014

# TABLE OF CONTENTS

## Keynote Addresses

## Invited Speakers

## Agile Development Track

## Automation and Tools Track

## Lifecycles, Processes and Methodologies Track

## Measurement and Metrics Track

## Mixed Track

## Mobile Track

## People and Management Track

## Performance Engineering Track

## Security Track

## Testing and Quality Assurance Track

# PNSQC 2014 – Conference At A Glance

| Monday Oct 20, 2014 • Day One • Technical Program | | | | |
|---|---|---|---|---|
| 7:00-8:00 AM | **Registration and Exhibits Open** Level 2 Mezzanine | | | |
| 8:00-8:15 AM | **Welcome and Opening Remarks** Level 3 Auditorium | | | |
| 8:15-9:45 AM | **KEYNOTE: Balancing Agility and Discipline: Bridging the Gaps Between Software and Systems Engineering**<br>Dr. Richard Turner, School of Systems and Enterprises, Stevens Institute of Technology, Level 3 Auditorium | | | |
| 9:45-10:15 AM | **Break in Exhibit Hall** Level 2 Mezzanine | | | |
| | **Level 3 Auditorium** | **Level 1 Plaza Room** | **Level 2 Mezzanine Room** | **Level 3 Skybridge Room** |
| | **INVITED SPEAKER** | **AGILE DEVELOPMENT** | **PEOPLE AND MANAGEMENT** | **AUTOMATION AND TOOLS** |
| 10:15-11:00 AM | **You're Doing It Wrong: How Your Decision-making Actually Increases Uncertainty and What To Do About It**<br>Hillel Glazer, Entinex, Inc. | **Does DAD Know Best, Is it Better to do LeSS or Just be SAFe? Adapting Scaling Agile Practices into the Enterprise**<br>Aashish Vaidya, Cambia Health Solutions | **To Build an Agile Company, Do Not Begin with Agile Development**<br>Michael Belding, Phyllis Thompson, ShiftWise | **Improving R&D Productivity by Triangulating Failures**<br>Richard Leveille, Synopsys Inc. |
| 11:10-11:55 AM | | **Quality Engineering for DevOps Customers**<br>Dwayne Thomas, CrowdCompass | **The Changing Role of Software Tester**<br>Anna Royzman, Liquidnet | **Using AOP Techniques as an Alternative Test Strategy**<br>Lian Yang, Yachi Tech Consulting |
| 12:00-1:25 PM | **Lunch Program – Deep-Dive Discussions to Choose From** Level 3 Terrace | | | |
| | **INVITED SPEAKER** | **AGILE DEVELOPMENT** | **PEOPLE AND MANAGEMENT** | **AUTOMATION AND TOOLS** |
| 1:30-2:15 PM | **Bridging to Offshore Testers**<br>Karen Johnson, Consultant | **Scrum Adoption in our Experience: TGCFO Approach**<br>Vadiraj Thayur, McAfee | **Creating Alignment During Change Efforts: The Bridge Between Vision and Execution**<br>Denise Holmes, Edge Leadership Consulting | **Automated Synthetic Exploratory Monitoring of Dynamic Web Sites Using Selenium**<br>Marcelo M De Barros, Microsoft |
| 2:25-3:10 PM | | **How to Fail at Agile Without Really Trying**<br>Heather Wilcox | **Cultivating Software Quality Engineers**<br>Tim Farley, Cambia Health Solutions | **Transition From a Rapid Prototyping to Programmatic Test Framework**<br>Robert Zakes, Brenden Beaman, Oregon Secretary of State |
| 3:15-3:45 PM | **Break in Exhibit Hall** Level 2 Mezzanine | | | |
| | **INVITED SPEAKER** | **AGILE DEVELOPMENT** | **PEOPLE AND MANAGEMENT** | **AUTOMATION AND TOOLS** |
| 3:45-4:30 PM | The International Institute for Business Analysis (IIBA) will be presenting technical analysis of the current state of business needs in the software quality fields. | **Success through Failure: How we Got Agile to Work on a Distributed International Team**<br>Erin Chapman, Tektronix | **This Would Be Easy if it Weren't for the People**<br>Danny R Faught, AgileAssets | **MetaAutomation: A Pattern Language to Bridge from Automation to Team Actions for Quality**<br>Matthew Griscom |
| 4:40-5:20 PM | | **The Agile Advantage: How Agile Integration Improves Outcomes For Software as a Service (SaaS) In The Corporate Environment**<br>Gregory Spehar, Mitusis Consulting Inc. | **Double-Loop Learning: A Powerful Force for Organizational Excellence**<br>Jean Richardson, Azure Gate Consulting, LLC | **Enabling ETL Test Automation in Solution Delivery Teams**<br>Subu Iyer, Cambia Health Solutions, Inc. |
| 5:30-7:30 PM | **Conference Social Kickoff – Exhibits & Poster Papers** Level 2 Mezzanine<br>Complimentary hors d'oeuvres & beverages; open to the public | | | |
| 6:30-7:30 PM | TAO (Technology Association of Oregon) Presentation<br>**Blood-Curdling Tales of Testing Terror** Frank D'Andrea, Tater Tot Design | | | |

# PNSQC 2014 – Conference At A Glance

| **Tuesday Oct 21, 2014 • Day Two • Technical Program** | | | | |
|---|---|---|---|---|
| 7:30-8:00 AM | **Registration Open** Level 2 Mezzanine | | | |
| 8:00-9:30 AM | **KEYNOTE: Live Site Quality: the Bridge Between Your Silos**<br>Jon Bach, eBay, Level 3 Auditorium | | | |
| 9:30-10:15 AM | **Break with Poster Papers** Level 2 Mezzanine | | | |
| | **Level 3 Auditorium** | **Level 1 Plaza Room** | **Level 2 Mezzanine Room** | **Level 3 Skybridge Room** |
| | **INVITED SPEAKER** | **MIXED** | **TESTING AND QUALITY ASSURANCE** | **LIFECYCLES, PROCESSES AND METHODOLOGIES** |
| 10:15-11:00 AM | **Scrum + Kanban, Sittin' In a Tree…**<br>Halim Dunsky, SolutionsIQ | **How Do I Get Started with Mobile Testing?**<br>Theodore Chan, Wellero Inc. | **The Path Not Taken: Maximizing the ROI of Increased Decision Coverage**<br>Laura Bright, McAfee | **Continuous Delivery: Bridging Quality Between Development and Customers**<br>John Ruberto, Intuit, Inc. |
| 11:10-11:55 AM | | **Non-Functional Risk Assessment Framework to Increase Predictability of Non-Functional Defects**<br>Vijayanand Chelliahdhas, HCL Technologies Ltd. | **Reducing the Cost of User Acceptance Testing with Combinatorial Test Design**<br>Steven Dyson, Cambia Health Solutions | **Software Quality Strategy Supported by People and Organizations**<br>Charles R Matthews, The Boeing Company |
| 12:00-1:25 PM | **Lunch Program – Deep-Dive Discussions to Choose From** Level 3 Terrace | | | |
| | **INVITED SPEAKER** | **MOBILE** | **PERFORMANCE ENGINEERING** | **LIFECYCLES, PROCESSES AND METHODOLOGIES** |
| 1:30-2:15 PM | **Mobile UX Make or Break**<br>Phillip Lew, XBOSoft | **Testing @ Tango**<br>Amit Mathur, Tango | **Scalability: Pushing the Limits**<br>Neha Rai, Tim Schooley, Tejas Patil, McAfee | **The Fab Experience: How I stopped Whining and Started to Appreciate Process**<br>Michael Stahl, Ron Moussafi, Intel |
| 2:25-3:10 PM | | **Developing Heuristic Based Mobile Apps**<br>Pooja Sinha, Samrat Dutta, IBM, Sundaresan Krishnaswami, Fiberlink | **Early Performance Testing**<br>Eric Proegler, Mentora Group | **Low-Tech ATDD**<br>Kevin Klinemeier, Davisbase, LLC |
| 3:15-3:45 PM | **Break with Poster Papers** Level 2 Mezzanine | | | |
| | **AUTOMATION AND TOOLS** | **MEASUREMENT AND METRICS** | **TBD** | **SECURITY** |
| 3:45-4:30 PM | **Example-Driven Architecture - Moving Beyond the Fragile Test Problem Once and for All**<br>Gerard Meszaros, Consultant | **Quality Data Management**<br>Christopher WH Davis, Nike Inc. | TBD | **Effective Security Testing**<br>Jeyasekar Marimuthu, McAfee |
| 4:40-5:20 PM | **Five Symptoms Your Test Automation Is Dying**<br>An Doan, AKQA | **Building Software That Captures Better Data**<br>Danny Casale, PGE, Daniel Elbaum, Con-Way Enterprise Services | TBD | **How Identity and Access Management Can Enable Business Outcomes and Enterprise Security**<br>Srikanth Thanjavur Ravindran, Suresh Chandra Bose, Ganesh Bose Cognizant |
| 5:30-6:30 PM | **Networking co-sponsored with the Rose City SPIN** Level 3 Terrace<br>Complimentary hors d'oeuvres & beverages; open to the public | | | |
| 6:30-7:30 PM | **SPIN Presentation – Inciting and Inviting a Quality Culture**<br>Jon Bach, eBay | | | |

## PNSQC™ BOARD MEMBERS, OFFICERS, and COMMITTEE CHAIRS

**Doug Reynolds – Board Member & President**
*Tektronix, Inc.*

**Ian Dees – Board Member & Vice President**
*Tektronix, Inc.*

**Tho Tran – Board Member & Secretary**
*Nike, Inc.*

**Bill Baker – Board Member, Treasurer, Program Co-Chair & Operations Infrastructure Chair**

**Brian Gaudreau – Board Member, Audit & Marketing Chair**

**Michael Dedolph – Board Member & Invited Speaker Chair**
*Levi Deal Consulting*

**Kal Toth – Board Member, Conference Chair & Program Co-Chair**

**Shauna Gonzales – Conference Format Chair**
*Nike, Inc.*

**Les Grove – Volunteer Chair**
*Catalyst IT Services*

**Kathy Iberle – Program Committee Co-Chair**
*Iberle Consulting*

**Philip Lew – Social Media Chair**
*XBOSoft*

**Dwayne Thomas – Networking Chair**
*Crowd Compass*

## PNSQC™ VOLUNTEERS

**Rick Anderson**

**Sue Bartlett**

**Chris Baudin**

**Laura Bright**

**Sadie Brundage**

**John Burley**

**Joshua Burley**

**Jeremiah Burley**

**Robert Cohn**

**Moss Drake**

**Bob Goetz**

**Bhushan Gupta**

**Leesa Hicks**

**Aaron Hockley**

**Randy King**

**Michael Larsen**

**Sumit Lohani**

**Cristina Marinovici**

**Launi Mead**

**Bill Opsal**

**Shivanand Parappa**

**Dave Patterson**

**Srilu Pinjala**

**Amith Pulla**

**Robert Ranslam**

**Emily Ren**

**Rajesh Kumar Sachdeva**

**Jeanette Schadler**

**Anurag Sharma**

**Keith Stobie**

**Vadiraj Thayur**

**Patt Thomasson**

**Heather Wilcox**

**Karyn Zerr**

## PNSQC Call for Volunteers

PNSQC is a non-profit organization managed by volunteers passionate about software quality. We need your help to meet our mission of enabling knowledge exchange to produce higher quality software. Please step up and volunteer at PNSQC.

**Benefits of Volunteering:** Professional Development, Contribution & Recognition

**Opportunities to Get Involved:**

- **Program Committee** — Issues the annual Call for Technical Paper and Poster Paper Abstracts. Receives and manages the paper selection and review process and coordinates program layout.
- **Invited Speakers Committee** — Collaborates with the Program Committee to identify and invite leaders in the global quality community to provide conference speakers.
- **Marketing Communications Committee** — Ensures the software community is aware of PNSQC events via electronic and print media; coordinates and collaborates with co-sponsors and other organizations to get the word out. Identifies potential exhibitors and solicits their participation.
- **Operations & Infrastructure Committee** — Develops techniques to enhance communications with PNSQC board and committee members as well as the PNSQC community at large. Responsible for the PNSQC website, SharePoint, and speaker recording.
- **Community & Networking Committee** — Implements networking opportunities for the software community. Manages the social networking channels of communication. Recruits and works with incoming volunteers to place them in committees. Provides programming for the networking opportunities at the conference. This includes the lunch time format and evening sessions. Responsibilities are recruitment of lead participants, compelling topics and titles and structure of the program.

**Contact Us:** by submitting your name in the conference survey which is a link sent to all PNSQC attendees or complete the contact form at www.pnsqc.org>About>Contact and address it to the PNSQC Volunteer Coordinator.

## PNSQC 2015 Call for Abstracts — Technical and Poster Papers

Inspired? Got an idea you want to tell us about? Consider submitting your ideas for PNSQC 2015 *Brewing Software Quality*. All it takes is a paragraph or two and selected authors receive waived fees.

Get more information at www.pnsqc.org, link to the Call for Abstracts page for more details. Become part of the program by submitting an abstract.

# Balancing Agility and Discipline: Bridging the Gaps Between Software and Systems Engineering

## Dr. Richard Turner, School of Systems and Enterprises, Stevens Institute of Technology

The success of agile management and technical approaches in software development has encouraged applying the same concepts to a broad range of activities, including systems engineering. This talk provides some insight into the concepts of discipline, agility, and their relationship within the context of systems engineering. It goes on to address whether or not agile systems engineering is a meaningful concept, and suggests a different approach to evolving a more useful paradigm for contemporary systems engineering.

***Dr. Richard Turner** has over 30 years of experience in systems, software and acquisition engineering. Currently, he is a Distinguished Service Professor at the Stevens Institute of Technology in Hoboken, NJ, and a Principle Investigator with the Systems Engineering Research Center. Richard is a member of the Executive Committee of the NDIA/AFEI Agile for Defense Adoption Proponent Team, the INCOSE Agile SE Working Group, and co-author of the new IEEE Computer Society/PMI Software Extension for the Guide to the PMBOK that spans the gap between traditional and agile approaches. He is a fellow of the Lean Systems Society, a Golden Core awardee of the IEEE Computer Society, and co-author of four books, the latest, The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software, coauthored with Barry Boehm, Jo Ann Lane, and Supannika Koolmanojwong, was published in June by Addison-Wesley Professional.*

Photo: Andrew Hall, PortlandBridges.com

# Balancing Agility and Discipline: *Bridging the Gaps Between Software and Systems Engineering*

Dr. Richard Turner
Stevens Institute of Technology
rturner@stevens.edu

*Pacific Northwest Software Quality Conference*

*Portland, OR* *October, 2014*

## A Balancing Act to Support a Bridge

The Problem

Fundamentals

Balancing as Bridge

A New Framework

Admiring the View

Photo: Andrew Hall, PortlandBridges.com

## The Current (and Future) Environment

Tick Tock
Credit Card
CNN Politics
Wonkbook: Obamacare's Web site is really bad
"The World is Changed…"
response & resources related to Target's data breach
answers to questions about the data breach
Nanotechnology

## Some software-intensive system trends

| Traditional Development | Current/Future Trends |
|---|---|
| Standalone systems | Everything connected (maybe) |
| Stable requirements | Rapid requirements change |
| Rqts. determine capabilities | COTS capabilities determine rqts. |
| Control over evolution | No control over COTS evolution |
| Enough time to keep stable | Ever-decreasing cycle times |
| Stable jobs | Outsourced jobs |
| Failures not critical | Failures critical |
| Reductionist systems | Complex, adaptive, emergent SOSs |
| Repeatability-oriented process, maturity models | Adaptive process models |

## Fundamental Concepts

## The Fundamental System Success Theorem

**A system will succeed if and only if it makes winners of its success-critical stakeholders**

**Proof of “if”:**

- Everyone significant is a winner.
- Nobody significant is left to complain.

**Proof of “only if”:**

- Nobody wants to lose.
- Prospective losers will refuse to participate or will counterattack.
- The usual result is lose-lose.

## The System Success Realization Theorem

**Making winners of your success-critical stakeholders requires**

- Identifying all of the success-critical stakeholders
- Understanding how each stakeholder wants to win
- Having the success-critical stakeholders negotiate among themselves a win-win set of product and process plans
- Controlling progress toward the negotiated win-win realization, including adapting it to change

## Fundamentals of Discipline

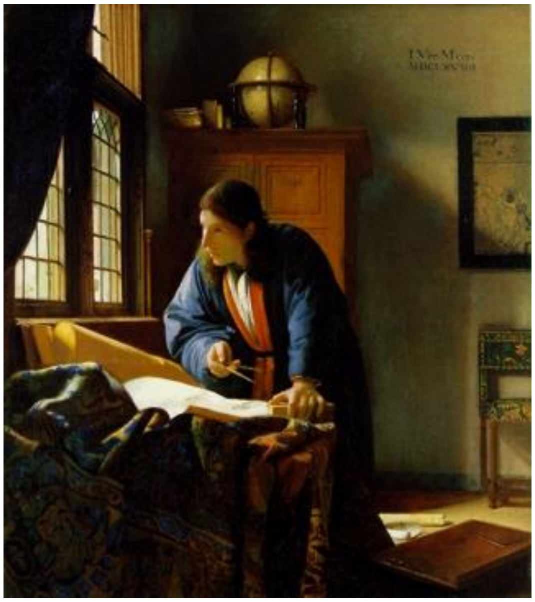
Training

Planning

Architectures

Basic Skills

Infrastructure



Experience

Practice

Principles

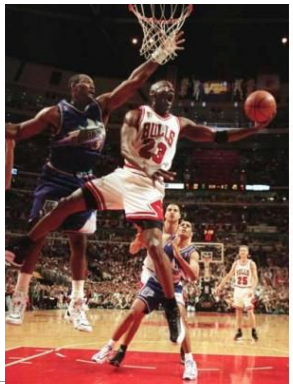
Alignment

Quality

# Fundamentals of Agility...

Flow

Improvisation



Adaptation

Creativity on the Fly

Insight

# The Conundrum as of 2003[1]

**Disciplined (CMMI, PM-BoK):**

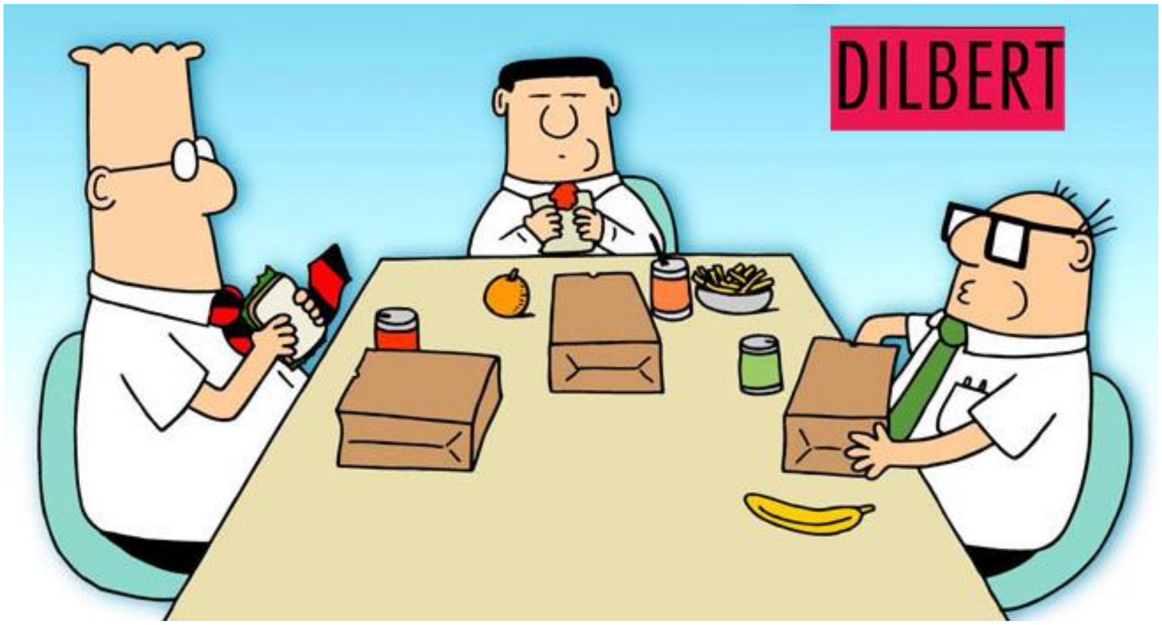
- Generally prefers stability, consistency, control, and stepwise refinement where no work is lost, revisited, or repeated.
- Designed to manage cost and schedule as well as engineering
- Big system focus, thus BDUF is expected
- Overly process-bound (MIL-STD-499, MIL-STD-1521)
- Often equated to systems engineering – a hold-over from the 60s and 70s (NASA SE successes), but now part of the problem

# The Conundrum as of $2003_2$

**Agile (XP, Scrum)**

- Handles rapid change/increased complexity
- Handles emergent, vague, volatile requirements
- Avoids late integration and validation problems
- Little or no documentation
- Seen as hacking or sloppy
- Rapid development cycles and refactoring confuse CM and QA folks

# More realistic media representatives?



## A Decade Later: SW Development

Small teams are best; scaling has proven difficult

Poor discipline when implementing agile practices

Agile management practices (i.e. Scrum) falsely but ubiquitously equated with agile development

Poor interface with business and executive management; system and SoS concerns not well supported

Focus on the small (TDD) with little understanding of the large (continuous integration implications at the system level)

Flow and WIP have become more important, but rare at scale

Wariness toward explicit architectures; now often “surreptitiously” provided by SDEs

Size and complexity still growing; legacy SW exploding; backwards compatibility increases technical debt

## A Decade Later: Systems Engineering

The Vee Model reigns supreme; seen as essential “bad” waterfall

SE decisions often made without software input; have led to ineffective SW solutions

Integrating the SW and System life cycles has proven challenging

More stovepipes have exacerbated complexity issues

Software portion of capability continues to increase while life cycle remains more hardware focused

Human components in complex systems are unpredictable

No real answer for engineering evolving Systems of Systems

Scarcity of qualified systems engineers causes resource bottlenecks

SE seems fixed on creating “agile systems engineering” as a fix

# Need for Balance in Both

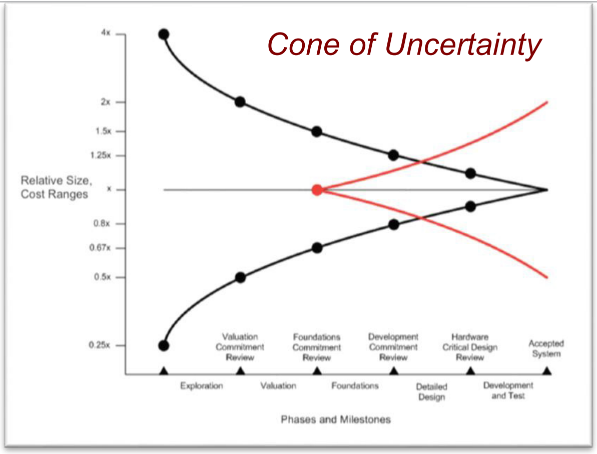
Acceleration of change expands the cone of uncertainty

Software's power is critical to systems, but challenges -ilties

Software must expand focus to the system level

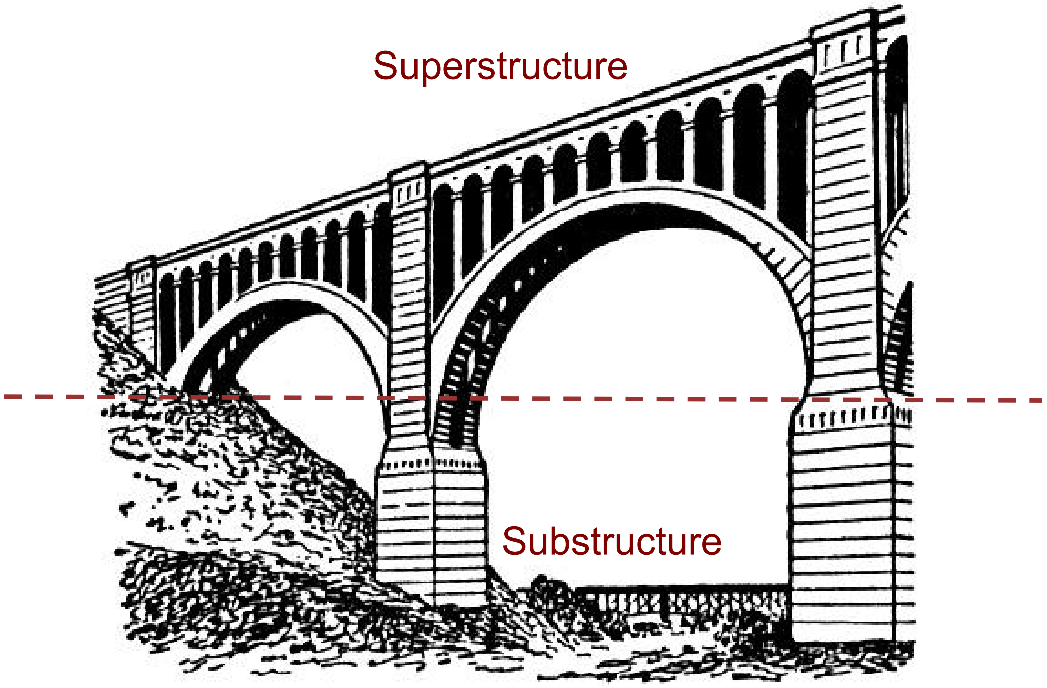
SE must reconcile to software values

SE must continue to



- •Involve, sound out, and manage a broad spectrum of stakeholders
- •Create, maintain, and evolve holistic, essential, and viable concepts of operations, governance models, and architectures
- •Resolve technical issues that cross discipline boundaries
- •Ensure the –ilities are considered and balanced across the system

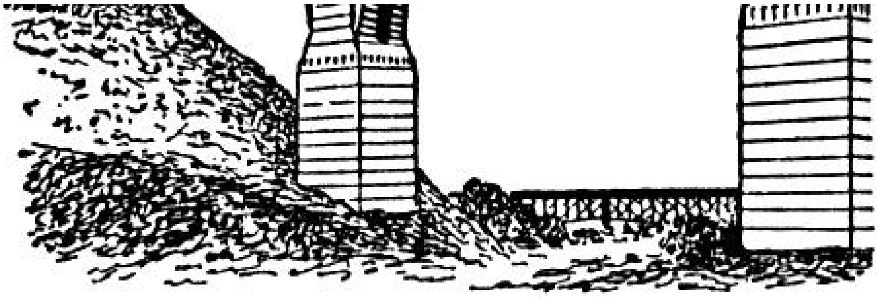
# Key components of a bridge



## The Two Strong Piers of Our Bridge Substructure

Software
Engineering
**Discipline**

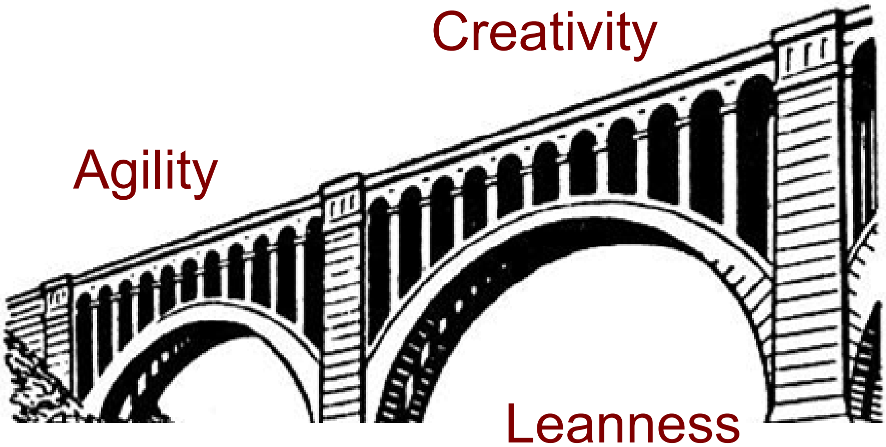
Systems
Engineering
**Discipline**



## Many Means of Developing the Superstructure

Creativity

Agility

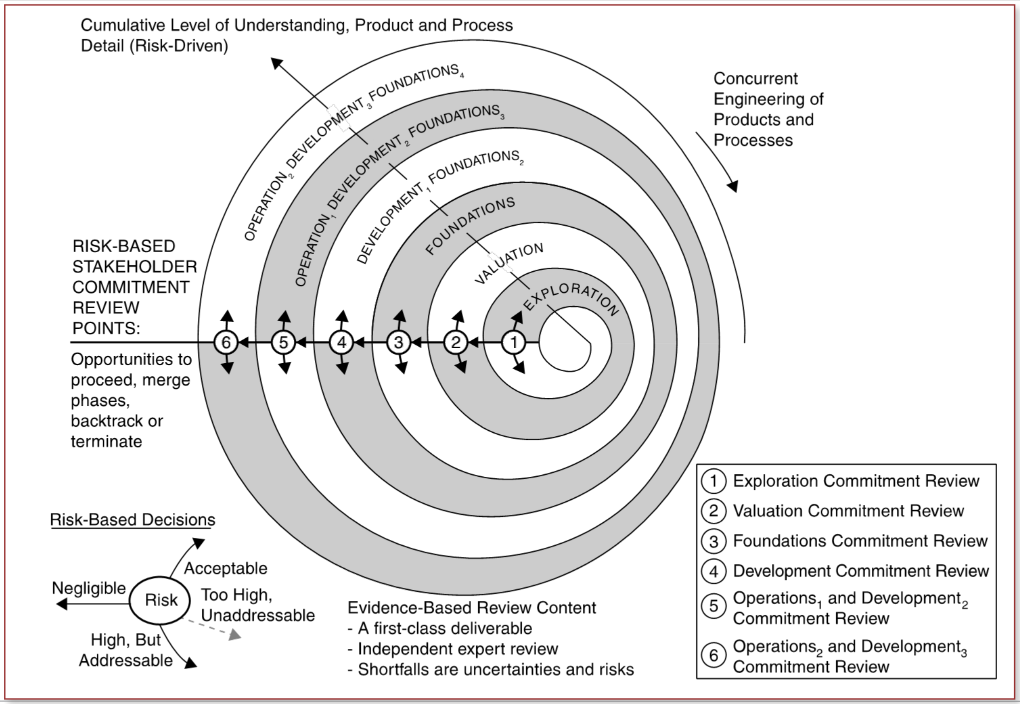


Values

Leanness

# An Illustration of the Need for Balance

# A New Framework for Balancing: The Incremental Commitment Spiral Model



Cumulative Level of Understanding, Product and Process Detail (Risk-Driven)
Concurrent Engineering of Products and Processes
OPERATION$_2$ DEVELOPMENT$_3$ FOUNDATIONS$_4$
OPERATION$_1$ DEVELOPMENT$_2$ FOUNDATIONS$_3$
DEVELOPMENT$_1$ FOUNDATIONS$_2$
FOUNDATIONS
VALUATION
EXPLORATION
6
5
4
3
2
1
Risk-Based Decisions
Acceptable
Negligible
Risk
Too High, Unaddressable
High, But Addressable

RISK-BASED STAKEHOLDER COMMITMENT REVIEW POINTS:

Opportunities to proceed, merge phases, backtrack or terminate

Evidence-Based Review Content
- A first-class deliverable
- Independent expert review
- Shortfalls are uncertainties and risks

1. Exploration Commitment Review
2. Valuation Commitment Review
3. Foundations Commitment Review
4. Development Commitment Review
5. Operations$_1$ and Development$_2$ Commitment Review
6. Operations$_2$ and Development$_3$ Commitment Review

# What is the ICSM?

**Risk-driven framework for determining and evolving best-fit system life-cycle processes**

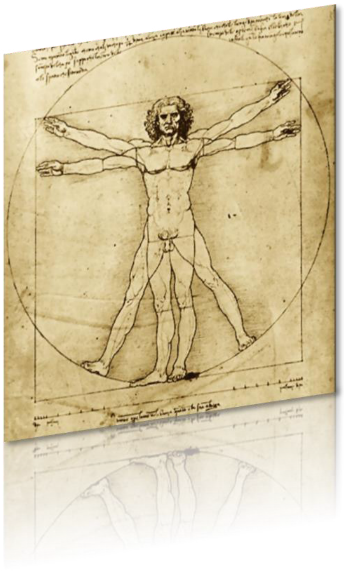
**Integrates the strengths of phased and risk-driven spiral process models**

**Synthesizes together principles critical to successful system development**

- Stakeholder value-based guidance
- Incremental commitment and accountability
- Concurrent multidiscipline engineering
- Evidence and risk-driven decisions

***Principles trump diagrams…***

# ICSM Nature and Origins



**Integrates hardware, software, and human factors elements of systems life cycle**

- Concurrent exploration of needs and opportunities
- Concurrent engineering of hardware, software, human aspects
- Concurrency stabilized via anchor point milestones

**Developed in response to a variety of issues**

- Clarify "spiral development" usage
  - Initial phased version (2005)
- Provide framework for human-systems integration
  - National Research Council report (2007)

**Integrates strengths of current process models**

- But not their weaknesses

**Facilitates transition from existing practices**

- Electronic Process Guide (2009)

# Incremental Commitment in Gambling

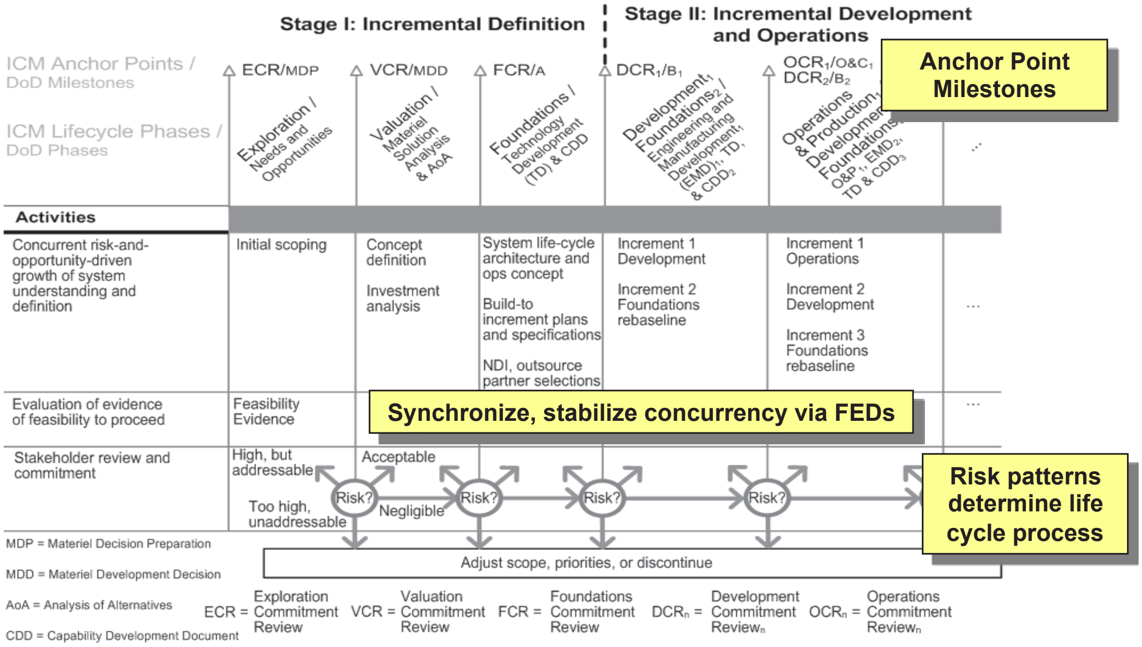
### Total Commitment: Roulette

- Put your chips on a number
  - E.g., a value of a key performance parameter
- Wait and see if you win or lose

### Incremental Commitment: Poker, Blackjack

- Put some chips in
- See your cards, some of others' cards
- Decide whether, how much to commit to proceed

# ICSM Phased View



Stage I: Incremental Definition

Stage II: Incremental Development and Operations

| ICM Anchor Points / DoD Milestones | ECR/MDP | VCR/MDD | FCR/A | $DCR_1/B_1$ | $OCR_1/O\&C_1$ $DCR_2/B_2$ | |
|---|---|---|---|---|---|---|
| ICM Lifecycle Phases / DoD Phases | Exploration / Needs and Opportunities | Valuation / Materiel Solution Analysis & AoA | Foundations / Technology Development (TD) & CDD | $Development_1$ $Foundations_2$ / Engineering and Manufacturing $Development_1$ $(EMD)_1$, TD, & $CDD_2$ | Operations & Production Developmen Foundations $O\&P_1$, $EMD_2$, TD & $CDD_3$ | ... |
| **Activities** | | | | | | |
| Concurrent risk-and-opportunity-driven growth of system understanding and definition | Initial scoping | Concept definition<br>Investment analysis | System life-cycle architecture and ops concept<br>Build-to increment plans and specifications<br>NDI, outsource partner selections | Increment 1 Development<br>Increment 2 Foundations rebaseline | Increment 1 Operations<br>Increment 2 Development<br>Increment 3 Foundations rebaseline | ... |
| Evaluation of evidence of feasibility to proceed | Feasibility Evidence | | | | | ... |
| Stakeholder review and commitment | High, but addressable<br>Too high, unaddressable | Acceptable<br>Negligible | | | | |

**Anchor Point Milestones**

**Synchronize, stabilize concurrency via FEDs**

**Risk patterns determine life cycle process**

MDP = Materiel Decision Preparation

MDD = Materiel Development Decision

AoA = Analysis of Alternatives

CDD = Capability Development Document

ECR = Exploration Commitment Review

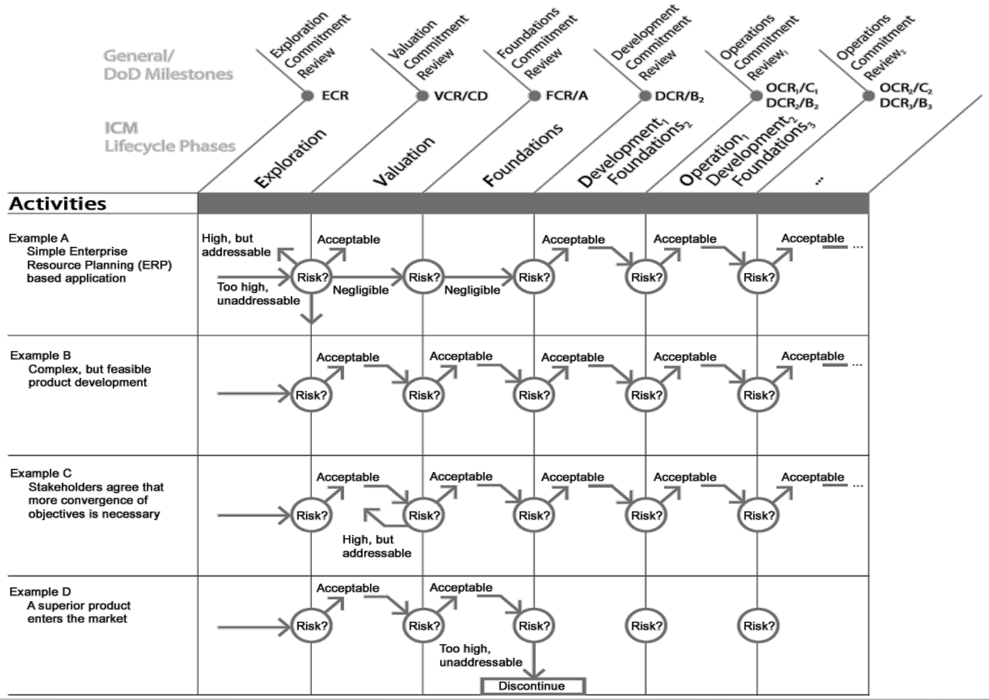
VCR = Valuation Commitment Review

FCR = Foundations Commitment Review

$DCR_n$ = Development Commitment $Review_n$

$OCR_n$ = Operations Commitment $Review_n$

# Different Risk Patterns Yield Different Processes



# Anchor Point Feasibility Evidence Descriptions

**Evidence provided by developer and validated by independent experts that:**

**If the system is built to the specified architecture, it will**

- Satisfy the requirements: capability, interfaces, level of service, and evolution
- Support the operational concept
- Be buildable within the budgets and schedules in the plan
- Generate a viable return on investment
- Generate satisfactory outcomes for all of the success-critical stakeholders

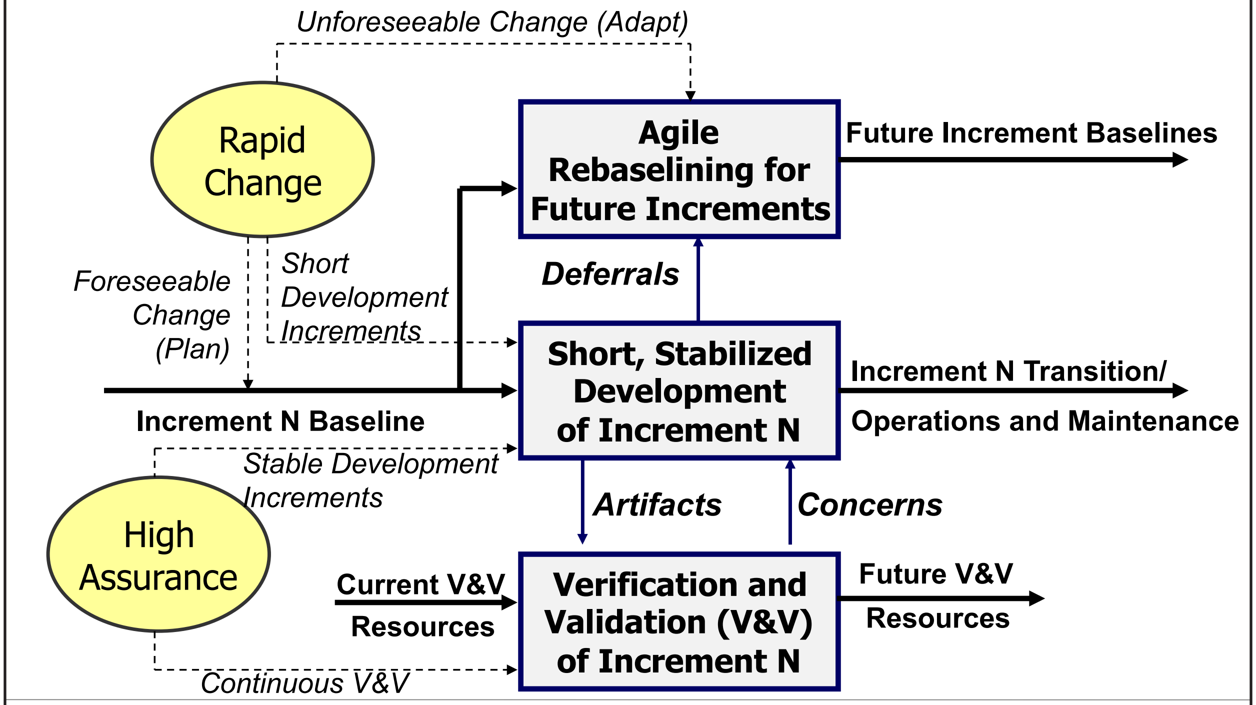
**All major risks resolved or covered by risk management plans**

**Serves as basis for stakeholders' commitment to proceed**

**Synchronizes and stabilizes concurrent activities**

***Can be used to strengthen current schedule- or event-based reviews***

# Risk-Driven Scalable Spiral Model: Increment View for each level of systems-of-interest



Unforeseeable Change (Adapt)

Rapid Change

Agile Rebaselining for Future Increments

Future Increment Baselines

Foreseeable Change (Plan)

Short Development Increments

Deferrals

Short, Stabilized Development of Increment N

Increment N Transition/ Operations and Maintenance

Increment N Baseline

Stable Development Increments

High Assurance

Artifacts

Concerns

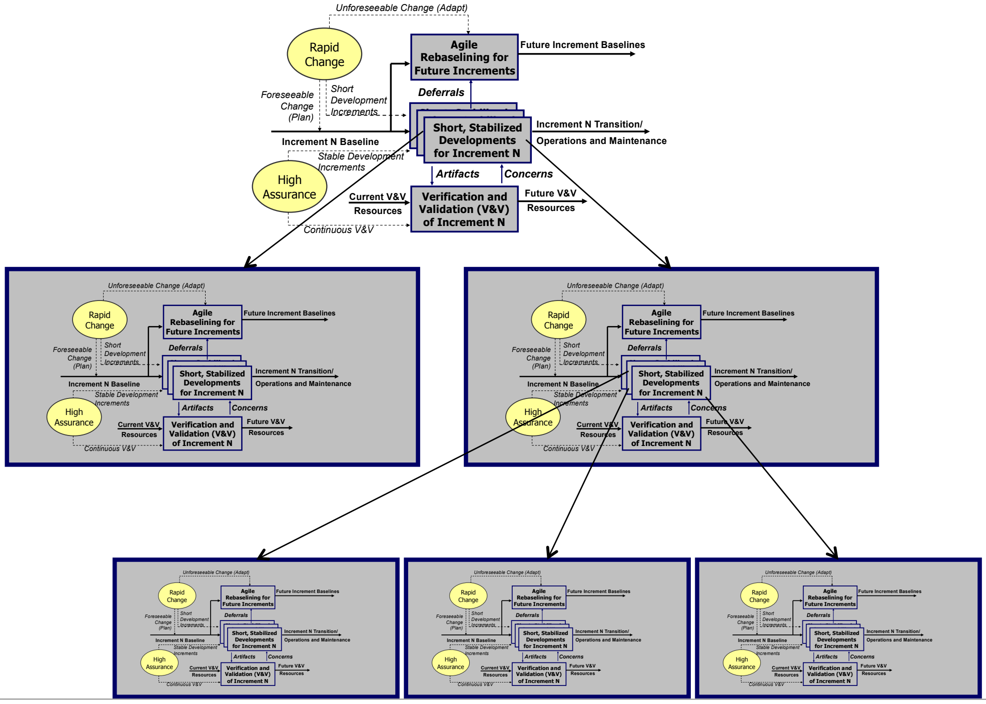
Current V&V Resources

Verification and Validation (V&V) of Increment N

Future V&V Resources

Continuous V&V

# Reality for Larger, Complex Systems



Unforeseeable Change (Adapt)

Rapid Change

Agile Rebaselining for Future Increments

Future Increment Baselines

Foreseeable Change (Plan)

Short Development Increments

Deferrals

Short, Stabilized Developments for Increment N

Increment N Transition/ Operations and Maintenance

Increment N Baseline

Stable Development Increments

High Assurance

Artifacts

Concerns

Current V&V Resources

Verification and Validation (V&V) of Increment N

Future V&V Resources

Continuous V&V

## Principles Trump Diagrams

Stakeholder value-based guidance

Incremental commitment and accountability

Concurrent system engineering

Evidence and risk-driven decisions

***Good example: Symbiq Medical Infusion Pump***

***Counter example: Healthcare.gov***

## Example ICSM Commercial Application: Symbiq Medical Infusion Pump

Winner of 2006 HFES Best New Design Award

Described in NRC HSI Report, Chapter 5

## Symbiq IV Pump ICSM Process - I

### Exploration Phase

- Stakeholder needs interviews, field observations
- Initial user interface prototypes
- Competitive analysis, system scoping
- Commitment to proceed

### Valuation Phase

- Feature analysis and prioritization
- Display vendor option prototyping and analysis
- Top-level life cycle plan, business case analysis
- Safety and business risk assessment
- Commitment to proceed while addressing risks

## Symbiq IV Pump ICSM Process - II

### Foundations Phase

- Modularity of pumping channels
- Safety feature and alarms prototyping and iteration
- Programmable therapy types, touchscreen analysis
- Failure modes and effects analyses (FMEAs)
- Prototype usage in teaching hospital
- Commitment to proceed into development

### Development Phase

- Extensive usability criteria and testing
- Iterated FMEAs and safety analyses
- Patient-simulator testing; adaptation to concerns
- Commitment to production and business plans

## Principles Satisfaction: Symbiq IV Pump

**Stakeholder value-based guidance**

- Extensive involvement of users, buyers, funders, regulators
- Extensive use of prototyping, safety analysis methods

**Incremental commitment and accountability**

- Expanding system definition and evidence elaboration
- Decision to start with composable 1- and 2-channel pumps

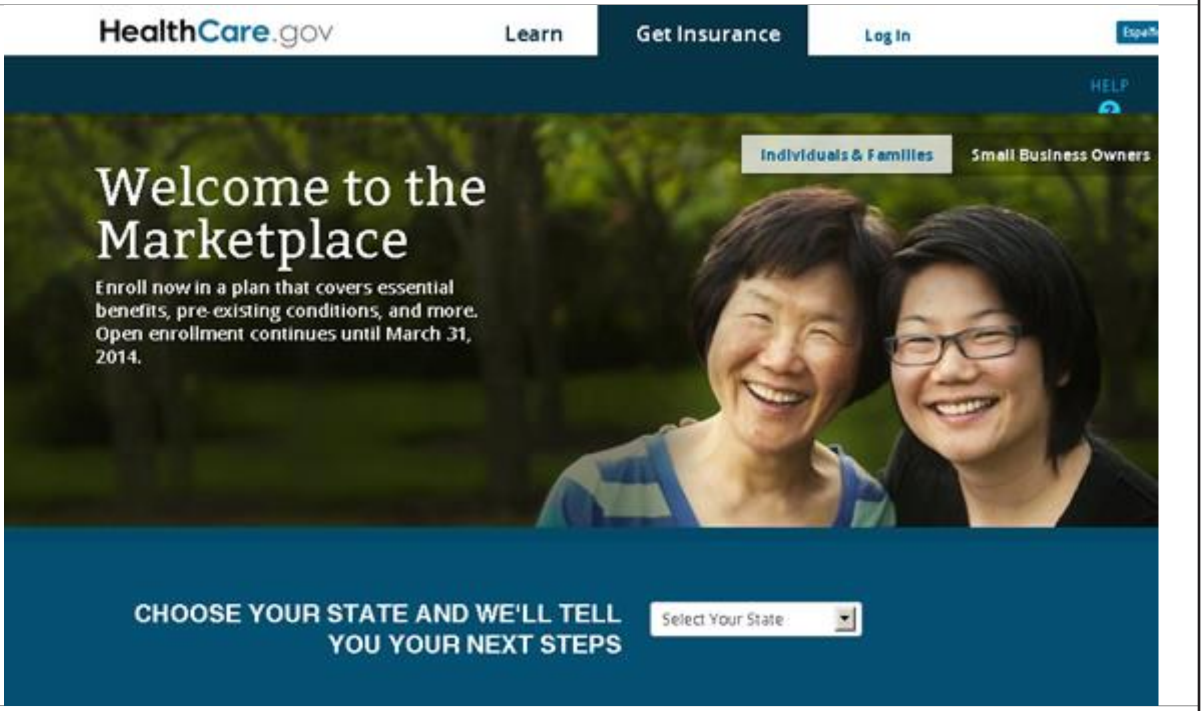
**Concurrent multidiscipline engineering**

- Concurrent evaluation of display, alarm, pump suppliers
- Concurrent definition, evaluation of safety and business cases

**Evidence and risk-driven decisions**

- Evidence-based reviews of technical and business feasibility
- Outstanding risks covered by next-phase risk mitigation plans

## Ways that ICSM Might Have Helped healthcare.gov

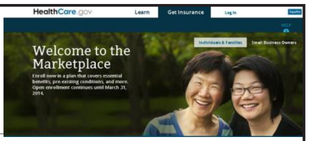


Welcome to the Marketplace

Enroll now in a plan that covers essential benefits, pre-existing conditions, and more. Open enrollment continues until March 31, 2014.

CHOOSE YOUR STATE AND WE'LL TELL YOU YOUR NEXT STEPS

# Healthcare.gov and the 4 principles$_1$



Welcome to the Marketplace

Enroll now in a plan that covers essential benefits, pre existing conditions, and more. Open enrollment continues until March 31, 2014.

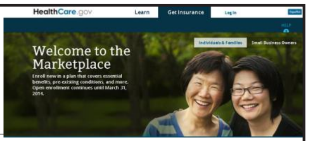
### Stakeholder Value-Based Guidelines

- System stakeholders, numerous and powerful, found it difficult to reconcile win conditions and success models.
- Stakeholder win conditions were not negotiated, documented, and continuously stabilized to ensure that the system would remain acceptable.
- There were no shared understanding of values, assumptions, and expectations; this allowed degeneration into blame assignment after the fact.

### Incremental Commitment and Accountability

- The complexity of the system (>5000 pages in ACA and >78,000 health plans) made the probability of ambiguous and conflicting requirements extremely high.
- Top-level HHS leadership had no significant experience in overseeing a systems development effort like healthcare.gov and little understanding of commitments.
- The development was set up with an all-or-nothing delivery date, resulting in a low probability of success given the complexity and visibility of the system.
- Expectations were initially set extremely high and never managed with respect to the reality of the developing system.

# Healthcare.gov and the 4 principles$_2$



Welcome to the Marketplace

Enroll now in a plan that covers essential benefits, pre existing conditions, and more. Open enrollment continues until March 31, 2014.

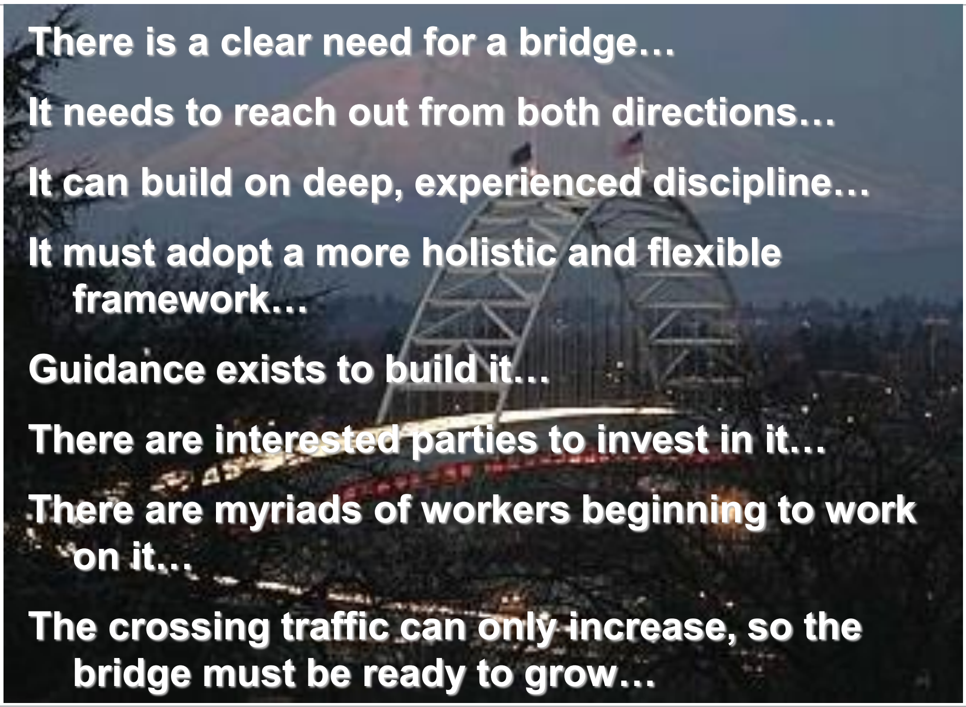
### Concurrent Multidisciplinary Engineering

- Concurrency is fundamental to roll out a large system in a relatively short time frame.
- Adequate expertise in human factors, insurance processing, architecture, and cognitive science was lacking.
- Requisite supply chain communications and engineering process adaptability were absent before contracting $600M of work to 55 companies.

### Evidence- and Risk-Based Decisions

- It is doubtful that validated evidence of the system capabilities meeting the anchor-point milestone criteria was presented to any authority by the developers in the last year of development.
- Decisions were made for political reasons rather than technical reasons, and the technical ramifications were not fully understood.
- Schedule was a key driver, yet it appears that there was no prioritization of key requirements and features to drop requirements or incrementally roll out capability.
- The risks associated with the magnitude and visibility of the site should have led to a collaborative, incremental development strategy not a single deployment.

## Admiring the View



There is a clear need for a bridge…

It needs to reach out from both directions…

It can build on deep, experienced discipline…

It must adopt a more holistic and flexible framework…

Guidance exists to build it…

There are interested parties to invest in it…

There are myriads of workers beginning to work on it…

The crossing traffic can only increase, so the bridge must be ready to grow…

Photo: Andrew Hall, PortlandBridges.com

## For more information….

**The book:**

**The Incremental Commitment Spiral Model:** *Principles and Practices for Successful Systems and Software*

- Available from Amazon, Addison-Wesley, all the usual suspects….

**Co-authors**

**Barry Boehm**

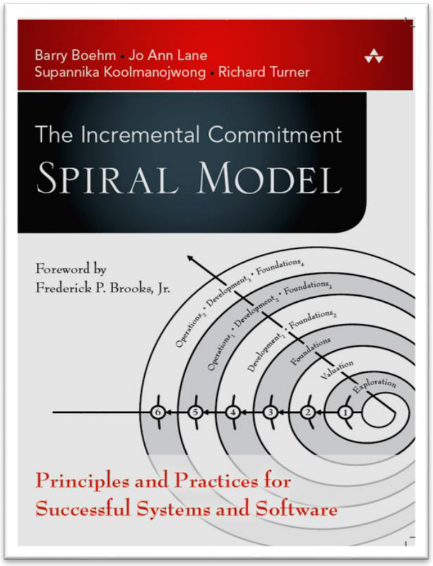
- boehm@usc.edu

**Jo Ann Lane**

- jolane@usc.edu

**Sue Koolmanojwong**

- koolmano@usc.edu



Barry Boehm · Jo Ann Lane
Supannika Koolmanojwong · Richard Turner

The Incremental Commitment
SPIRAL MODEL

Foreword by
Frederick P. Brooks, Jr.

Principles and Practices for
Successful Systems and Software

# Questions?

# Live Site Quality: the Bridge Between Your Silos

## Jon Bach, eBay

Silos aren't a bad thing. Like sovereign countries, they are meant to define and distinguish a culture, identity, and purpose. Furthermore, being in a silo can help you focus and get things done without having to worry about solving every important problem in the company. But just like a nation, if your organization takes an isolationist policy, it can pose some risks.

In this keynote, he talks about his two and a half years of experience as a Quality Engineering Director at eBay, and introduces a concept he calls "Live Site Quality" that he thinks isn't often talked about among software testing professionals. It means the value your customers get while experiencing different activity flows through your online product or service. It's the impression they're left with as they try to solve a problem or meet a need; no matter if it's a simple task or a complex exploration of your site's capabilities. Jon's main idea is "selling" stakeholders on different "Live Site Quality" perspectives and getting eBay colleagues from different parts of the organization to buy the idea that bugs that most affect their Live Site Quality may lie between their team and another.

*With almost 20 years of experience in software testing, **Jon Bach** has held technical and managerial positions at Hewlett-Packard, Microsoft, and eBay. His role as Director of Live Site Quality at eBay involves being on a team dedicated to building "end-to-end" tests (activity flows) to discover bugs on eBay's sites that threaten its core business. Jon is most notable for being the brother of renowned testing expert James Bach, with whom he created Session-Based Test Management. He can usually be found wearing a ball cap, hanging out in conference hallways, talking about testing techniques and philosophy.*

# You're Doing it Wrong: How Your Decision-making Actually Increases Uncertainty and What To Do About It

## Hillel Glazer, Entinex, Inc.

Ever wonder why decisions fail even after the most rigorous analysis? Ever go back and redo the analysis and apply a different option without significantly better results? Subtracting out pathological and group dynamic issues, the problem isn't your analysis. The problem is, almost everything you've ever been taught about decision-making and analysis and the techniques you were asked to use to make decisions has been fatally flawed.

The biggest part of that problem is that you've been completely unaware of it. As a result, we've all been trained to do more analysis or admonished to do a better job of it. It's actually a self-defeating cycle where the "solution" is actually more of the same problem.

If we want better business performance (read: features, quality, profit, etc.) to be what the outside world sees, then we need a better understanding of what's going on on the inside so that better performance on the outside is actually a reasonable expectation and not mere delusional wishful thinking. What's going on on the inside is a function of decision-making. Therefore better performance requires better decision-making processes, not more of the same processes that lead to ineffective decisions.

The outside world sees the naked truth of performance. We know what we like about how that looks. However, how good it looks is a result of inner workings the outside world doesn't see. If the inner workings are an established and sustained high-performance operation, then what's left is to figure out how get there. Therein lies part of the problem. Often the problem is measuring the wrong things and just as often is the compounding error incurred when these wrong (or even right!) metrics are further analyzed deterministically.

Instead of moving towards a decision with great confidence, we're actually unknowingly moving towards a decision with unfounded confidence. In fact, the decision we make in this fashion is fraught with extremely high built-in error and low tolerances for error. All because we don't understand what's going on but we think we do. Worse still, we were trained to be blind to reality while thinking we have deep understanding.

High performance operations use probabilistic decision-making approaches instead of deterministic approaches. Using relatively straight-forward tools, we can obtain performance data change from a deterministic approach to a probabilistic approach that uses performance prediction models. Even without going so far as to work with models, the benefits of being aware of the limitations in our traditional decision-making mechanisms will improve how you make your next product development or service-related decision.

A few simple techniques borrowed from lean thinking can be inserted into the natural, every day, ordinary work to give us everything we need to create and sustain a high performance operation. These same techniques give us non-measurement based tools to help sustain ongoing high performance behaviors by simply doing the basic tenets of the lean activities. Simple yet powerful means of pursuing and realizing high performance results are available to organizations just starting to sort themselves out as well as the highly experienced and mature organizations.

***Hillel Glazer*** *is the founder, Principal, CEO, and "all-around Performance Jedi" of Entinex, Inc., a Baltimore, MD-based management consulting firm made up of aerospace (and other) engineers. Its mission: to bring the same skills and critical thinking used every day in aerospace to solve complex business problems. An internationally recognized authority on bringing lean and agile values and principles into the regulated world, Glazer was selected as a Fellow of the Lean Systems Society in its inaugural fellows induction. He and his company have close ties to the CMMI Institute at Carnegie Mellon University in Pittsburgh, where Glazer serves as an advisor on ways to bring together CMMI (Capability Maturity Model) process improvement models that create high-performance, high-maturity cultures, with lean and agile practices. He also serves as special counsel to the Lean Kanban University and Program Chair of the Lean Kanban North America conference. A University of Maryland-University College graduate, he is the author of the 2011 book, "High Performance Operations," and has written widely on the subject of high performance systems, models and organizations.*

http://prezi.com/aalndxlrkaut/?utm_campaign=share&utm_medium=copy

# Bridging to Offshore Testers

# Karen Johnson, Consultant

As an independent software test consultant, I've had several clients who have outsourced and offshored testing. I've worked with several offshore testing companies on projects using both waterfall and Agile development. Once I also had the opportunity to work directly from within an offshore services company. From the client perspective (the receiver of services) and from the vendor perspective (the provider of services), as well as an independent consultant in the middle of several projects and arrangements, I've seen quite a few issues. Karen shares her thoughts on how to bridge the gaps in offshore testing.

Highlights:

1. Staffing: anonymous resources, occasional rock stars and the revolving door
2. Giving work to another person or team
3. Understanding what gets done; measuring the output and status of work
4. The business and politics of buying and selling testing
5. Addressing cultural-language gaps
6. Working on Agile projects that are not collocated

***Karen N. Johnson*** *is an independent software test consultant. Her client work is often centered on helping organizations at an enterprise level. In recent years, she has helped companies transitioning to Agile software development. While focused on software testing and predominantly working with the testers throughout an organization, Karen helps teams and organizations improve quality overall. Her professional activities include speaking at conferences both in the US and internationally. Karen is a contributing author to the book, Beautiful Testing by O'Reilly publishers. She is the co-founder of the WREST workshop, the Workshop for Regulated Software Testing. She has published numerous articles; she blogs and tweets about her experiences. Find her on Twitter as @karennjohnson (note the two n's) and her website: http://www.karennicolejohnson.com*

# Bridging to Offshore Testers

PNSQC Conference 2014
October 2014

Karen N. Johnson

## Karen N. Johnson

- Software Test Consultant
- Published Author (Beautiful Testing). For a list of published articles, see my website.
- I teach Software Testing
- I speak at conferences
- I am the co-founder of WREST, the Workshop on Regulated Software Testing
- My Website: www.karennicolejohnson.com or www.karennjohnson.com
- On Twitter: @karennjohnson

# BRIDGING TO OFFSHORE TESTERS

As an independent software test consultant, I've had several clients who have outsourced and offshored testing. I've worked with several offshore testing companies on projects using both waterfall and Agile development. Once I also had the opportunity to work directly from within an offshore services company. From the client perspective (the receiver of services) and from the vendor perspective (the provider of services), as well as an independent consultant in the middle of several projects and arrangements, I've seen quite a few issues. Karen shares her thoughts on how to bridge the gaps in offshore testing.

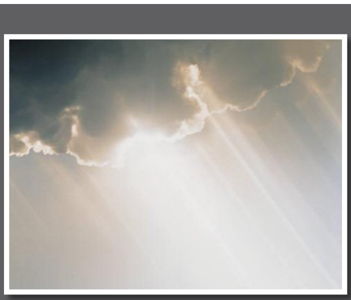
Highlights:

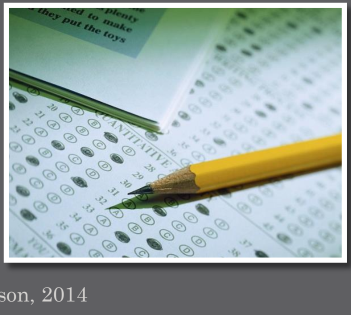
- Staffing: anonymous resources, occasional rock stars and the revolving door
- Giving work to another person or team
- Understanding what gets done; measuring the output and status of work
- The business and politics of buying and selling testing
- Addressing cultural-language gaps
- Working on Agile projects that are not collocated

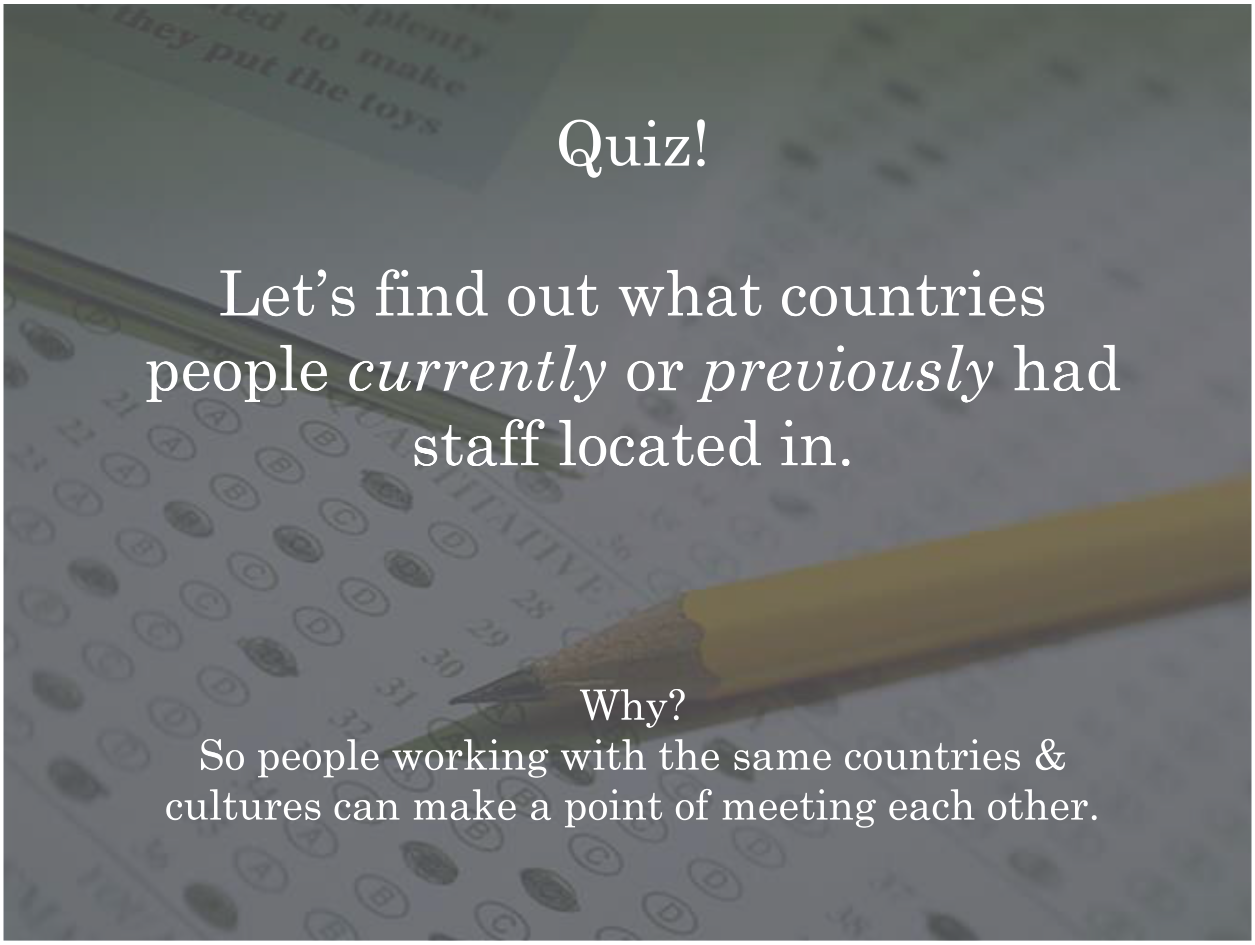
During this talk:

I want to make sure people have a chance to talk, ask questions, share their ideas, move around and have time to meet other people.

- Mini-brainstorming sessions to generate and share ideas.
- A team exercise where you can work with and meet other people
- And I'll give you a couple of "tests" throughout this program. Just kidding! Well – sort of – I do have a few pop quizzes get you thinking!







# Quiz!

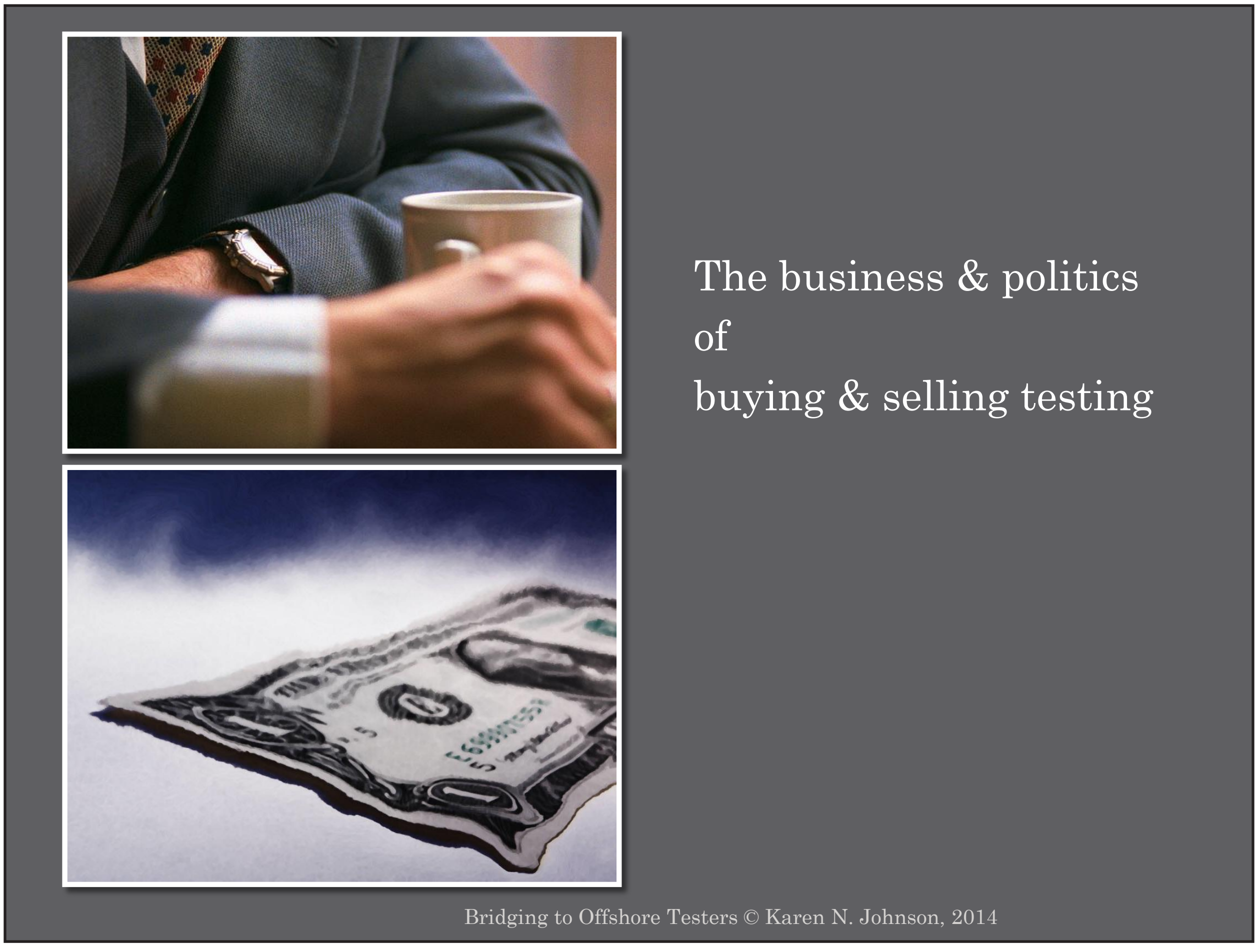
Let’s find out what countries people *currently* or *previously* had staff located in.

Why?
So people working with the same countries & cultures can make a point of meeting each other.

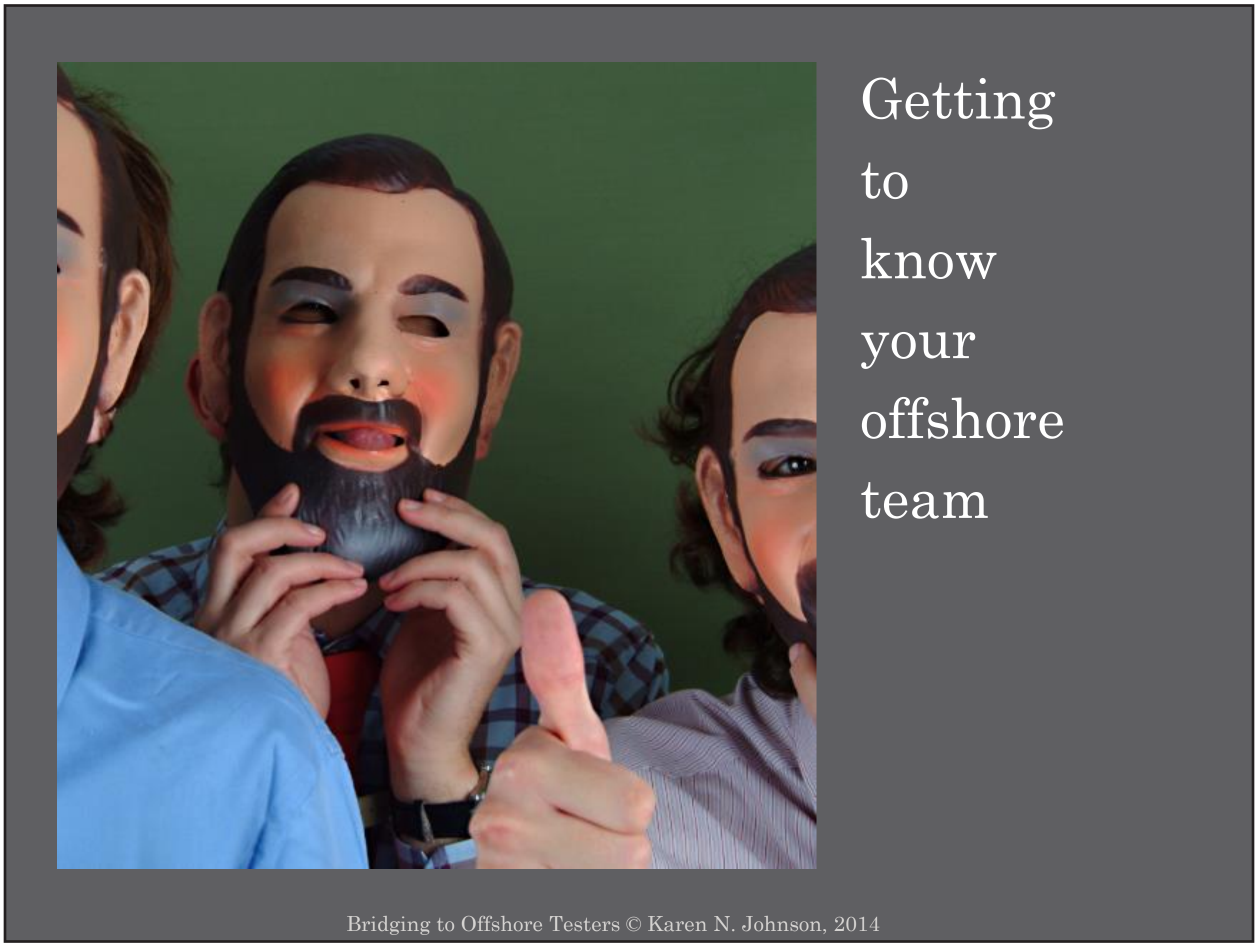
# Working with your Account Manager

- Realize your account manager may not know anything about software testing itself.
- Realize they are in the business of staffing not testing.
- Be mindful that in every conversation or meeting your account manager is assessing whether there will be staffing changes and that this is their primary perspective.

- Your account manager should be your first go-to person for staffing issues.
- The account manager should be part of frequent operational meetings; they should never be surprised in those meetings by the “state of the state.”

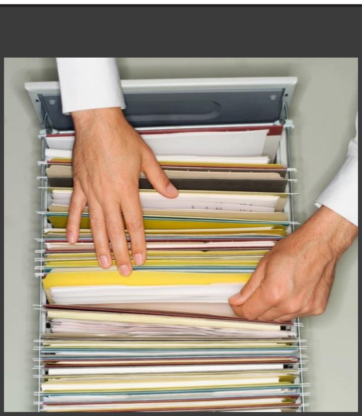


The business & politics
of
buying & selling testing



Getting
to
know
your
offshore
team

- Do you know your current offshore staff members?
- Do you interview replacement staff members?
- Does your onsite staff know the offshore staff?



**Experience Report**

I've had clients with offshore testers where the onsite staff has never met the offshore staff or even seen a picture of the other people. For one client, I collected photos of each offsite staffer and shared the photos.

Next  we (a colleague and I) assembled chairs in the working area and attached a photo to each chair. The area filled up quickly and it became more obvious how big the team really was. We left the working area like this for a few weeks and watched other teams walk through and observe.

*A visual of each person helps to make each person come to life. Add video to meetings whenever possible.*

Tips ~

Getting to know your offshore staff

- Exchange photos of the onsite and offshore staff.
- Hold video meetings (not just audio meetings).
- Host one-on-one sessions with offshore staff just like you currently do with onsite staff.
- Provide input to offshore staff reviews.
- Observe meeting participation and encourage offshore staff to join in.

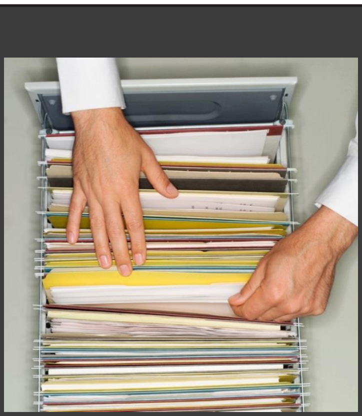
## Brainstorm

Do you have solutions you've used
For getting to know offshore resources?

Let's make a list.

# Identify shadow resources

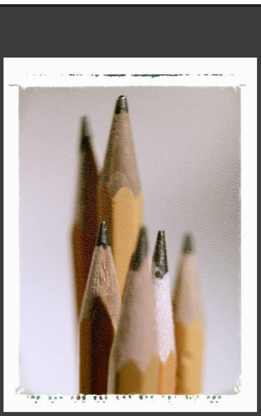
- Do you know what a shadow resource is?
- Shadow resources are backup resources for the existing offshore staff should something happen to a team member, the shadow resource is ready to step in.
- Shadow resources are often not billed directly (distinctly) to a client so shadow resources are often not known.
- Account managers often promise and “give” shadow resources at “no cost” to you so what’s the harm?



**Experience Report**

At one client location, the use of shadow resources meant that for each person working offshore, there was a person assigned who was working alongside them (also offshore). Shadow resources were never introduced (to the client). For the most part, the client was not supposed to know the shadows existed – unless a staffer was out and then the shadow would step up and fill the spot.

*There was no consideration given for whether the shadow resource was trained or able to fill the role.*



Tips ~

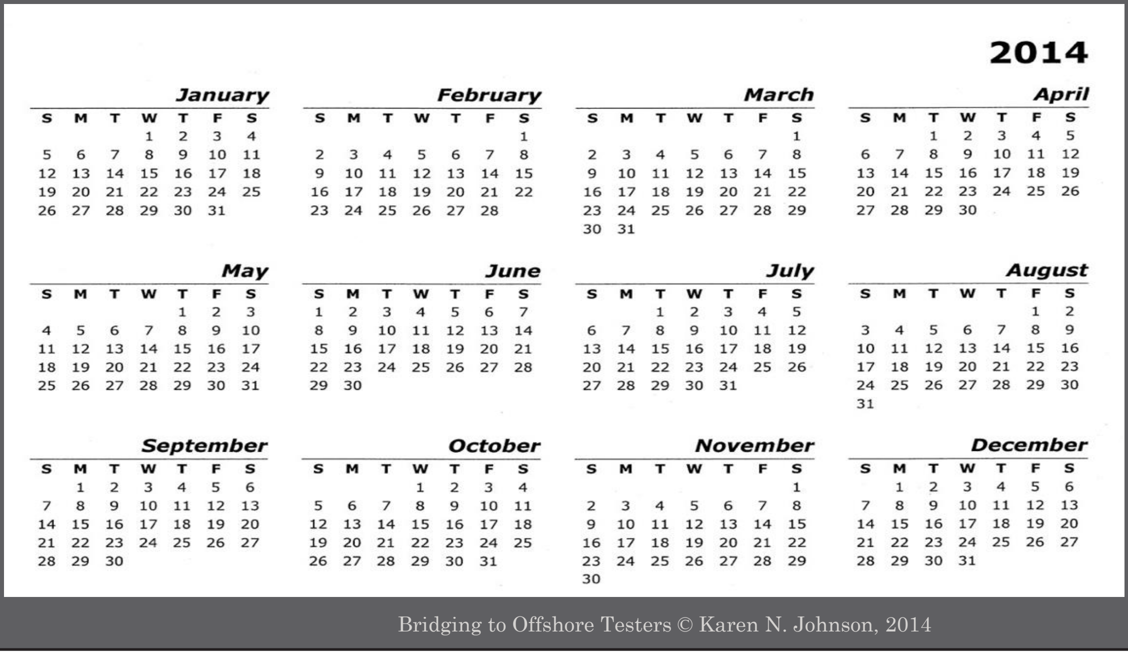
On managing shadow resources

- Ask your account manager if there are shadow resources.
- Request that when/or if a shadow resource is being used or needs to refill a position, that you receive notification.
- If standup meetings and other meetings are visual as well as audio – it will be more obvious when another resource joins the team (for a day, a week or more).

# Working with international holidays

# CALENDAR QUIZ

## HOW MANY INTERNATIONAL HOLIDAYS DO YOU KNOW?



2014

January

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |

February

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | | | | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | |

March

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | | | | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | | | | | |

April

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | | | |

May

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

June

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | | | | | |

July

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | | |

August

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | | | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | | | | | | |

September

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | | | | |

October

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | 29 | 30 | 31 | |

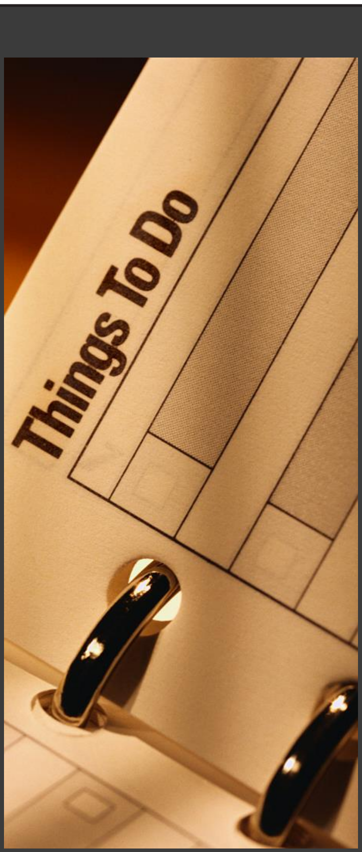
November

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | | | | | | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | | | | | | |

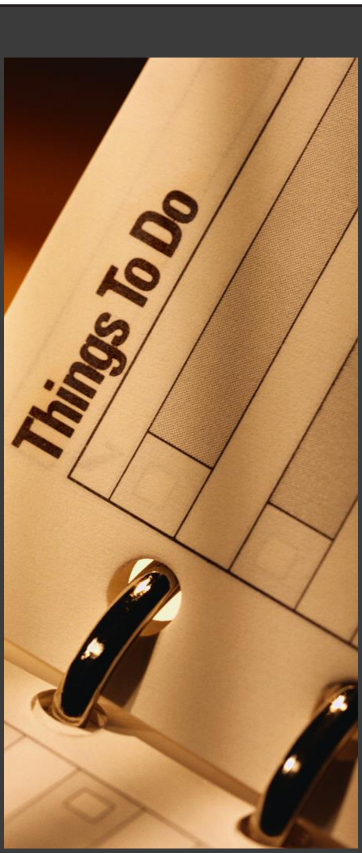
December

| S | M | T | W | T | F | S |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 | | | |

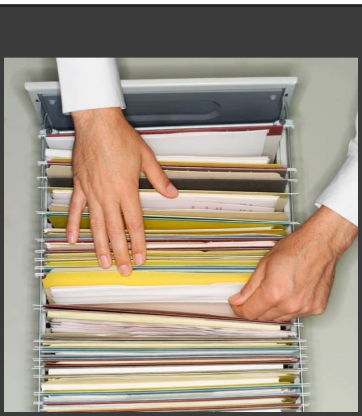
# Getting the work done



- When you assign work tasks, are the tasks understood?
- How do you gauge whether work tasks are  understood?
- Do you receive the needed outcome from assignments?



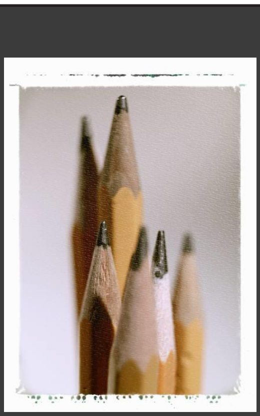
- Sometimes it is necessary to "codify" the work.
- Codifying "thinking" work is challenging because you might *only* get exactly what you ask for.
- You might find you need to learn the difference between the following:
    - Knowing what you need
    - Asking for what you need
    - Giving offshore staff freedom to think outside the box while still following directions



**Experience Report**

There is a term called: codify. Codify in the dictionary is: to put (laws or rules) together as a code or system. In outsourcing, codify means to break down a work task into small steps so that the process is understood. In a thought job such as testing, this can be challenging because to define steps that include how to think, how to find what is not expected (defects) is difficult. What may seem obvious to you, may need to be codified to an offshore staff.

*With one client, the need to codify the steps to test software were difficult to write in detail for me as the process has become intuitive and to define the process explicitly was a challenge.*



Tips ~

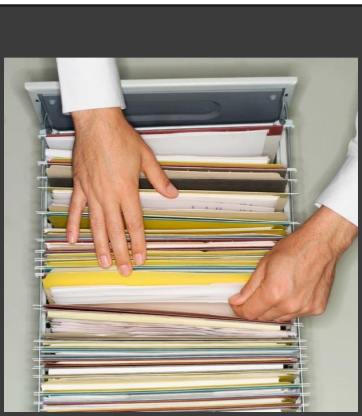
Getting

the

work done

- Use the most basic language to communicate.
- Follow up conversations with emails.
- Do not be vague. Be specific about what you need accomplished.
- Clarify exactly what you want delivered.
- Explain what it will take for a task to be “complete.”

Getting work product reviewed *by the vendor*



**Experience Report**

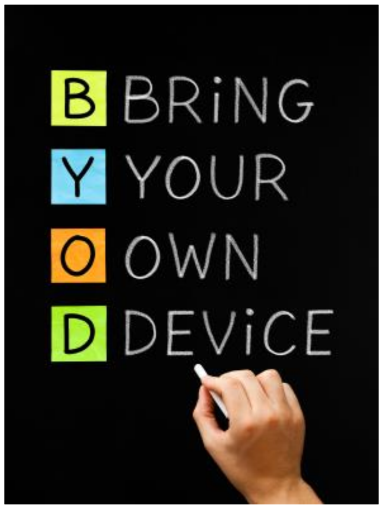
In one client experience, I discovered that the testing staff was mostly fairly junior in experience and the individual testing teams did not have anyone reviewing their work product. Junior testers were writing plans and executing without much supervision or guidance.

*When you have outsourced testing, do you know if anyone is reviewing the testers work product before the work is being turned over to you? Remember, you are the client of your outsourced team.*

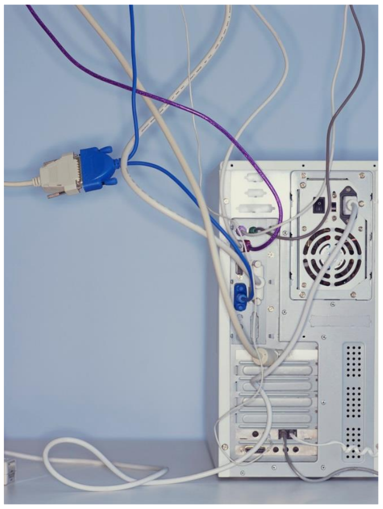
## Supervising offshore work

## Managing from afar …

Test automation



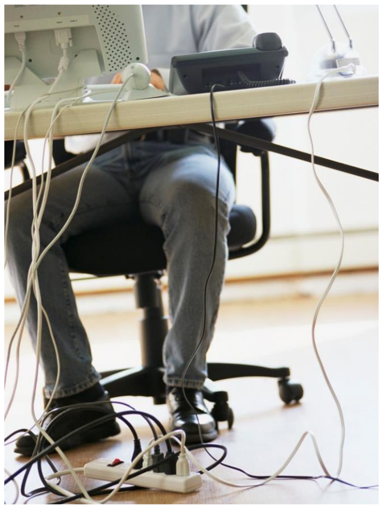
Mobile devices



Equipment

## Managing from afar …

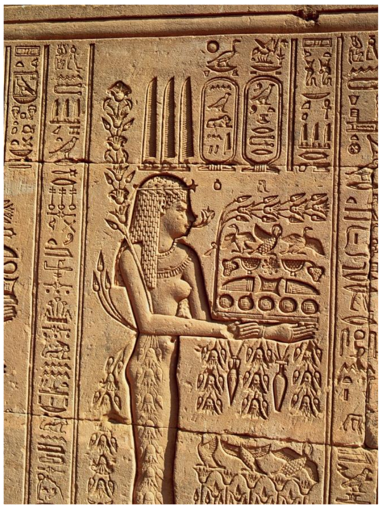
Multiple time zones



Employee conditions

Lines of loyalty

## Managing from afar …



Language

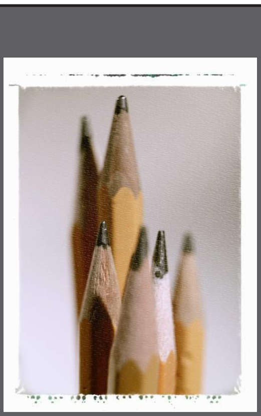
Trust

Education

## Idioms & Business “Speak”

- Don’t assume “basic” business lingo is understood
- Be open to learning expressions from your offshored resources in exchange.
- Learning is not just one way.

*A **fork in the road** is a metaphor, based on a literal expression, for a deciding moment in life or history when a major choice of options is required.*



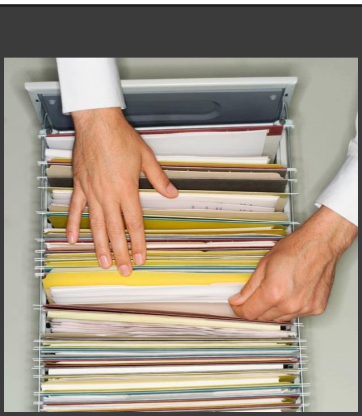
Tips ~

Managing work from afar

- Be involved.
- Ask questions.
- Realize how you and your onsite staff may create conflicting goals and priorities for offshore staff.
- Create an environment where anyone feels comfortable to ask questions.

## Agile teams & offshore staff

- Does your offshore team actively participate in all team meetings including daily standup and retrospectives?
- Does your offshore staff have additional meetings with their employer (your vendor) that you are aware of?
- Does your onsite staff interact casually and regularly with offshore staff?



**Experience Report**

In the most successful environment I have seen with Agile teams that included offshore staff, the video screens were used on a regular basis for standup meetings and other meetings. In addition, IM (instant messenger) sessions were used rapidly and frequently throughout the day. And the offshore staff worked on a rotating basis onsite at the client for weeks/and or months at a time.

Tips ~

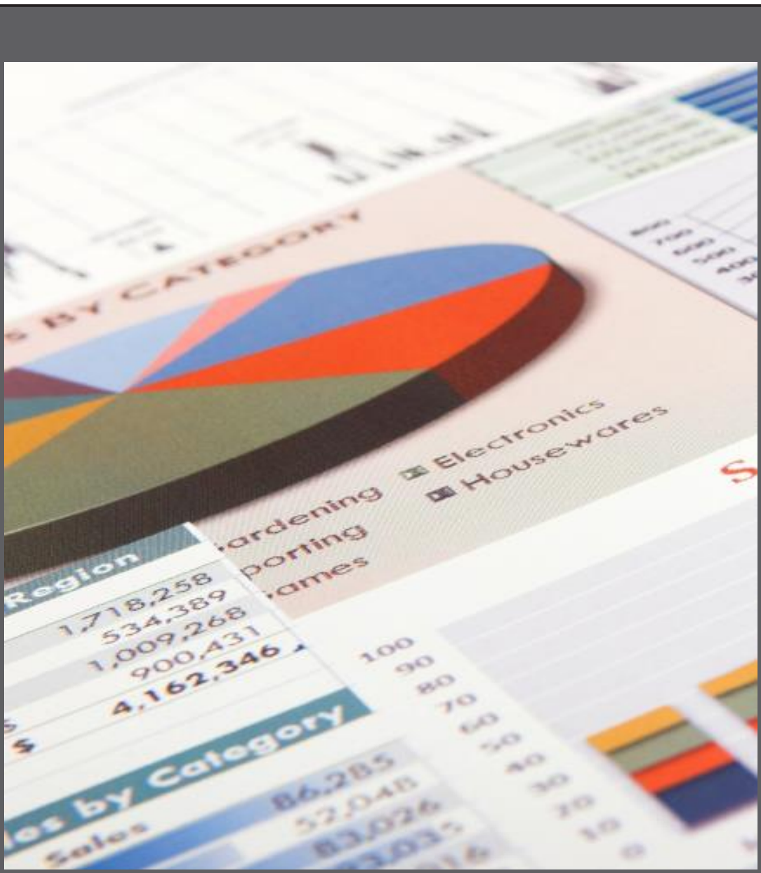
Agile teams & offshore resources

- Frequency (contact & communication) breeds familiarity.
- Communicate to offshore staff that it is not “acceptable” to not participate in meetings.
- Train onsite staff to allow time for offshore staff to join in conversations.

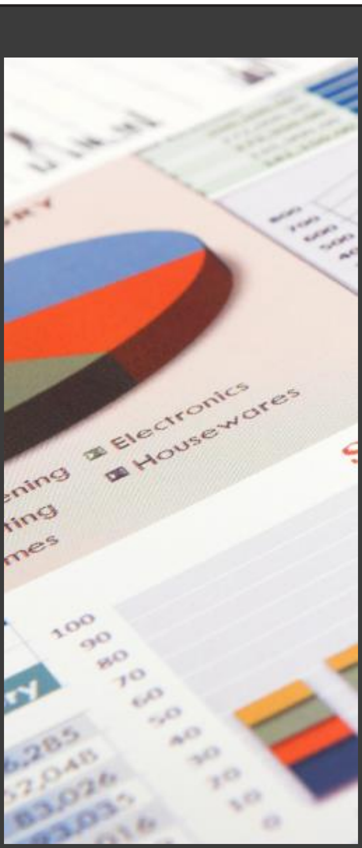
## Working with multiple teams in multiple time zones

Brainstorm
How do you include offshore staff in meetings?

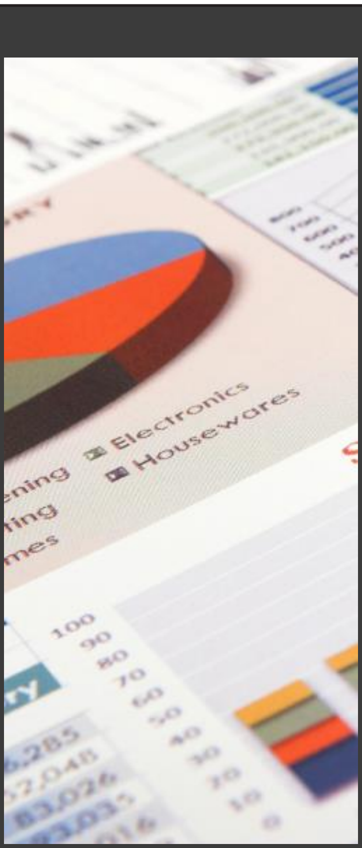
Let’s make a list.



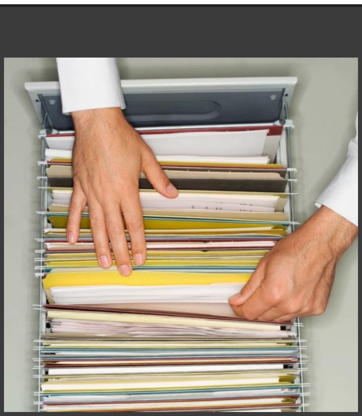
Status Reports: Charts, Graphs & Power Point Slides



- Do status reports provide information that is practical and useful?
- Do you read the status reports you receive?
- Are status reports filled with helpful metrics or meaningless metrics?



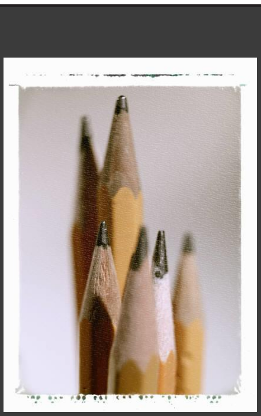
- How much work does it take to generate the status reports you ask for?
- Are you aware that what you ask for in status reporting drives how the team spends their time?
- Do you sense that status reports are not a reflection of how the team actually spends their time?



**Experience Report**

One offshore team proudly produced a monthly newsletter highlighting team accomplishments. The newsletter was filled with numbers such as how many test had been automated or how many defects had been reported.

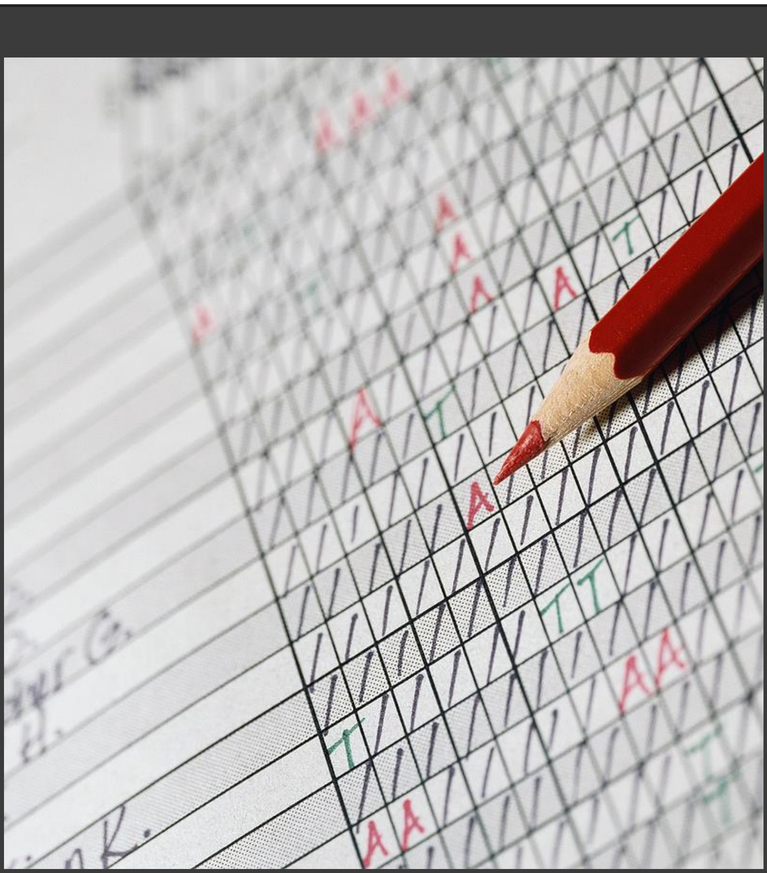
*But the newsletter did not explain the specifics behind what had been automated or whether the defects found were important, high priority or low priority defects. I cringed to think how much time was spent generating such a newsletter.*



Tips ~

Receiving meaningful status reports

- Ask questions.
- If you receive status reports that you do not read and pull any meaningful information from then why continue to have staff spend time generating meaningless reports.



# Managing and rating the vendor

- The location of the vendor has an influence on the work.
- From the book, “Outsource It!” (Pragmatic Press, author Nick Krym)  – the “uninspected deteriorates”
- From a 2005 Harvard Business Review article: “What a firm doesn't measure, it can't offshore well.”
- At the operational level, additional people (beyond your account manager) from the vendor may get involved in calls and plans.

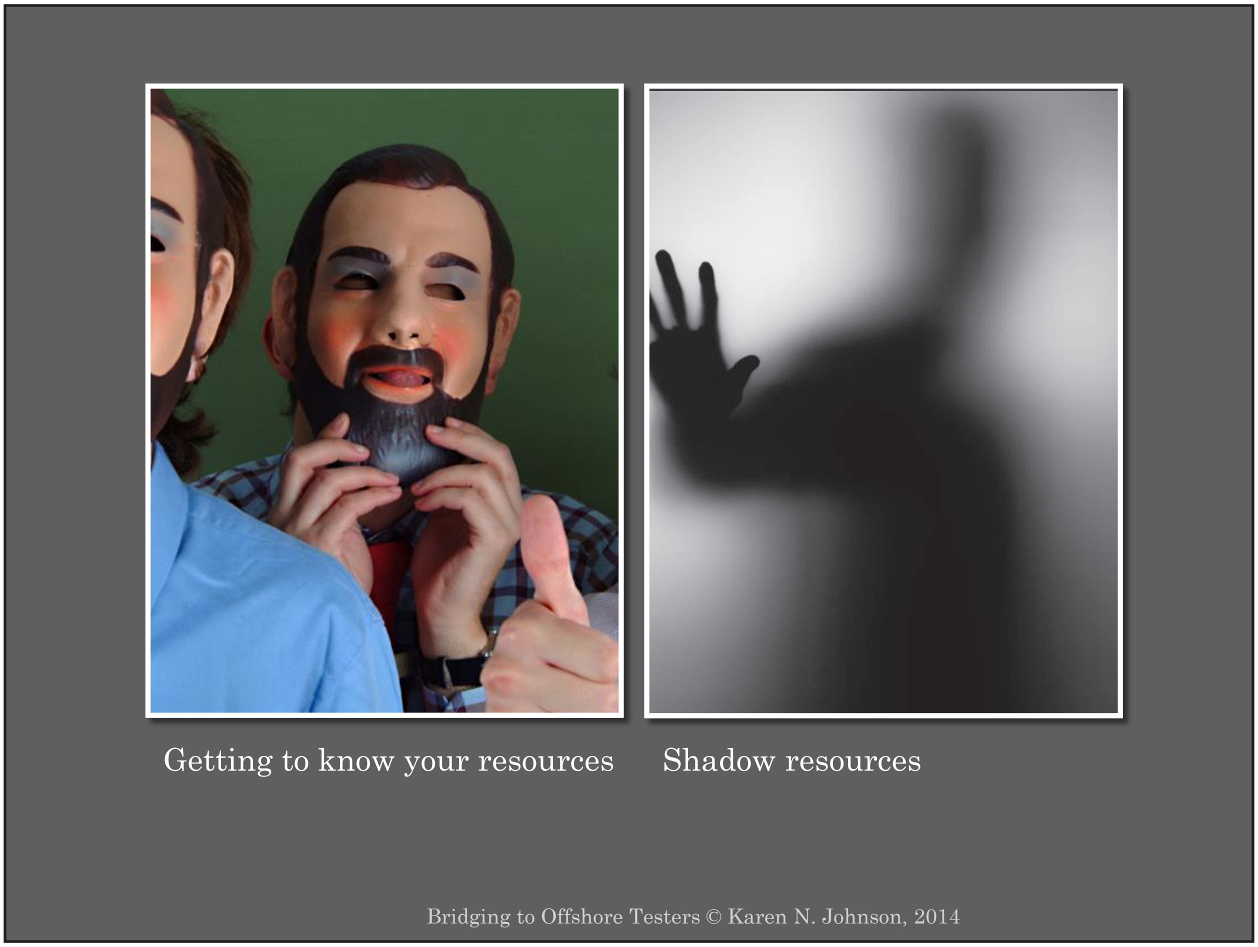
- Martin Fowler outlines how to setup and the importance of an ambassador to bridge the gap between teams and locations.
- Fowler highlights the value of trust. He also highlights the value of the “seeding” visit.

## SUMMARY

- Manage and direct offshore staff
  - Know your staff.
  - Be involved.
  - Ask questions.
  - Review work product.
- Integrate offshore people with your onshore team
- Manage your vendor relationship
- Be mindful of the business of buying and selling testing services.

Account manager

Money



Getting to know your resources

Shadow resources

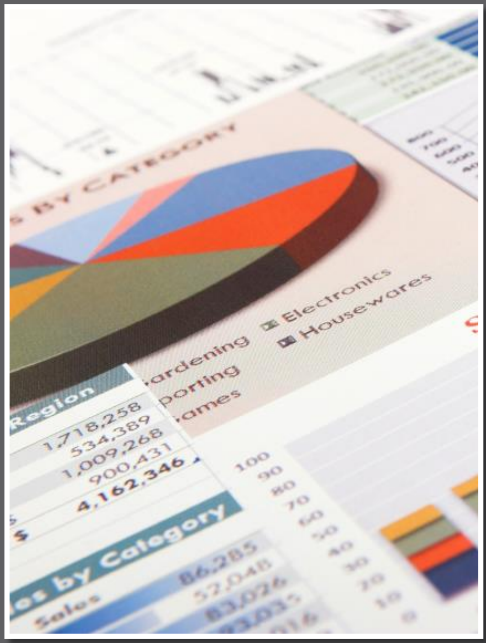
Assigning work

Reviewing work

Supervising

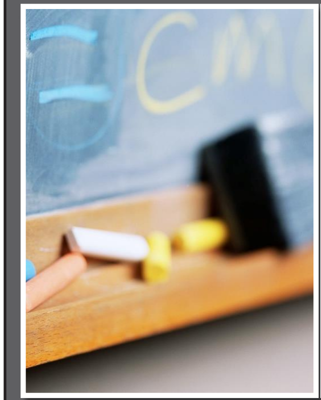
Agile teams

Multiple time zones

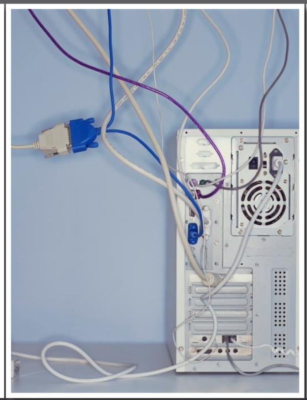


Status reporting

Internal reporting

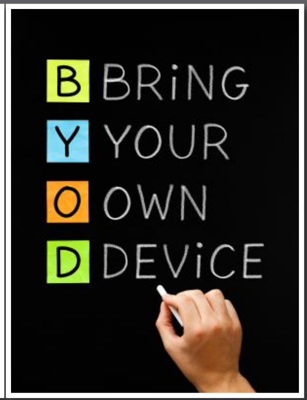


Education

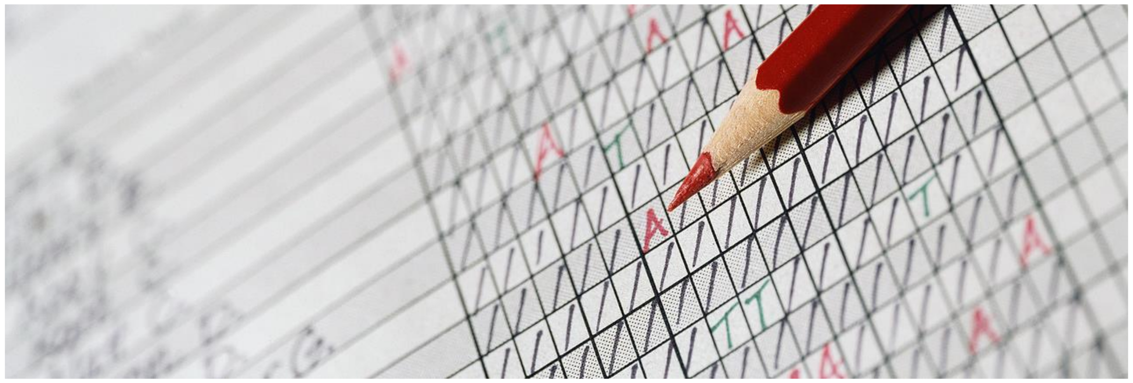


Equipment

Automation



BYOD

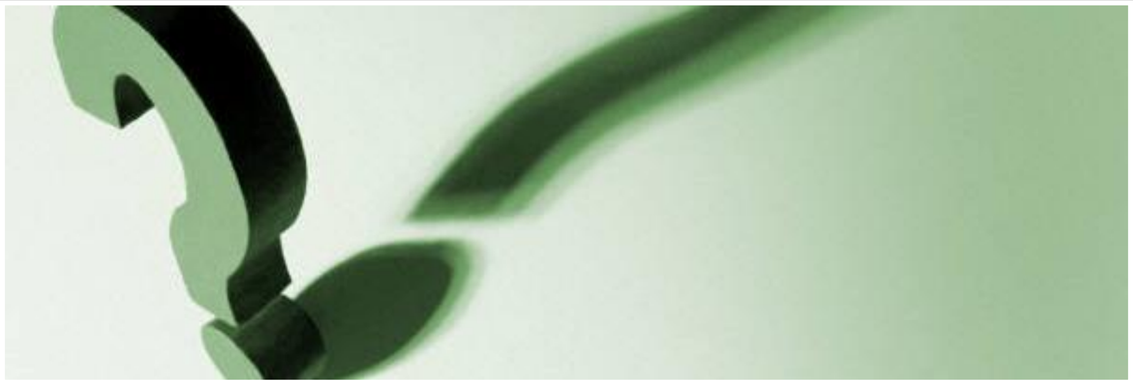


## Working with and rating your offshore vendor

Website:
[www.karennicolejohnson.com](http://www.karennicolejohnson.com)
or
[www.karennjohnson.com](http://www.karennjohnson.com)

Email:
[karen@karennjohnson.com](mailto:karen@karennjohnson.com)

Twitter:
@karennjohnson



# Questions?

# RESOURCES
# FOR MORE INFORMATION

How to Build a Strong Team
InformIT
Karen N. Johnson
May 2010

Outsourcing Confidential
Software Test & Performance magazine
Karen N. Johnson
September 2009

Working Through Language, Time and Cultural Differences
FTP Press 2006
Karen N. Johnson

Harvard Business Review
Getting Offshoring Right
Ravi Aron and Jitendra V. Singh
December 2005

# RESOURCES
# FOR MORE INFORMATION

Using an Agile Software Process with Offshore Development
http://martinfowler.com/articles/agileOffshore.html
Martin Fowler
Jul 18, 2006

Outsource It!
Pragmatic Press
Nick Krym
Jan 10, 2013

# A Case for Scrumban

Halim Dunsky
hdunsky@solutionsiq.com
@halimdunsky

## Abstract

Scrum and Kanban are Agile workflow frameworks that can contribute significantly to software quality. Aspects of a workflow context can be assessed to suggest using one or the other in a particular situation. The two approaches also share many attributes, and have been shown to work well together in mixed and blended configurations, generically termed Scrumban. Several Scrumban configurations are presented and discussed.

## Biography

Halim Dunsky is a Director at Agile consultancy SolutionsIQ, and has over 35 years of experience in leadership and technical roles at firms including Microsoft, Deloitte and Touche, and Bank of America. At SolutionsIQ, Halim has helped shape large Agile transformation and organizational change programs and has used Scrum and Kanban with clients and as a member of the SIQ management team. Halim was Director of Operations and a founding Core Faculty member at Bainbridge Graduate Institute, where he taught systems thinking to MBA students. He received his M.A. in Organizational Systems from Saybrook Graduate School.

## Introduction

> It's a shame that parallel lines never meet. They have so much in common!
>
> – Anonymous

It's likely that many readers have heard or participated in debates – sometimes verging on religious argument – about Scrum and Kanban and why one is better than the other. It seems clear, though, that the best tool will differ depending on the situation. Workflow methods can be important contributors to software quality, and finding a good match for the need is worthwhile.

It also turns out that Scrum and Kanban have a lot in common, and work well in combination. These combined uses can generically be termed Scrumban. *Mixed designs* configure multiple teams such that some use Scrum and some, Kanban. *Blended designs* combine attributes of both into the operations of a single team.

This paper assumes that readers have a foundational understanding of Scrum and Kanban[1,2]. We will first review some of the common attributes of these Agile workflow management frameworks. Next, we'll consider the question of the potential benefits for software quality of the two. We'll also look at contextual indications favoring one or the other. In the final section, several mixed designs will be introduced and discussed, and a sample blended design will be presented.

## Similarities of Scrum and Kanban

It will be helpful first to consider some of the dimensions of similarity between these two Agile workflow frameworks.

**Pull-Based:** Work items are pulled by the team as their capacity allows, they are not pushed onto the team by a scheduling authority. This is fundamental to the reduction of waste and the optimization of throughput. As the authority to pull reflects the autonomy of the team, it can also have a significant beneficial effect on the psychological atmosphere[3].

**Process Policies:** Policies state the criteria under which which an item of work is permitted to advance from one state to another. Policies generally reflect a quality bar. Teams establish policies by agreement; this is again autonomy.

**Incremental and Iterative:** By handling work in small pieces or small batches, both approaches make it natural to build on an early version of a feature with progressive improvement. Although Kanban is not intrinsically iterative, due to the provision for continuous input queuing, a response to feedback concerning a completed work item is easily incorporated in a future work item. This is explicit in Scrum in the expectation that sprint review feedback influences the future product backlog. These approaches allow both for new direction due to learning and for prioritization over time of successive improvements.

**Kaizen:** Continuous process inspection and improvement is built in as a foundational expectation.

## Why Does Workflow Matter for Software Quality?

Let's begin with a quick review of the dimensions of software quality.

It's common in Agile circles to first distinguish *doing the right thing* and *doing the thing right*, where the former refers to requirements fit and the latter refers to quality of construction. Dimensions of software quality can be elaborated in many ways. Here's a simplified rendering of one example[4]:

- Accessibility
- Compatibility
- Concurrency
- Efficiency
- Functionality
- Installability
- Localizability
- Maintainability
- Performance
- Portability
- Reliability
- Scalability
- Security
- Testability
- Usability

I would add Timeliness and feel pretty good about this list.

Many of the attributes of Scrum and Kanban lend themselves to the support of these dimensions or goals:

**Small, well-defined units of work:** A small unit of functionality or work item, particularly when rendered as a well-formed user story, is easier to understand, update, and test than a large specification, which may suffer from overlaps, inconsistencies, unrecognized dependencies, prioritization difficulties, and obsolescence. A modular specification format shares many of the advantages of modularity in software: well-defined work items have low coupling, high cohesion, and well-defined interfaces. Well-defined modularity in specifications can support the emergence of well-defined modularity in the software under construction.

**Incremental and iterative development:** Teams are encouraged to specify and build first the simplest thing that can possibly work. This creates the fastest opportunity to begin to prove the intended concept, supporting mid-course correction as the functionality is extended and enriched. It also makes explicit the opportunity to conduct in stages the work that will support more challenging dimensions of quality.

**Focus on delivering working software:** In contrast to large Waterfall projects that deliver major units of functionality at long intervals, these Agile methods explicitly focus on producing small units of working, tested software at short intervals. This supports the incremental and iterative emergence of a near-continuously functional codebase that lends itself to more frequent releases and assessment of goodness of fit.

**Focus on quality standards:** Definition of Done and Process Policies explicitly constitute a team agreement to hold themselves and each other accountable to a range of applicable quality standards. This is one foundation for a healthy culture of shared responsibility for excellence.

**Focus on bringing testing early into development:** Testing is viewed as ultimately or aspirationally the responsibility of the delivery team. Testing in all its dimensions is brought forward as much as possible to the time when the software is first created, when defects are less costly to repair.

**Continuous stakeholder participation:** Daily contact with Product Owners helps insure that software fit to requirements by clarification and controlling drift. Frequent demonstrations to other stakeholders serve a similar purpose: every two weeks or on a similar short cadence, incremental working software is presented for feedback.

**Frequent product feedback:** Shorter release cycles get the product or system in front of its users more quickly, allowing for more rapid corrective feedback to improve product-market fit. The value here can be explicitly emphasized under conditions of high uncertainty with the Lean Startup cycle of Build-Measure-Learn, which can be used equally well with Scrum or Kanban.

**Get ROI on-stream earlier, stop when it makes sense:** Shorter release cycles can also bring ROI on-stream earlier, which can support the continued funding of valuable software efforts and can help organizations recognize when the value may not justify continued investment.

**YAGNI, 80/20:** YAGNI stands for You Ain't Gonna Need It. If the organization can refrain from trying to do too much, this can reduce the schedule pressure that leads to cutting corners on software quality. Prioritization is a fundamental element of these Agile approaches, which emphasize building the software with the highest business or technical value first. Modular specification and the concept of a continuously evolving backlog, when coupled with appropriate organizational approaches to funding, act to counter historical pressures to jam as much functionality as possible into a requirements definition.

**Kaizen:** The practice of regular team retrospectives is a foundation for a culture of continuous improvement within the team. The practice of making impediments explicit and escalating them as needed can also be a foundation for organizational and technological improvement where the roots of an issue or its mitigations are beyond the team.

## Making a Choice

Many characteristics of a workflow or team context will suggest whether Scrum or Kanban may be a better choice for that situation. Here are some general guidelines.

### Scrum is Good When…

**A sprint cadence provides helpful discipline:** A regular sprint cadence serves to help teams develop good habits of defining, completing, and demonstrating small units of work that can be accomplished within the sprint length. This forcing function is not present in Kanban. The sprint cadence can also simplify some cross-team planning situations and support regular engagement of the stakeholder community.

**Story size is not too big:** A product backlog item needs to be completed within the timespan of a single sprint. Where some units of work seem to be inherently too big for this without awkward or expensive partitioning, Kanban may be a better choice. Note, though, that requirements often arrive in large units of functionality, and learning to break stories down to consumable size can take practice.

**Work items are practical to estimate:** It's important in Scrum to be able to know enough about an item of work that it can be sized to a reasonable approximation before scheduling. Relative sizing methods are generally used for this, and they deliver good results. Another approach to this is to break stories into units that are so small they are all about the same size and can simply be counted. One type of work unit for which either method is problematical is the defect. In the general case it's notoriously difficult to know ahead of time even the approximate size of the job of resolving a given defect. Defect flows are difficult to accommodate within a planned Scrum backlog, but sometimes this is desirable or necessary in order to keep the fix responsibility with those who generated the issue.

**The organization is ready for a significant step:** Succeeding with Scrum can require significant shifts in role expectations, planning, requirements definition, and the working relationship between a software development organization and its business stakeholders.

### Kanban is Good When…

**There is a continuous flow of work item input:** The continuous flow of work through a Kanban team is best matched by a continuous flow of arriving work requests. This is often different from the large batch arrival of work that is typical larger projects and is more suited to a Scrum context.

**Work items tend to be:**

> **Smaller and/or repetitive:** Although a workflow of items of this kind can be handled in Scrum, a Kanban flow seems ideal for maximizing throughput in this situation. Control chart analysis will yield better predictability when items are repetitive or similar than when the workflow has a lot of variation. Predictability is key to being able to establish service level expectations with stakeholders.

**Bigger than a sprint and hard to partition:** In some situations it may not be possible to break stories down into small units that can be completed within a sprint. These can be handled in Kanban without process stress. Note, however, that many items that look big have simply not been decomposed yet.

**Emergent, date-driven, or expected quickly rather than at sprint boundaries:** Very quick turnaround is not natural to Scrum, where planning and delivery occur at intervals of a sprint length, typically 2 or 3 weeks. Kanban is oriented to delivering items immediately as completed. Emergent date-driven items can easily be given priority in Kanban, whereas this can be achieved but is less natural in Scrum.

**Subject to rapidly shifting priorities:** Changing priorities is difficult on short notice in Scrum, where plans are expected to be firm for the duration of a sprint. In Kanban, it is easy to shift priorities in an intake queue; there is no pre-commitment to the next work item to be started.

**Difficult to estimate:** Defect flows, for example, due to the typical uncertainty within many or most items, are difficult to schedule in Scrum but easy to handle in Kanban. Kanban does not require planning based on an estimate of size or duration.

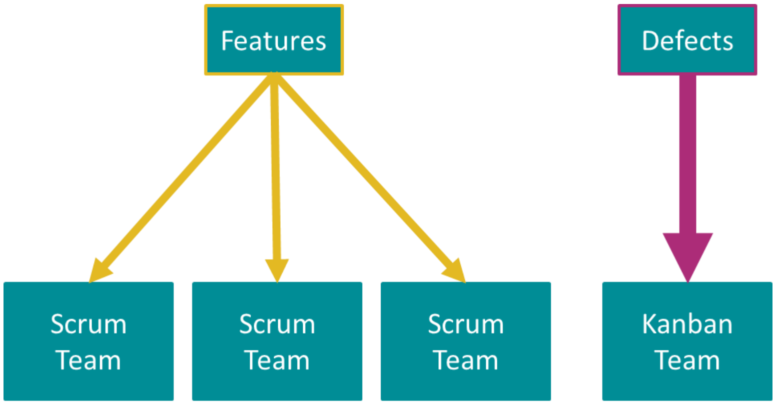
**Organization change needs to be taken in small increments:** Kanban is less prescriptive than Scrum. In some cases it may be more practical for an organization to begin with a Kanban approach built closely on existing practices, then gradually apply improvements as the bottlenecks in the workflow become visible.

## Mixed and Blended Scrumban Designs

Many organizations have found that their needs represent a mix of the considerations reviewed above. Thankfully, the affinity between Scrum and Kanban is so close that it’s quite practical to combine the two approaches, deploying them to work in concert or simply side by side.

Here are a number of designs drawn from real-world examples.

## Rotating Defect Teams (Telco A)



Features

Defects

Scrum Team

Scrum Team

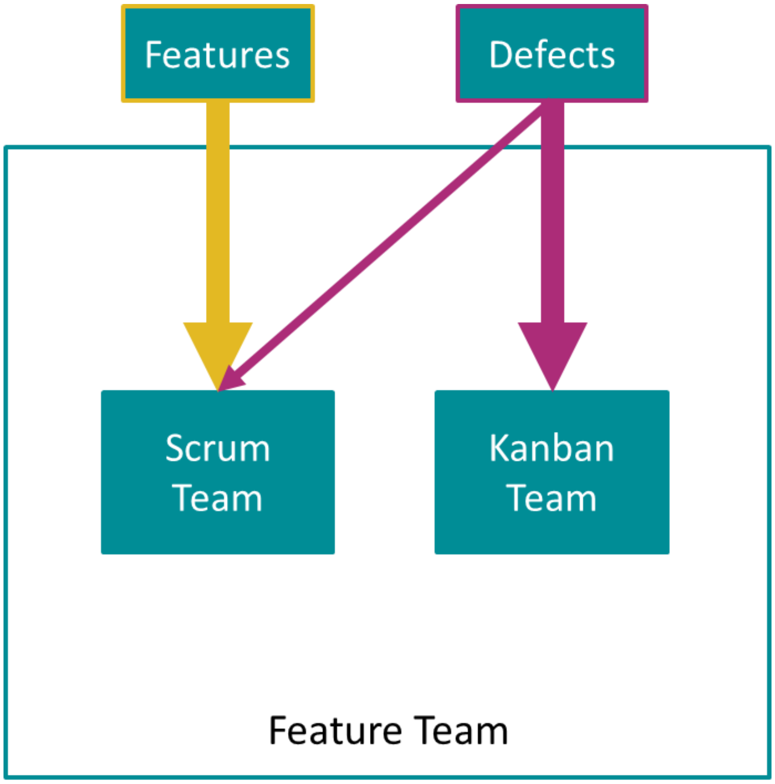
Scrum Team

Kanban Team

| | | | |
|---|---|---|---|
| A | B | C | D |
| A | B | D | C |
| A | C | D | B |
| B | C | D | A |

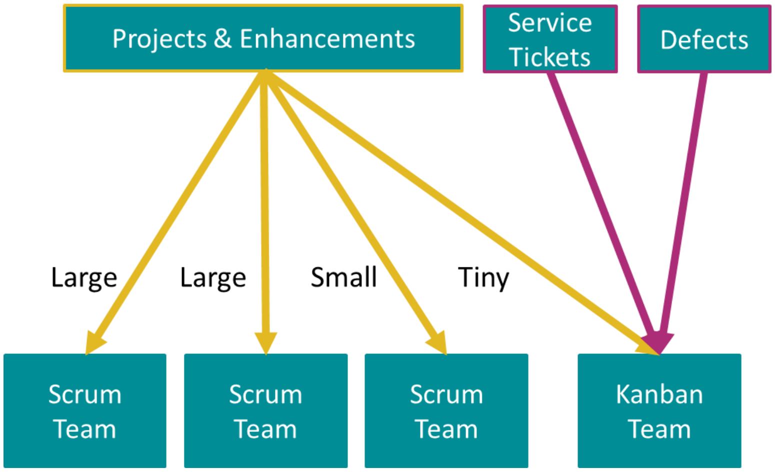
In this example, new feature flow is directed through a set of Scrum teams, while a dedicated Kanban team is tasked with handling defect flow during the endgame of a release hardening period. To avoid consigning one team to perpetual defect jail, the responsibility is rotated with successive releases.

### Defect Sub-Team (Software Company A)



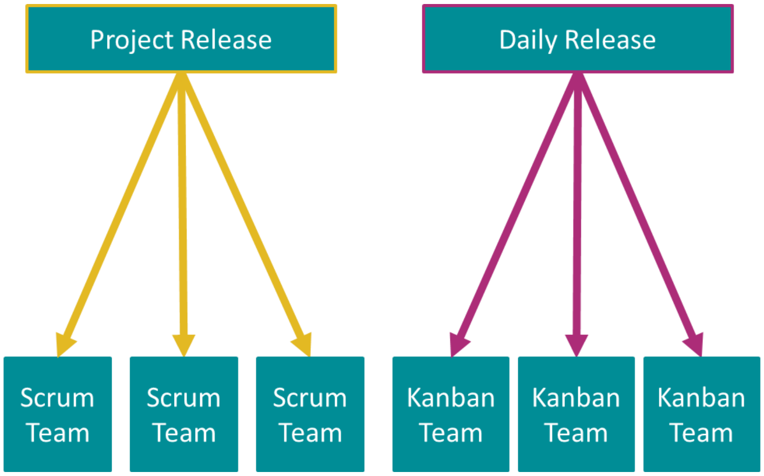
This organization has a feature team structure, with some feature teams comprising multiple Scrum teams. In this case, there is a single Scrum team handling mostly the new feature flow, with a Kanban team dedicated to handling most of the defect load.

### Large and Small Items (Energy Company)



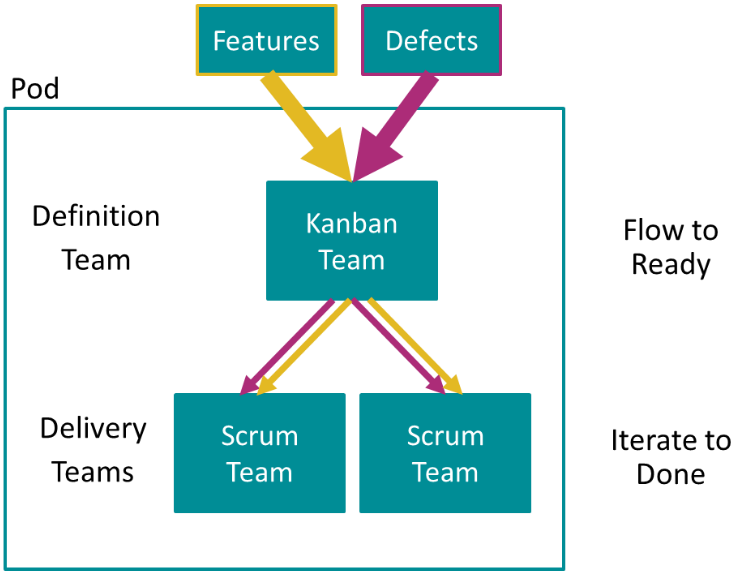
This organization serves both software and operational needs. A set of Scrum teams handles projects and enhancements, with tiny enhancements joining the flow of service tickets and defects to the Kanban team.

### Project and Daily Release (Software Company B)



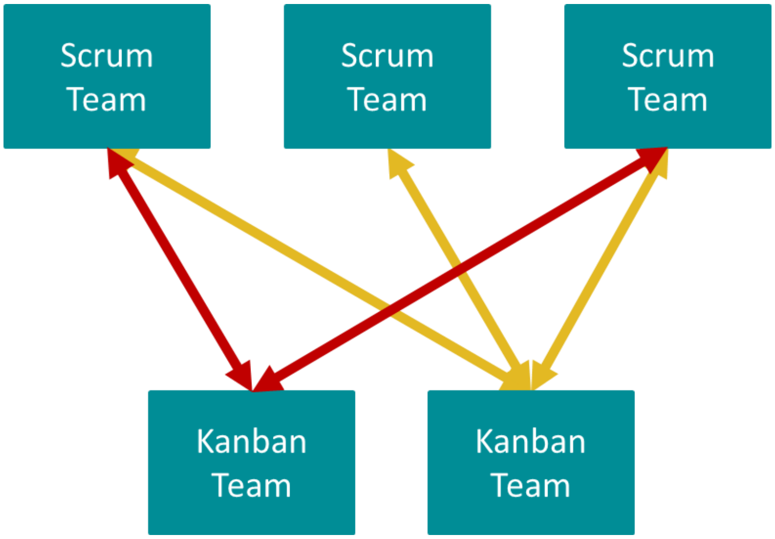
This organization supports releases on a project timescale as well as daily content changes.

### Definition and Delivery Teams (Telco B)



Each Pod in this organization comprises one or more delivery teams supported by a single definition team. The definition team is an extension of the Product Owner concept that brings additional specialist expertise into that function. User stories are made ready by the definition team using a Kanban flow, then consumed by the delivery teams in a Scrum flow[5].

## Specialty Service Queues



When the workflow (value stream) is not encapsulated within the Scrum team, throughput is impeded. The team will experience idle work and/or idle people while waiting for outside parties to execute on dependencies. This arrangement mitigates that situation by putting the outside parties into a Kanban workflow in which transparency and service level expectations can be better established.

## Scrumban Blend

Experience has shown that Kanban teams can benefit from many of the same roles, ceremonies, artifacts, and concepts that provide benefit to Scrum teams. In the following table, elements commonly associated with (though not in all cases formally part of) Scrum and Kanban are arranged according to their approximate conceptual parallels. The Scrumban column contains recommendations for a blended approach.

| Scrum | Kanban | Scrumban | Notes |
|---|---|---|---|
| Sprints | Continuous Flow | Continuous Flow | |
| Sprint Planning | Pull work as needed | Pull work as needed | |
| Sprint Review, Per-Item Demo | Per-Item Completion | Per-Item Completion, Periodic Review | Each item should be demonstrated to the Product Owner or responsible stakeholder at the time of completion. It may be advantageous to also assemble stakeholders periodically to review the cumulated work delivered in the period. |
| Daily Stand-Up | | Daily Stand-Up | |
| Retrospective | | Retrospective | |
| Product Owner | | Product Owner | Product Owner may be unnecessary if items are served FIFO and limited work item clarification, stakeholder negotiation, or prioritization is required. |
| ScrumMaster | | ScrumMaster | The ScrumMaster facilitates a team's activities of continuous improvement and helps them recognize best practices, opportunities for improvement, and self-directed course corrections. These are equally applicable to Kanban teams. |
| Cross-Functional Team | | Cross-Functional Team | Throughput is enhanced when the team does not experience bottlenecks due to limitations in specialist expertise. |

| Scrum | Kanban | Scrumban | Notes |
|---|---|---|---|
| User Stories | Work Items | User Stories, Work Items | User Story format may be unnecessary for simple work items. |
| Potentially Shippable Product Increment | Completed Item | Potentially Shippable Product Increment | Emphasizes that quality and complete testing are part of work completion, not to be added later or by someone else. |
| Definition of Done | Process Policies | Process Policies | |
| Story Point Estimates | Classes of Service, SLAs | Classes of Service, SLAs | Story Point Estimates may provide supplementary value in conjunction with Velocity as a cross-check or while statistics are accumulating |
| Velocity | Cycle Time or Lead Time | Cycle Time or Lead Time | |
| Story Board | Kanban Board | Kanban Board | |
| Tasks usually or often | Usually no tasks | Discretionary | Team choice; more useful for larger work items. |
| Product Backlog | Open & Ready Queues | Open & Ready Queues | |
| Sprint Backlog | Work in Process (WIP) | Work in Process (WIP) | |
| Priorities at Planning | Priorities by class & on demand | Priorities by class & on demand | |
| Burn-Down Chart | Continuous Flow Diagram, Cycle Chart | Continuous Flow Diagram, Cycle Chart | |

## Conclusion

Scrum and Kanban can be mixed and blended to suit your needs. There's no need to be limited to a single tool. Once you understand how these frameworks deliver value, you can be creative in designing a solution to meet your needs[6].

[1] For foundational information about Scrum, see http://www.scrumalliance.org.

[2] For foundational information about Kanban, see *Kanban* by Daniel J. Anderson (2010).

[3] For a discussion of autonomy as a motivating factor for knowledge workers, see www.ted.com/talks/dan_pink_on_motivation. For the book version, see *Drive* by Daniel Pink (2011).

[4] For more detail, see http://SoftwareTestingFundamentals.com/dimensions-of-software-quality.

[5] The concept of Flow to Ready, Iterate to Done originated with Serge Beaumont. See http://blog.xebia.com/2009/07/04/flow-to-ready-iterate-to-done/.

[6] Also see *Kanban and Scrum: making the most of both* by Henrik Kniberg and Mattias Skarin (2009), http://www.infoq.com/minibooks/kanban-scrum-minibook.

SolutionsIQ

# Scrum + Kanban, Sittin’ in a Tree

## Pacific Northwest Software Quality Conference 2014

6801 185th Ave NE, Suite 200
Redmond, WA 98052
solutionsiq.com
1.800.235.4091

# Halim Dunsky

**Director at SolutionsIQ**

**Over 35 years in the business**

**Was TRS-80 owner**

**Agile Transformation**

**Organizational Change**

**Consultant, Coach, Trainer**

**CSM, CSPO, CSP**

**M.A., Organizational Systems**

**@halimdunsky**

# Who's Here Today?

**What disciplines are in the room?**

**Who has used…**

- **Scrum?**
- **Kanban?**
- **Both?**

# Agenda

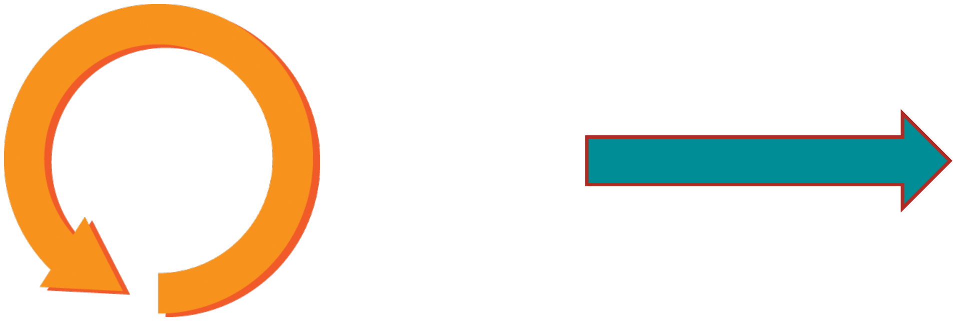
- **Agile Workflow Frameworks: Scrum & Kanban**
- **Workflow and Quality**
- **How to Choose?**
- **Mixed and Blended Designs**

# Agile Workflow Frameworks

## Vive la Différence!

Scrum

Kanban



Iterations

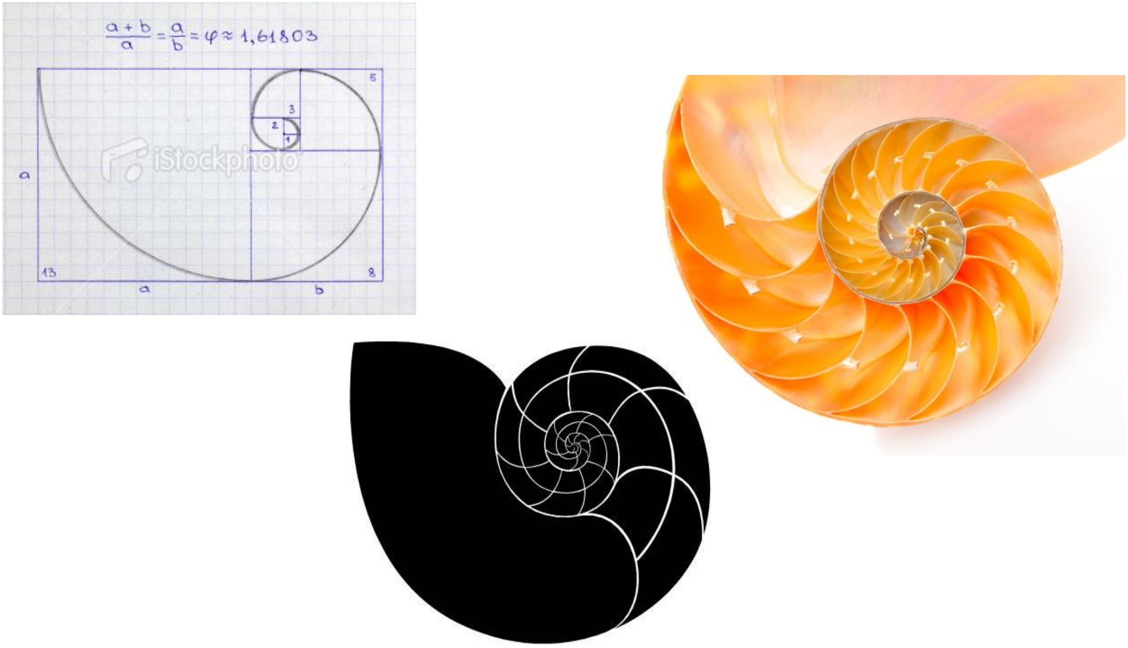
Continuous Flow

# Pull-Based Systems

# Explicit Process Policies

# Incremental

# Iterative



## Improve Collaboratively (Kaizen)

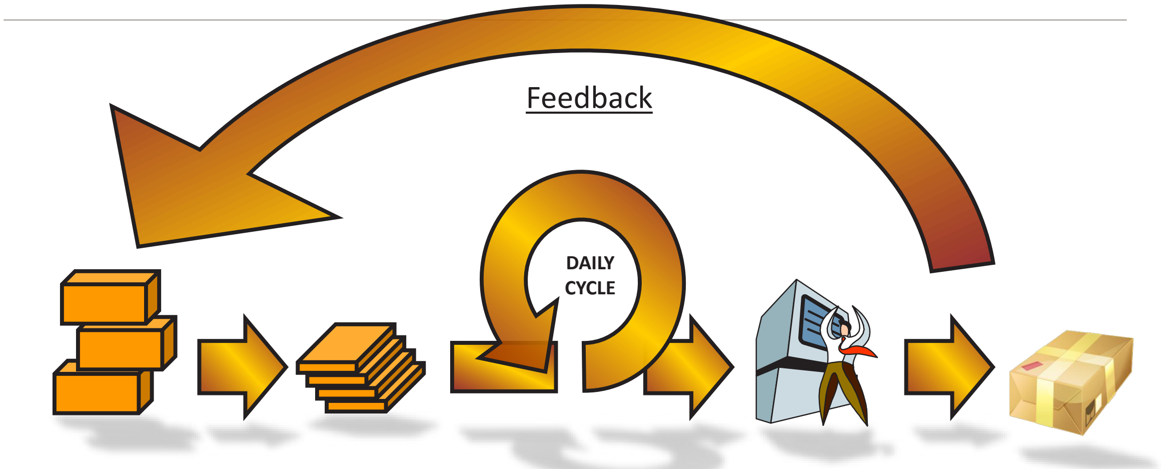
## It All Comes Down to This

- Break the work into little pieces (small batches)
- Prioritize
- Visualize and manage the workflow
- Limit multitasking
- Deliver frequently
- Get immediate feedback and improve
- Work together as a team
- Be good to each other

# Agile Workflow Frameworks

## Scrum

# How Does Scrum Work?



**Product Backlog**
Create or update the Product Backlog, an ordered list of the product features desired by the customer

**Sprint Planning**
Create the Sprint Backlog, a list of the features pulled by team for a Sprint

**Daily Scrum**
A 15-minute meeting, where the team comes to communicate

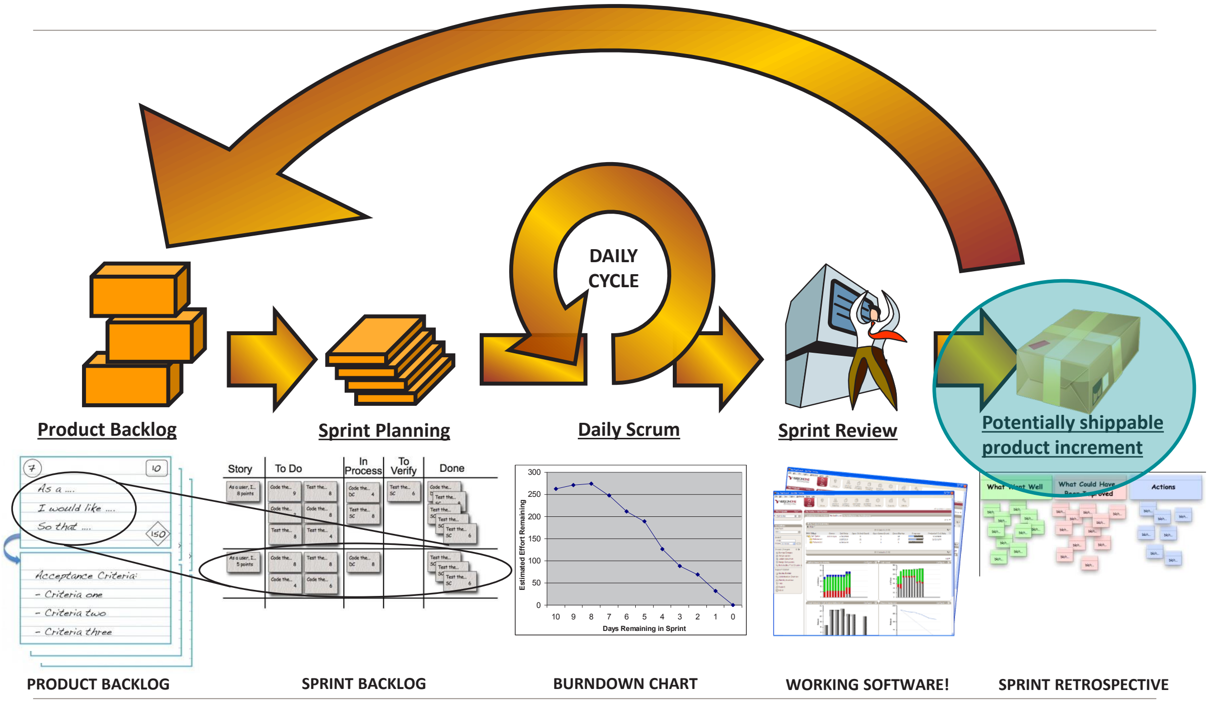
**Sprint Review**
Demonstrate the functional, incremental results of the Sprint

**Sprint Retrospective**
Team meets for process improvement

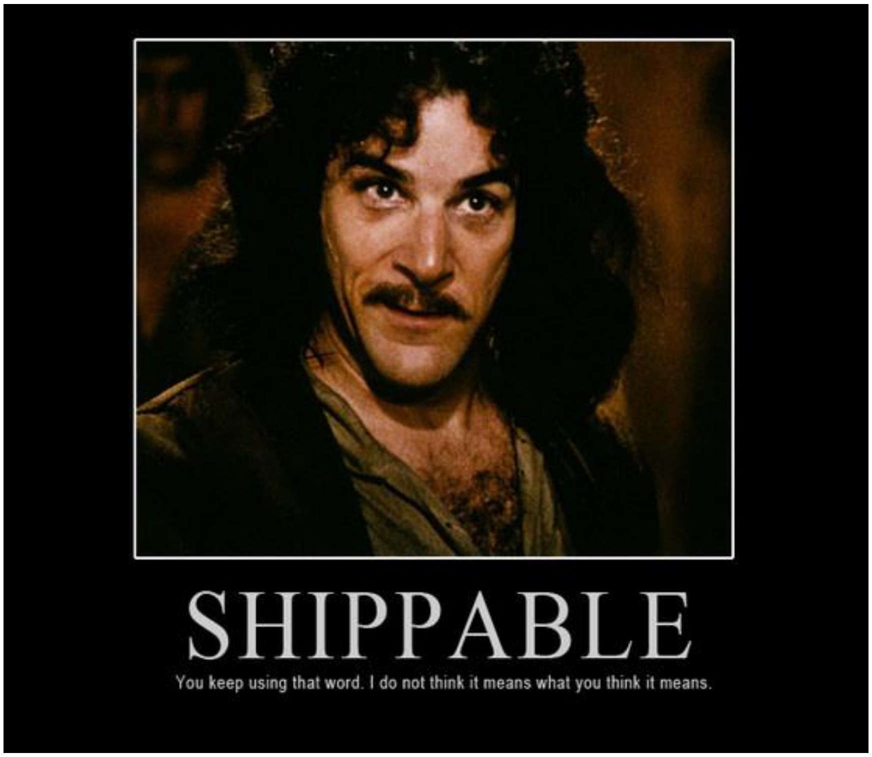
**Potentially shippable product increment**

Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

# How Does Scrum Work?



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.



SHIPPABLE
You keep using that word. I do not think it means what you think it means.

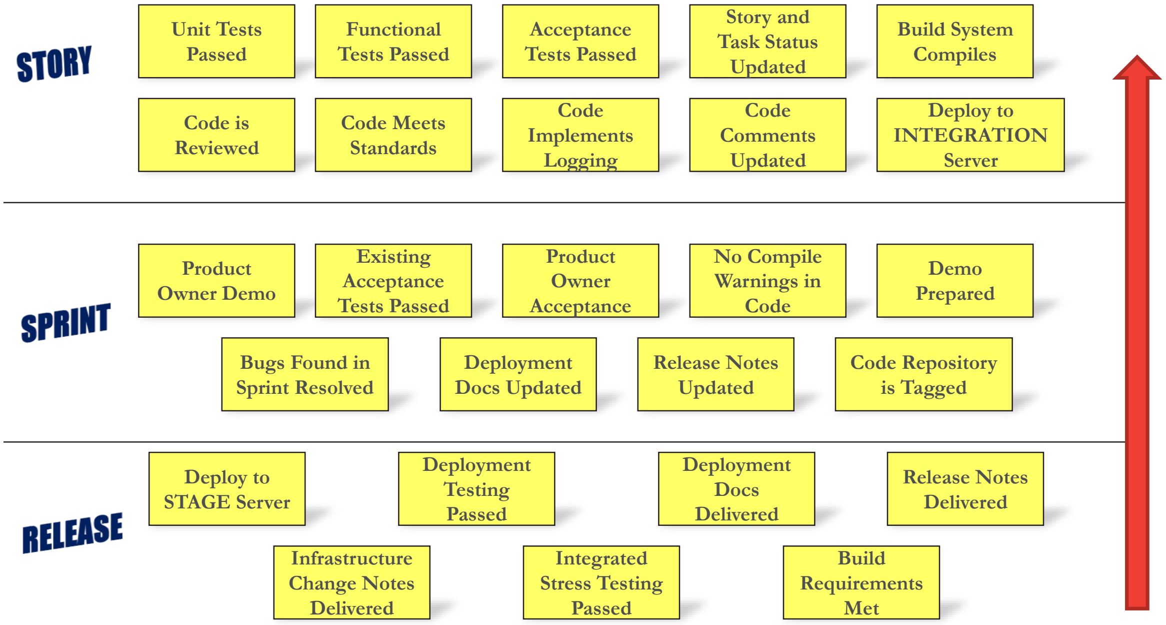
## What is a Potentially Shippable Product Increment?

Working software at production quality

Specified in the "Definition of Done"

Quality standards and common requirements the team holds itself to

## Extending the Definition of Done



STORY

Unit Tests Passed

Functional Tests Passed

Acceptance Tests Passed

Story and Task Status Updated

Build System Compiles

Code is Reviewed

Code Meets Standards

Code Implements Logging

Code Comments Updated

Deploy to INTEGRATION Server

SPRINT

Product Owner Demo

Existing Acceptance Tests Passed

Product Owner Acceptance

No Compile Warnings in Code

Demo Prepared

Bugs Found in Sprint Resolved

Deployment Docs Updated

Release Notes Updated

Code Repository is Tagged

RELEASE

Deploy to STAGE Server

Deployment Testing Passed

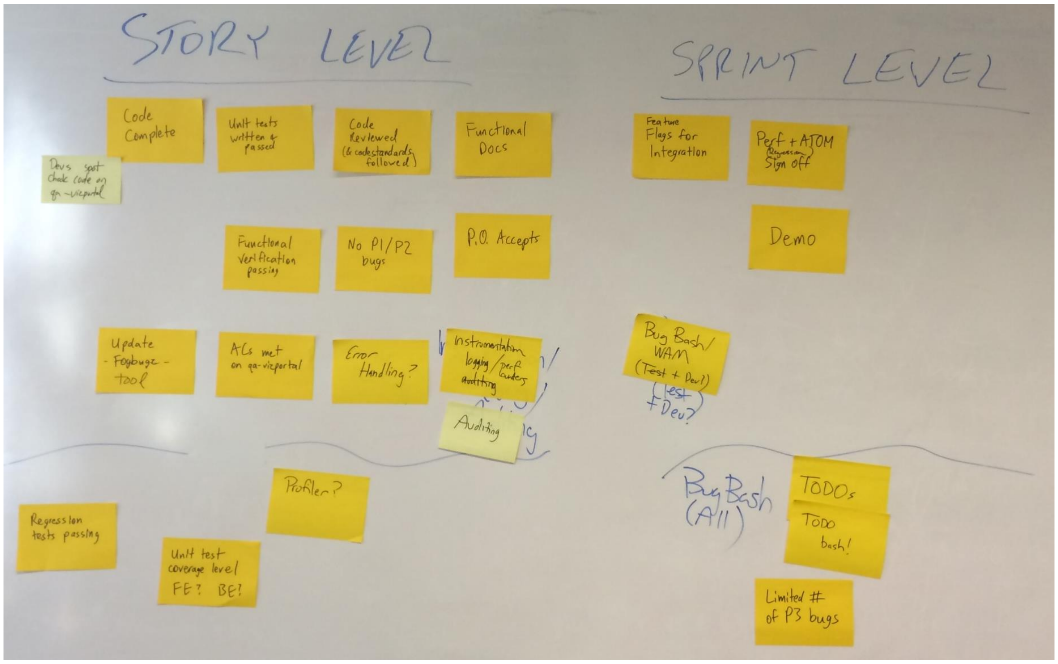
Deployment Docs Delivered

Release Notes Delivered

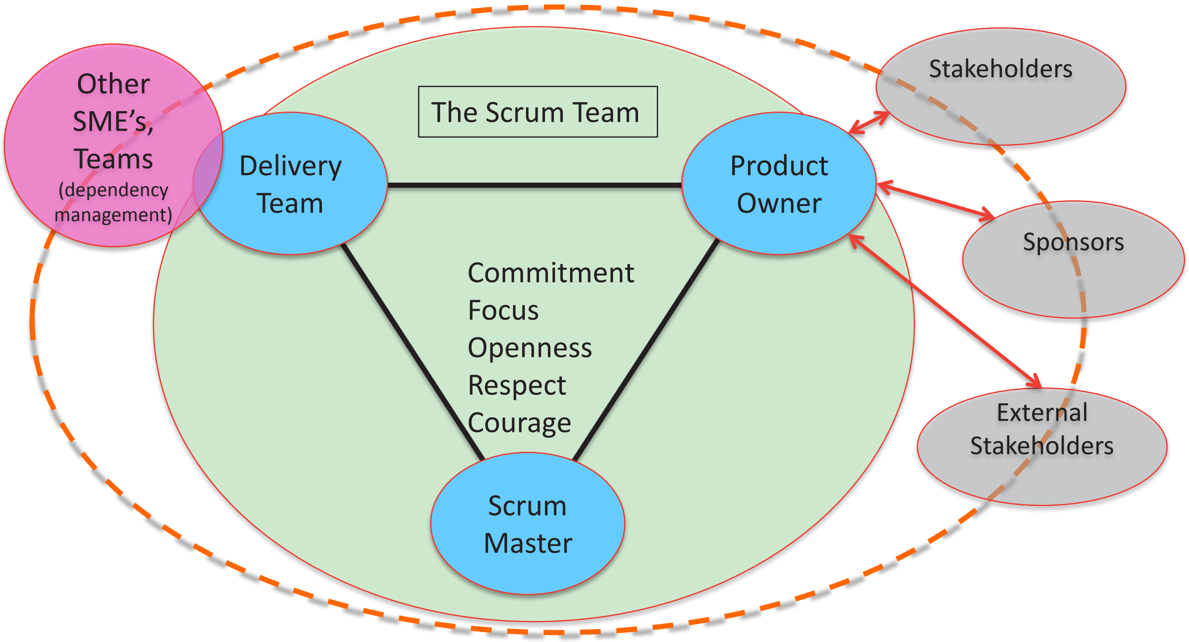
Infrastructure Change Notes Delivered

Integrated Stress Testing Passed

Build Requirements Met


STORY LEVEL
Code Complete
Devs spot check code on qa-vizportal
Unit tests written & passed
Code Reviewed (& codestandards followed)
Functional Docs
Functional Verification passing
No P1/P2 bugs
P.O. Accepts
Update - Fogbugz - tool
ACs met on qa-vizportal
Error Handling?
Instrumentation logging / perf counters auditing
Auditing
Profiler?
Regression tests passing
Unit test coverage level FE? BE?
SPRINT LEVEL
Feature Flags for Integration
Perf + ATOM (Regression) Sign off
Demo
Bug Bash/ WAM (Test + Dev?)
(Test + Dev?)
Bug Bash (All)
TODOs
TODO bash!
Limited # of P3 bugs

## Scrum Roles


Other SME's, Teams (dependency management)
The Scrum Team
Delivery Team
Product Owner
Scrum Master
Commitment
Focus
Openness
Respect
Courage
Stakeholders
Sponsors
External Stakeholders

## Ken Schwaber on Scrum Values

Scrum *is based on*

- ✓Commitment
- ✓Focus
- ✓Openness
- ✓Courage
- ✓Respect

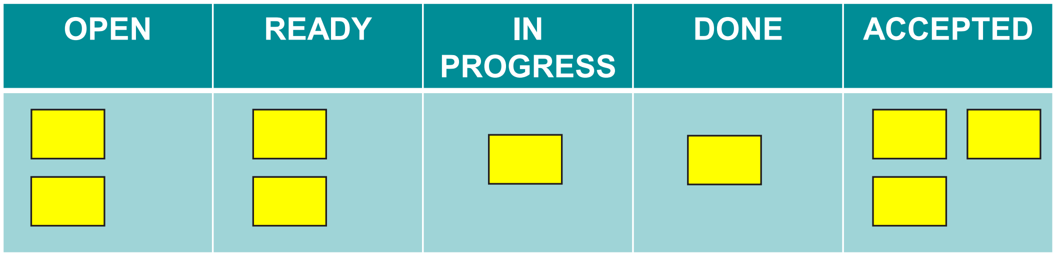
*It asks you to* ***commit*** *to a goal and then provides you with the authority to meet those commitments. Scrum insists that you* ***focus*** *all your efforts on the work you're committed to and ignore anything else.* ***Openness*** *is promoted by the fact that everything about a Scrum project is visible to everyone. Scrum tenets acknowledge that the diversity of team members' background and experience adds value to your project. Finally, Scrum asks you to have the* ***courage*** *to commit, to act, to be open and to expect* ***respect****.*

Source: http://www.ddj.com/dept/architect/184414912?cid=Ambysoft

## Agile Workflow Frameworks

### Kanban

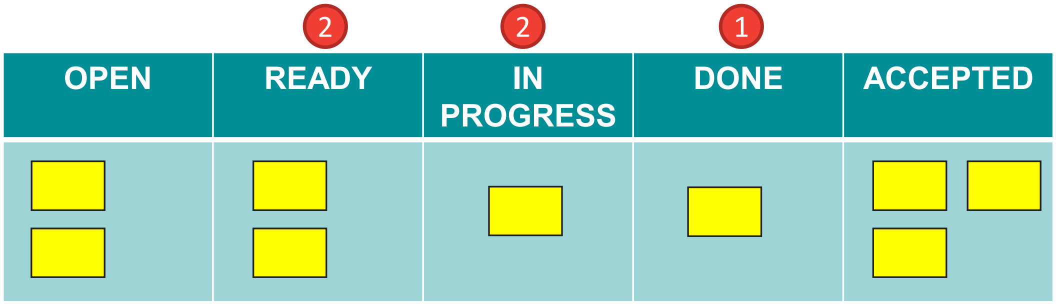
## Kanban Queues



OPEN | READY | IN PROGRESS | DONE | ACCEPTED

- **OPEN:** Item in backlog that has not necessarily been groomed or prioritized and is not ready to be worked by the team.
- **READY:** Item that is groomed and prioritized and eligible to be pulled by the team.
- **IN PROGRESS:** Item that has been pulled and started by the team.
- **DONE:** Item that the team asserts has been finished.
- **ACCEPTED:** Product Owner or other responsible stakeholder has accepted the item as complete.

## Work-In-Process Limits



OPEN | READY | IN PROGRESS | DONE | ACCEPTED

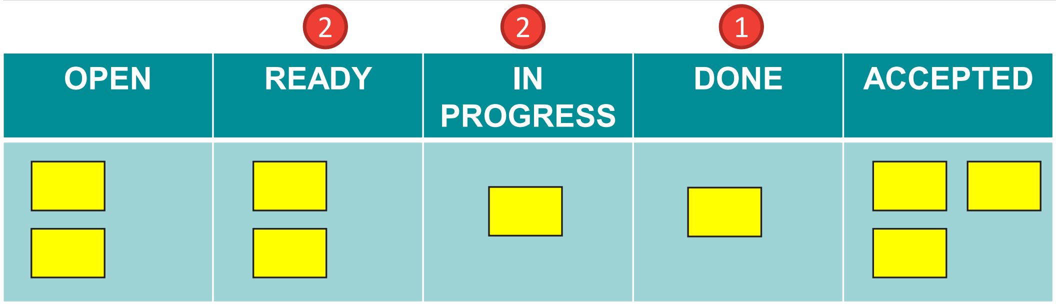
**Work-In-Process (WIP) limits are a discipline to help the team maximize throughput.**

**Don't try to do too many things at once, it only slows you down!**

**When a limit is reached:**

- **No more items can be put into that state (column).**
- **Team "swarms" to help relieve the blockage.**

## Setting WIP Limits



|  | 2 | 2 | 1 |  |
|---|---|---|---|---|
| OPEN | READY | IN PROGRESS | DONE | ACCEPTED |

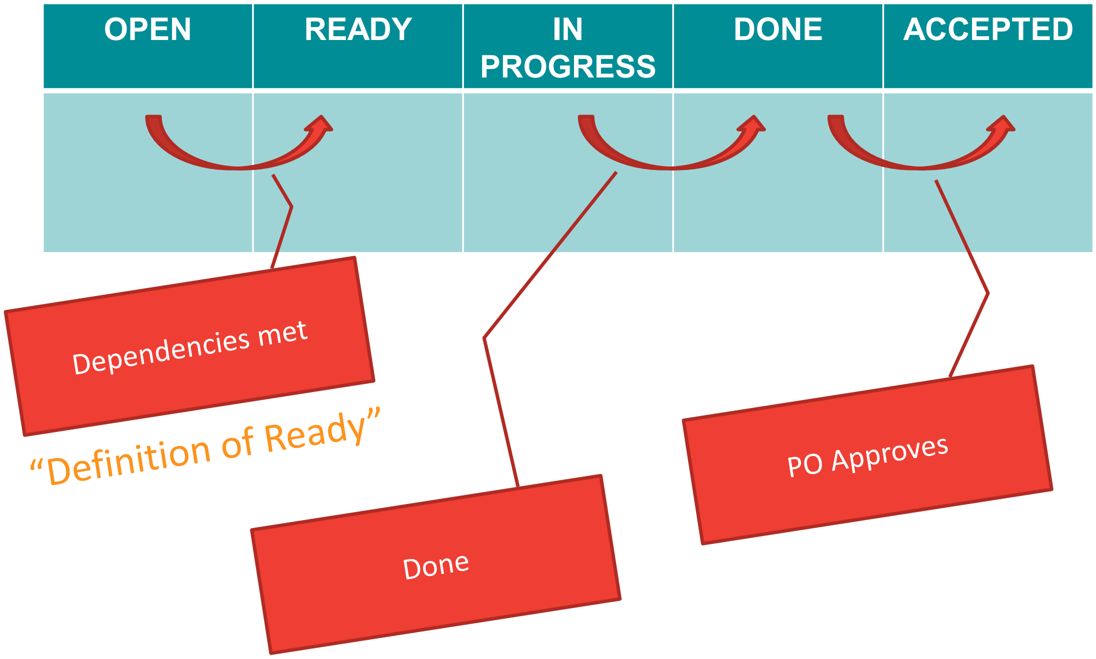
Suggested Initial WIP limits:

- Ready = 1 per Team member
- In Progress = 1 per Team member
- Done = ½ per Team Member

Inspect and adapt at Retrospective

- Idle people? → Limit is too low
- Idle stories? → Limit is too high

# Process Policies

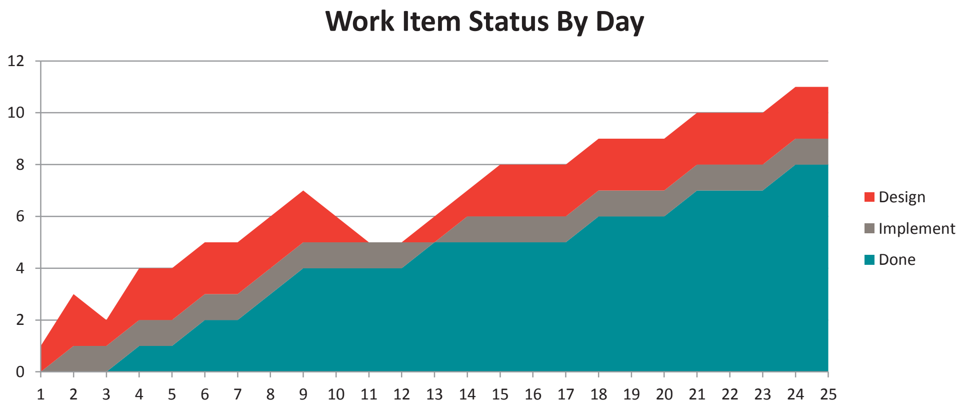


| OPEN | READY | IN PROGRESS | DONE | ACCEPTED |
|---|---|---|---|---|

# Classes of Service

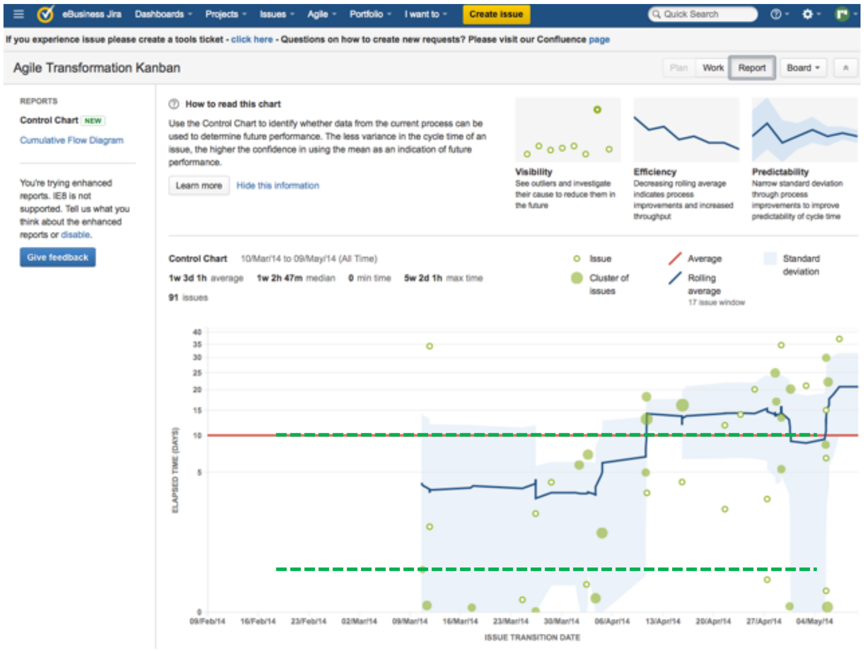
| OPEN | READY | IN PROGRESS | DONE | ACCEPTED |
|---|---|---|---|---|
| | Emergent | | | |
| | Fixed Date | | | |
| | Regular | | | |

# Cumulative Flow Diagram



SolutionsIQ

# Control Chart



If you experience issue please create a tools ticket - click here - Questions on how to create new requests? Please visit our Confluence page

Agile Transformation Kanban

You're trying enhanced reports. IE8 is not supported. Tell us what you think about the enhanced reports or disable.

How to read this chart

Use the Control Chart to identify whether data from the current process can be used to determine future performance. The less variance in the cycle time of an issue, the higher the confidence in using the mean as an indication of future performance.

**Visibility**
See outliers and investigate their cause to reduce them in the future

**Efficiency**
Decreasing rolling average indicates process improvements and increased throughput

**Predictability**
Narrow standard deviation through process improvements to improve predictability of cycle time

**Control Chart** 10/Mar/14 to 09/May/14 (All Time)

**1w 3d 1h** average **1w 2h 47m** median **0** min time **5w 2d 1h** max time

**91** issues

SolutionsIQ

# Agile Workflow and Quality

## Two Kinds of Doing

**Doing the right thing**

**Doing the thing right**

## Example Dimensions of Software Quality

- Accessibility
- Compatibility
- Concurrency
- Efficiency
- Functionality
- Installability
- Localizability
- Maintainability
- Performance
- Portability
- Reliability
- Scalability
- Security
- Testability
- Usability
- Timeliness [-HD]

» Source: SoftwareTestingFundamentals.com

## How Agile Workflow Promotes Quality

- Small, well-defined units of work
- Incremental and iterative development
- Focus on delivering working software
- Focus on quality standards
- Focus on bringing testing early into development
- Continuous stakeholder participation
- Frequent product feedback
- Get ROI on-stream earlier, stop when it makes sense
- YAGNI, 80/20
- Kaizen

# Scrum + Kanban

## How to Choose?

SolutionsIQ

## Scrum is Good When…

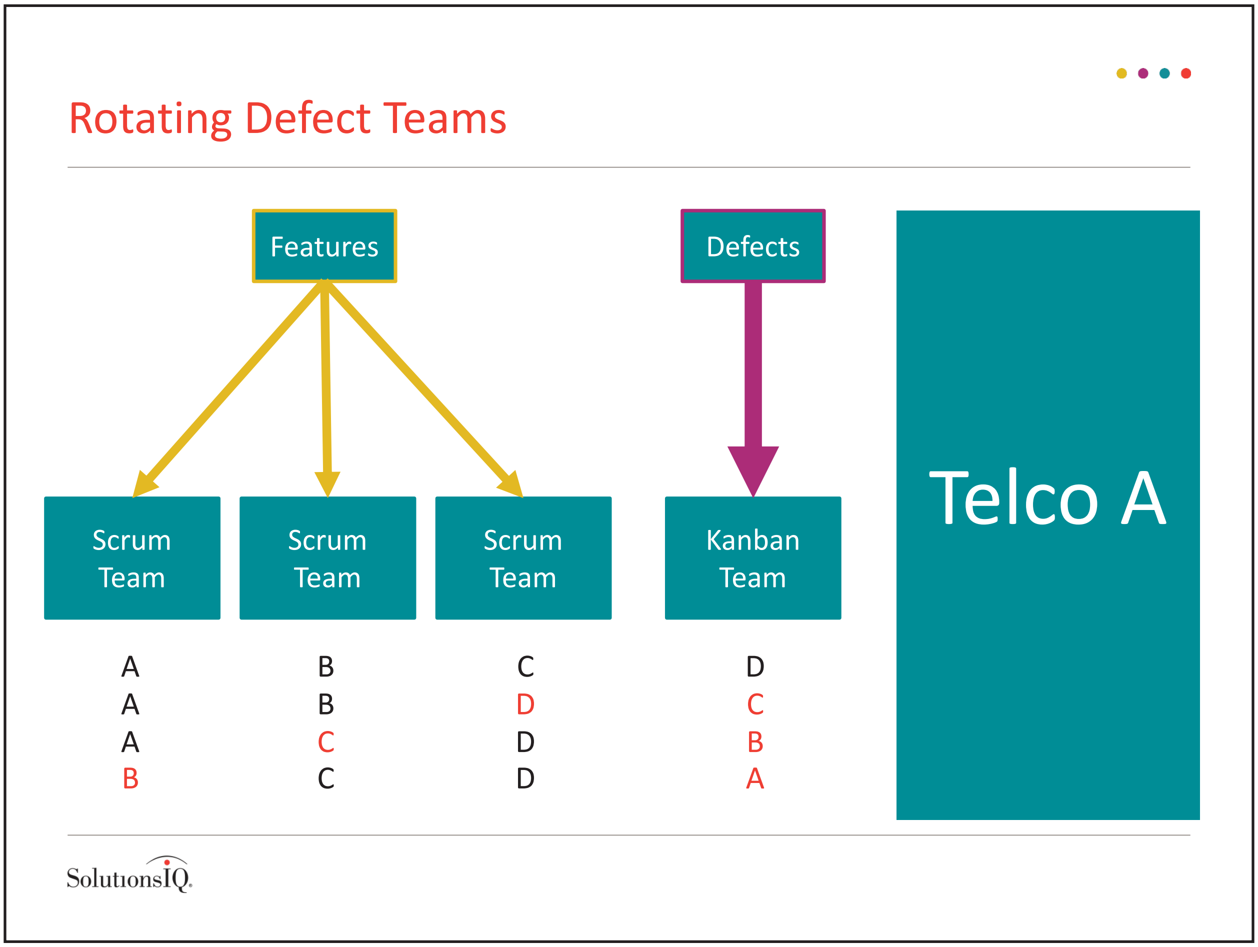
- A sprint cadence provides helpful discipline
- Story size is not too big
- Work items are practical to estimate
- The organization is ready for a significant step

SolutionsIQ

## Kanban is Good When…

- There is a continuous flow of work item input
- Work items tend to be:
  - Smaller and/or repetitive
  - Bigger than a sprint and hard to partition
  - Emergent, date-driven, or expected quickly rather than at sprint boundaries
  - Subject to rapidly shifting priorities
  - Difficult to estimate (e.g., from a defect report)
- Organization change needs to be taken in small increments

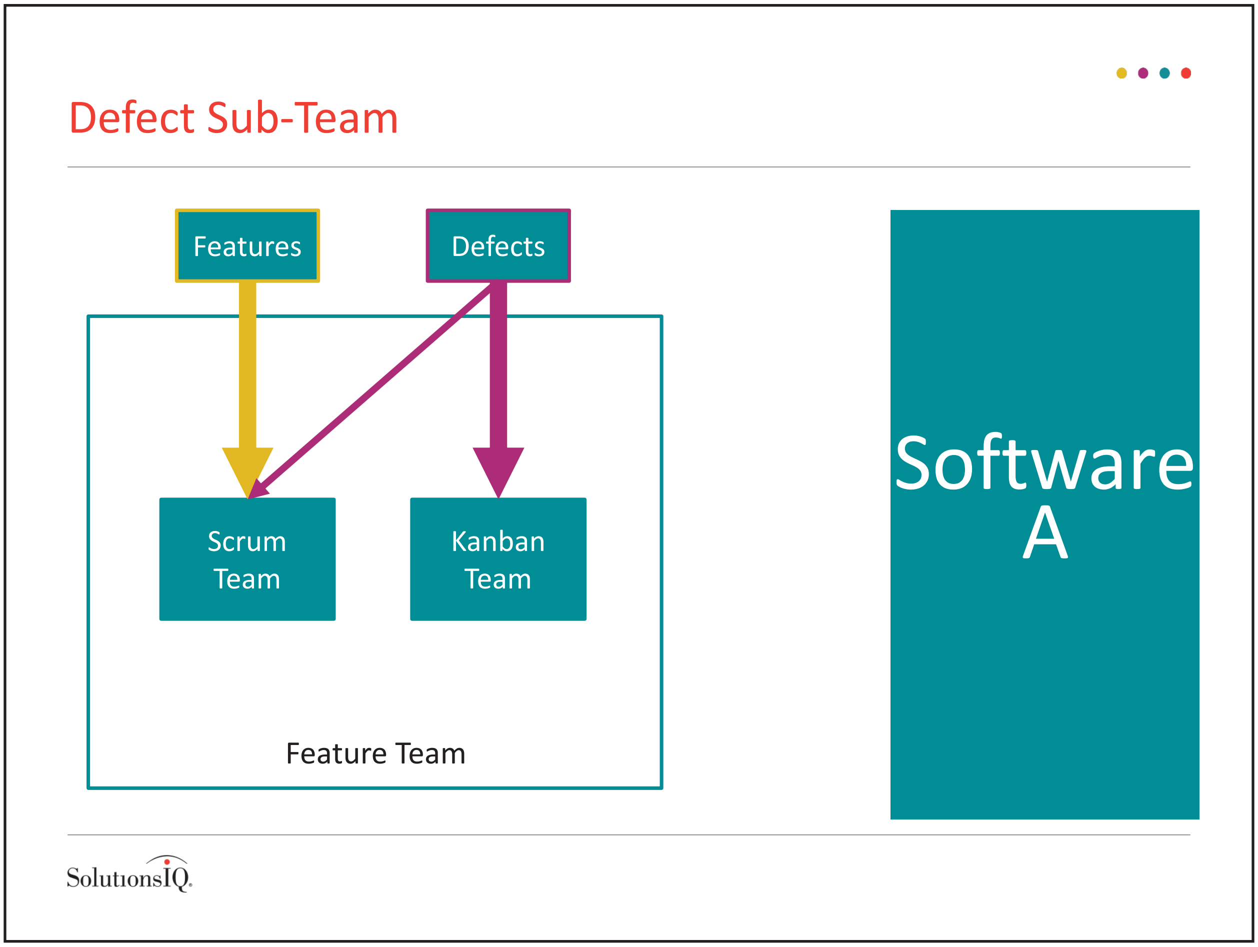
## Mixed and Blended Designs



## Rotating Defect Teams

| Scrum Team | Scrum Team | Scrum Team | Kanban Team |
|---|---|---|---|
| A | B | C | D |
| A | B | D | C |
| A | C | D | B |
| B | C | D | A |

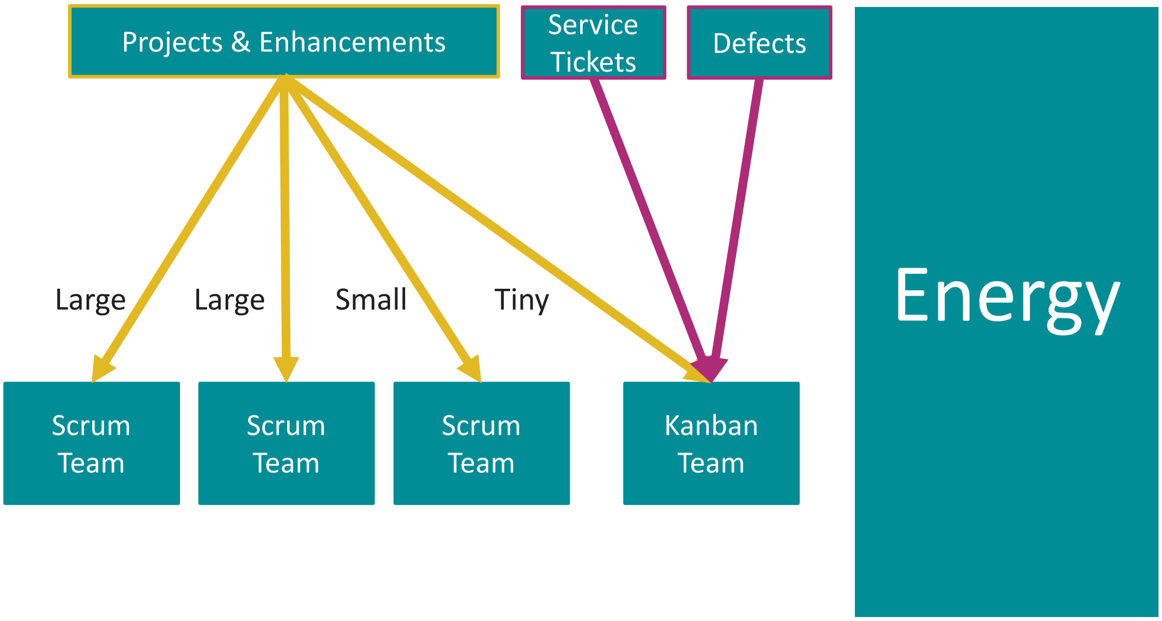
Telco A



## Defect Sub-Team

Software A

## Large and Small Items



Projects & Enhancements

Service Tickets

Defects

Large

Large

Small

Tiny

Scrum Team

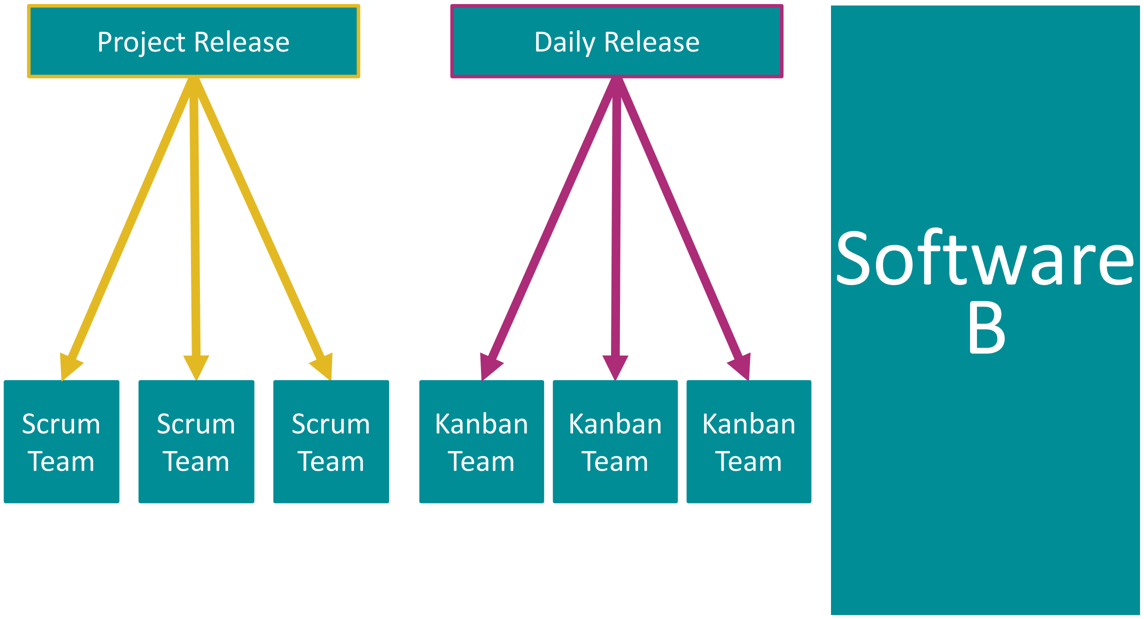
Scrum Team

Scrum Team

Kanban Team

Energy

## Project and Daily Release



Project Release

Daily Release

Scrum Team

Scrum Team

Scrum Team

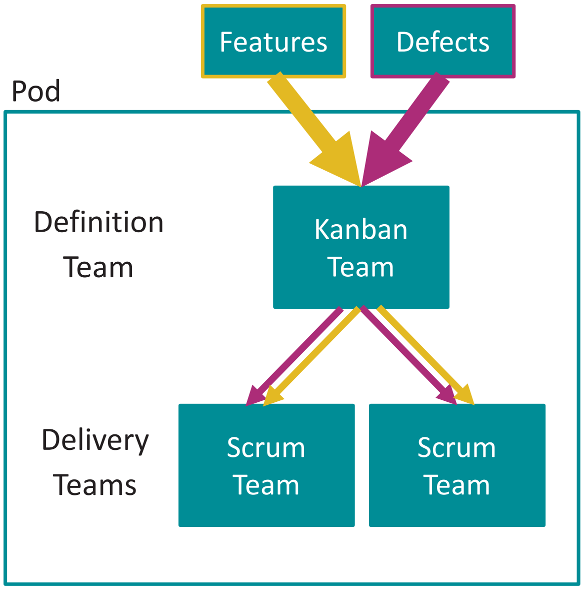
Kanban Team

Kanban Team

Kanban Team

Software B

## Definition and Delivery Teams



Features

Defects

Pod

Definition Team

Kanban Team

Delivery Teams

Scrum Team

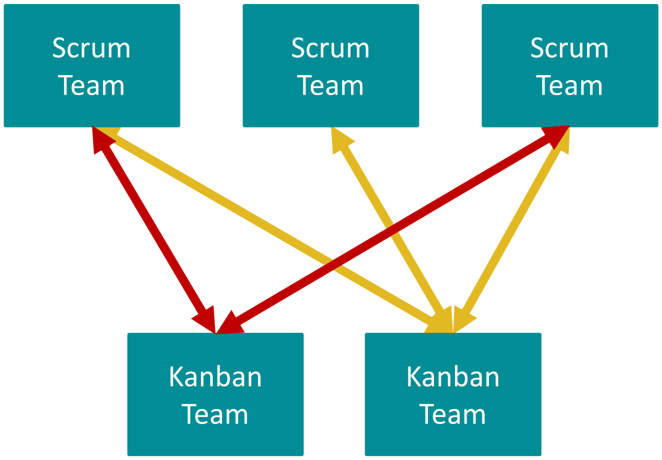
Scrum Team

Flow to Ready

Iterate to Done

Telco B

## Specialty Service Queues



Scrum Team

Scrum Team

Scrum Team

Kanban Team

Kanban Team

Fantasy Football

# Questions

# Thank You!

# Mobile UX Make or Break

## Philip Lew, XBOSoft

With Mobile apps and subscription business models, if users can't figure your application out in one minute (for a mobileapp, maybe 30 seconds), they're gone. Usability and UX are key factors in satisfying users so understanding and measuring them is critical to success and rollout for today's mobile applications. But Usability and User Experience (UX) can sometimes be confusing, not only in differentiating them, but in understanding them. So, first, this talk explores the meanings of the two and how they are related, and then examines their importance for today's mobile applications. This talk will define usability and user experience measurements and crystalize abstract concepts like "wow factor" so that they can be measured, with metrics to evaluate and improve your product. Numerous examples are presented to demonstrate the concepts and how to improve your mobile app.

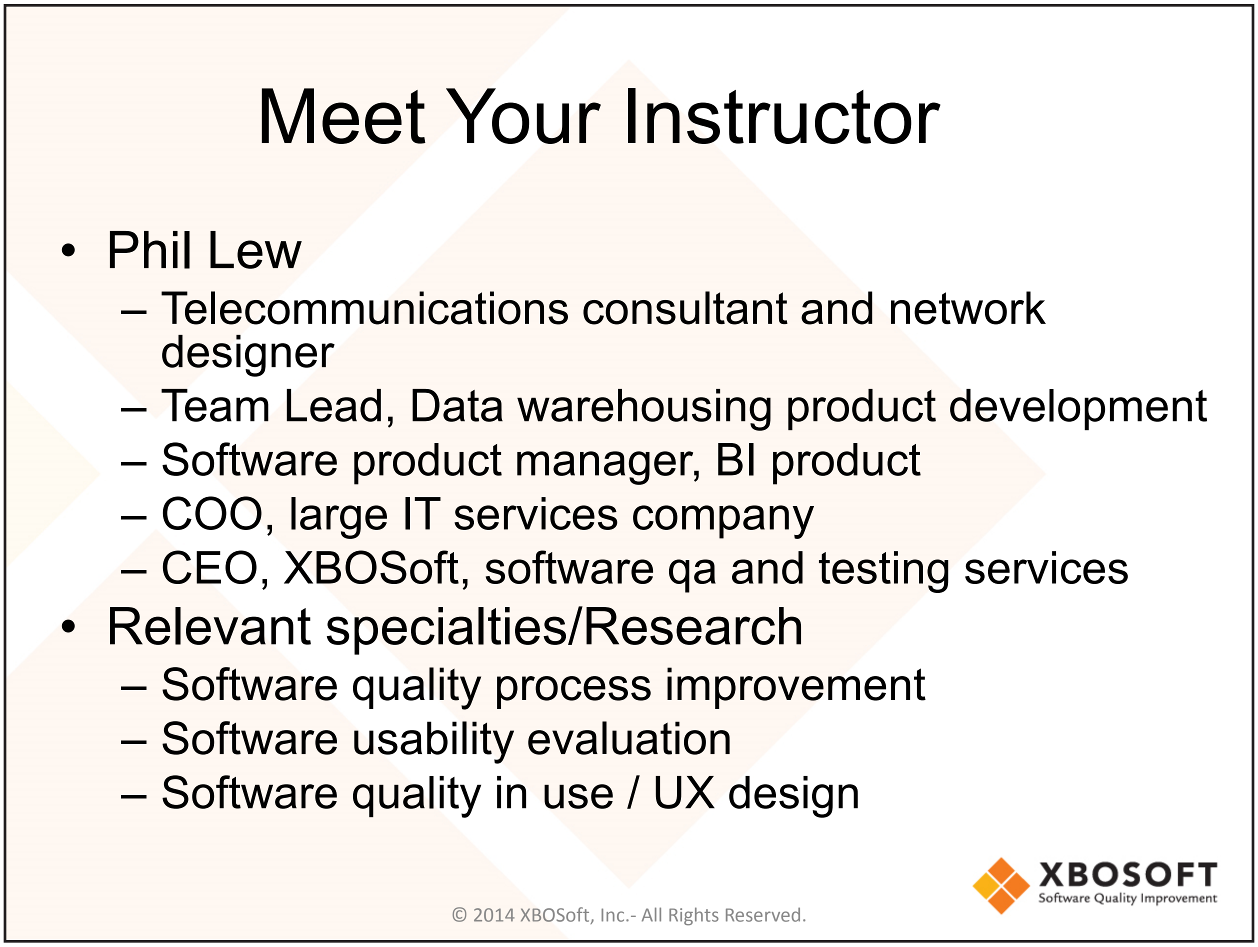
***Philip Lew**, CEO of XBOSoft, has both founded companies and managed them on behalf of others during his quarter-century-plus professional career. Although he has focused on software quality since joining XBOSoft in 2006 and has deep technical expertise as a software engineer, his experience is broad and eclectic. He's led systems integration and new product development initiatives, prepared companies to go public, and served as an advisor on technology and business processes. Prior to joining XBOSoft, he served as an Ernst & Young consultant; founded and ran Pulse Technologies Inc. until it was purchased by EIS International; led the Systems Integrations Services Group at EIS; and served in a variety of executive roles for technology firms both in the U.S. and abroad. He holds a B.S. and a Masters Degree in operations research and engineering from Cornell University, and a Ph.D. in computer science and engineering from Beihang University. He's a fanatical bicyclist, doesn't own a car, loves social media and has visited more than 60 countries in an attempt to satisfy his travel lust.*

# Mobile UX Make or Break

Usability is Only the Beginning

Philip Lew

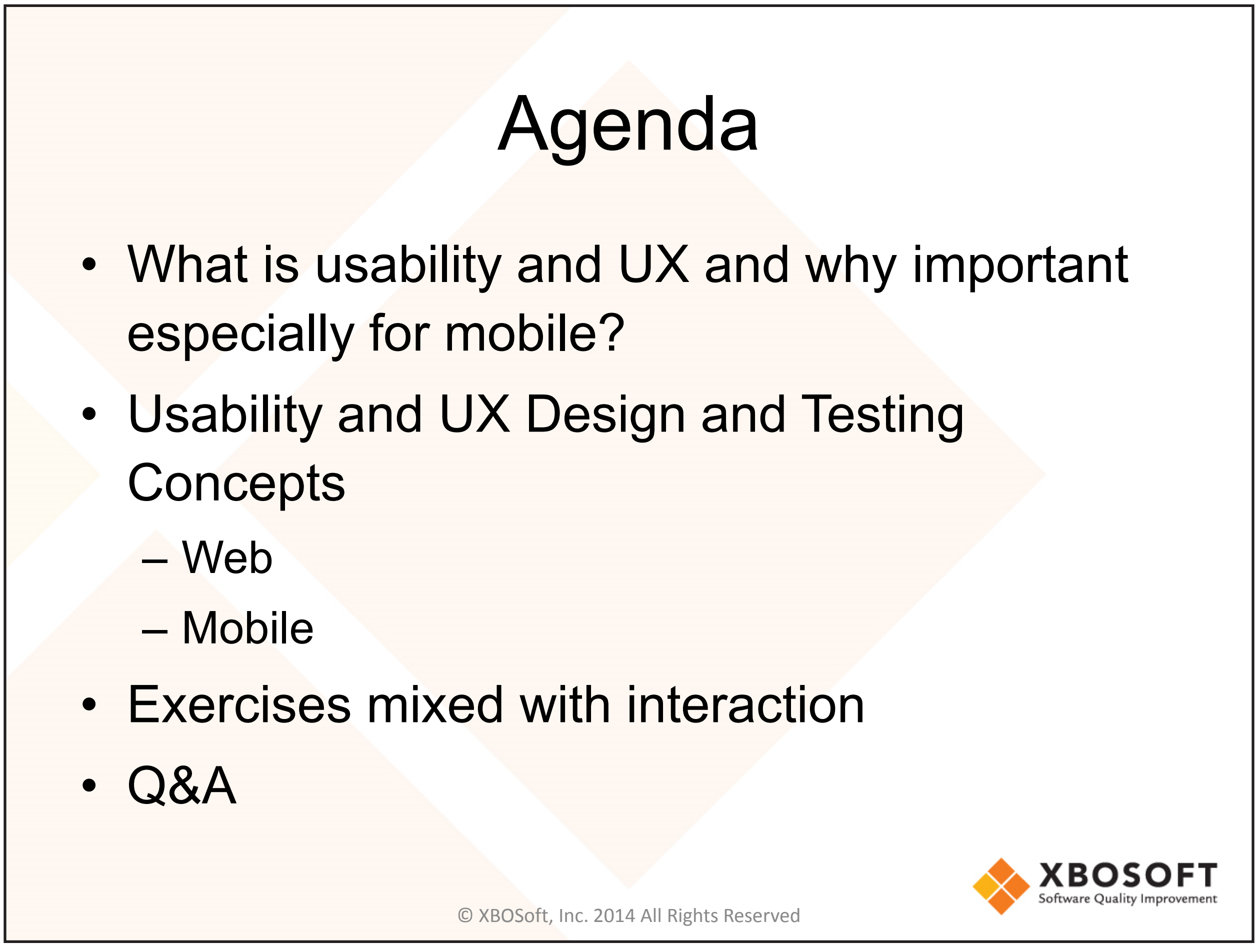
# "That Is The Question!"



# Meet Your Instructor

- Phil Lew
  - Telecommunications consultant and network designer
  - Team Lead, Data warehousing product development
  - Software product manager, BI product
  - COO, large IT services company
  - CEO, XBOSoft, software qa and testing services
- Relevant specialties/Research
  - Software quality process improvement
  - Software usability evaluation
  - Software quality in use / UX design

XBOSOFT Software Quality Improvement



# Agenda

- What is usability and UX and why important especially for mobile?
- Usability and UX Design and Testing Concepts
  - Web
  - Mobile
- Exercises mixed with interaction
- Q&A

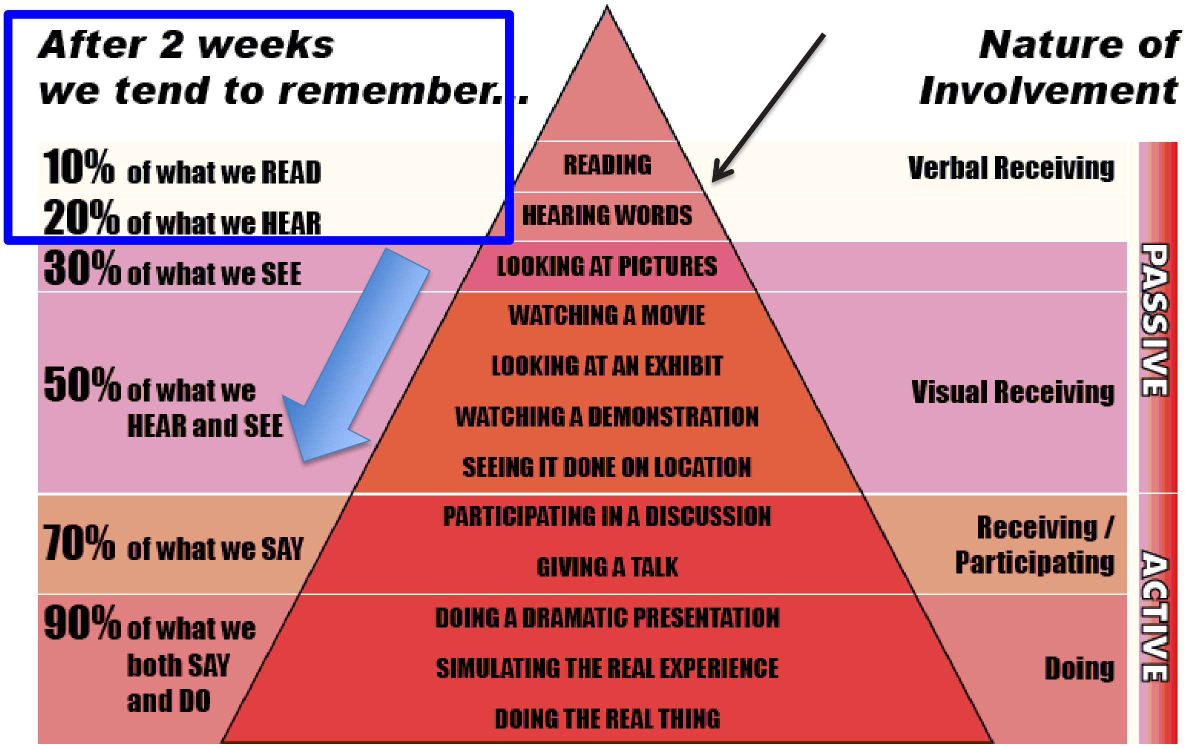
XBOSOFT Software Quality Improvement

# Workshop/Session Spirit

- Interactive
- I won’t read the slides…
- Slides for you as a take-away (lots)
- Ask questions… OR I will !!!

XBOSOFT
Software Quality Improvement

# Cone of Learning (Edgar Dale)



***After 2 weeks we tend to remember…***

**Nature of Involvement**

**10%** of what we READ — READING — Verbal Receiving

**20%** of what we HEAR — HEARING WORDS

**30%** of what we SEE — LOOKING AT PICTURES

**50%** of what we HEAR and SEE — WATCHING A MOVIE; LOOKING AT AN EXHIBIT; WATCHING A DEMONSTRATION; SEEING IT DONE ON LOCATION — Visual Receiving

**70%** of what we SAY — PARTICIPATING IN A DISCUSSION; GIVING A TALK — Receiving / Participating

**90%** of what we both SAY and DO — DOING A DRAMATIC PRESENTATION; SIMULATING THE REAL EXPERIENCE; DOING THE REAL THING — Doing

PASSIVE

ACTIVE

Edgar Dale, *Audio-Visual Methods in Technology*, Holt, Rinehart and Winston.

# Usability-UX and Its Importance

## Mobile is King

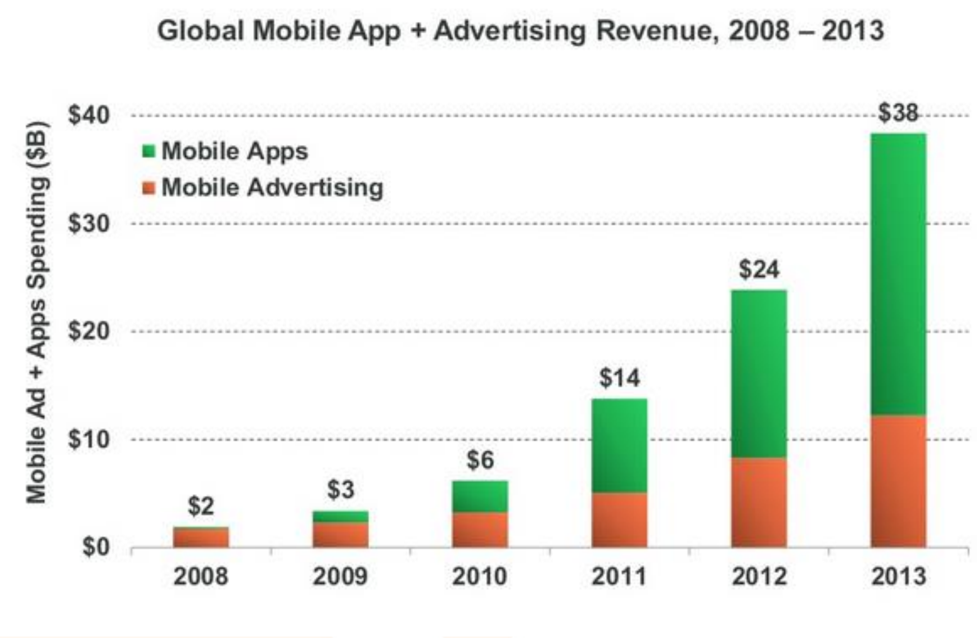
- 91% of American adults own a mobile phone.
- Over 60% of mobile phones sold are smartphones.
- Mobile device web access is predicted to overtake desktop web access in 2014.
- 48% use or would like to use a smartphone to shop while in-store or on the go.
- 90% of people start a task on one device, then complete it on another.

# Mobile UX – Up Up and Away

- UX is relative to user expectations
  - Web users' expectations used to be lower when using a mobile device versus using their desktop computer. As more sites deliver better mobile UX the bar is higher.
  - Mobile users no longer expect a lesser experience, expect an equal or mobile-centric experience.
  - Mobile UX is no longer nice to have, but a critical component of project success.

XBOSOFT
Software Quality Improvement

# MobileApps Are and Will be Dominant Revenue Source



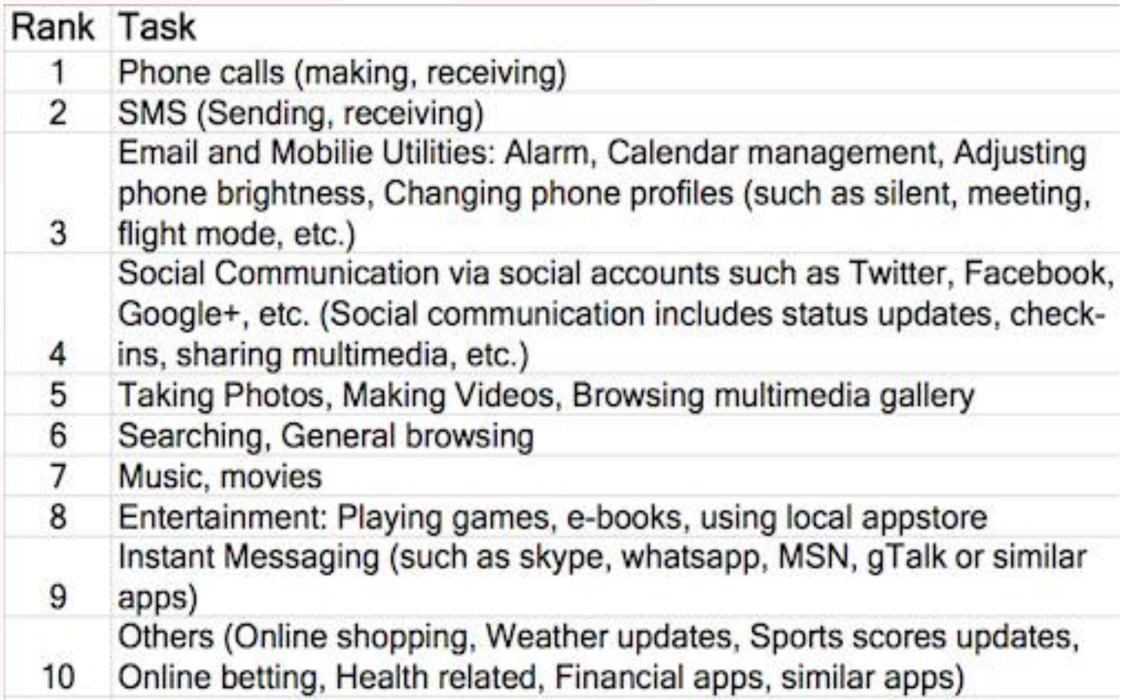
XBOSOFT
Software Quality Improvement

# Mobile User Expectations

- Subscription Business Model
  - Instead of paying upfront and ‘owning’ the software
  - Pay as you go
- Cloud and mobile convergence
- Behavior and expectations have changed

61% of Verizon users now have smartphones

# What Users Do With Their Mobile



| Rank | Task |
|---|---|
| 1 | Phone calls (making, receiving) |
| 2 | SMS (Sending, receiving) |
| 3 | Email and Mobilie Utilities: Alarm, Calendar management, Adjusting phone brightness, Changing phone profiles (such as silent, meeting, flight mode, etc.) |
| 4 | Social Communication via social accounts such as Twitter, Facebook, Google+, etc. (Social communication includes status updates, check-ins, sharing multimedia, etc.) |
| 5 | Taking Photos, Making Videos, Browsing multimedia gallery |
| 6 | Searching, General browsing |
| 7 | Music, movies |
| 8 | Entertainment: Playing games, e-books, using local appstore |
| 9 | Instant Messaging (such as skype, whatsapp, MSN, gTalk or similar apps) |
| 10 | Others (Online shopping, Weather updates, Sports scores updates, Online betting, Health related, Financial apps, similar apps) |

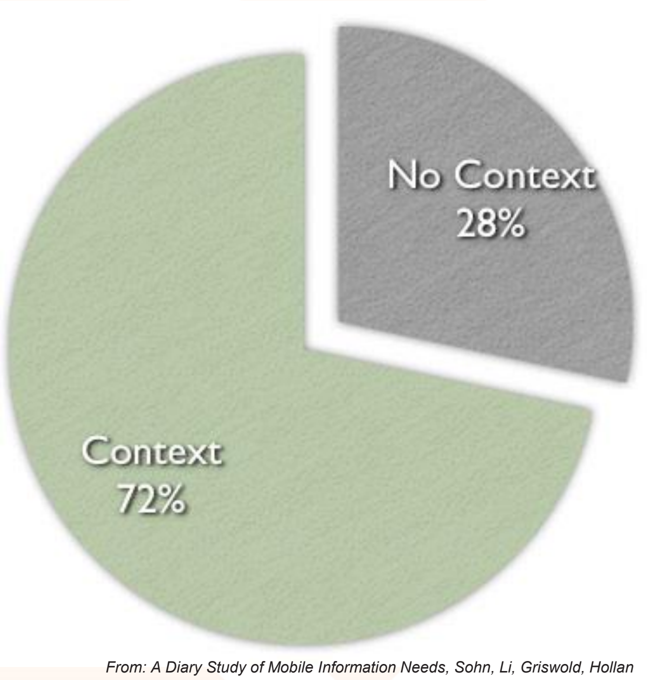
http://marketingland.com/smartphone-activities-study-email-web-facebook-[illegible]

# Mobile now means more…

- Not just smart phones…
- Poll
  - How many of you have a wearable computing device?
  - How many have more than one?

# Mobile Tasks

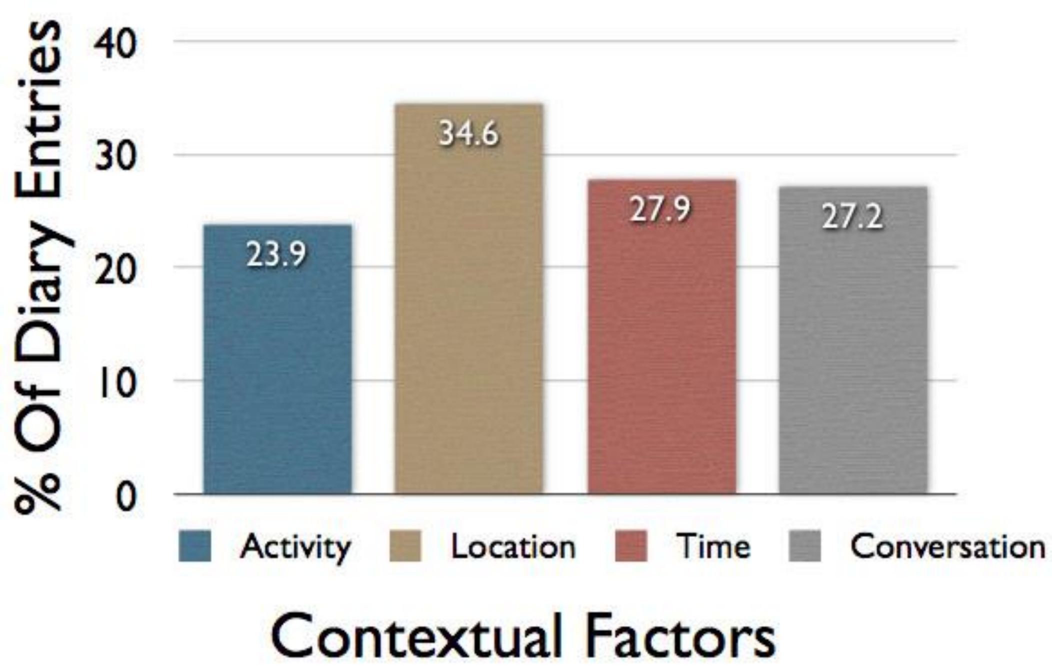
Importance of User Context For Mobile Apps



**What is context?**

*From: A Diary Study of Mobile Information Needs, Sohn, Li, Griswold, Hollan*

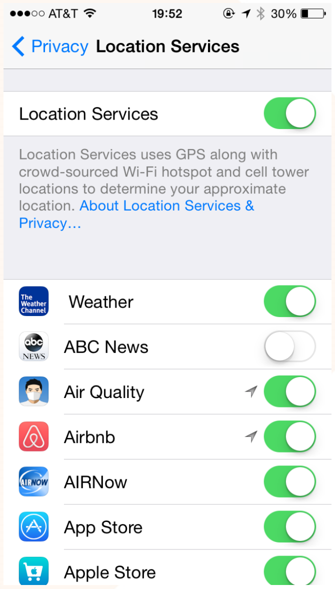
# Context of the User



*From: A Diary Study of Mobile Information Needs, Sohn, Li, Griswold, Hollan*

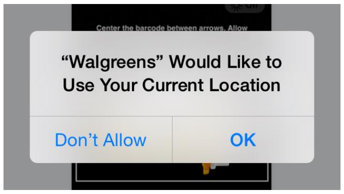
# Mobile as a Context Funnel

- Importance of and Use of Context is Changing
- Context aggregator
  - Social media
  - Data
  - Location
  - Sensors



Location Services

Location Services uses GPS along with crowd-sourced Wi-Fi hotspot and cell tower locations to determine your approximate location. About Location Services & Privacy…



"Walgreens" Would Like to Use Your Current Location

# What's the Big Deal About Big Data?

- What is critical about all this data?
  - Social media
  - Location
    - Sensors

- Timeliness
- Accuracy
- Reliability
- What else?

**How will we test all this?**

# Anticipation

- With context you can anticipate/predict what your clients/customers want.
- Does not just mean selling them stuff…

UX in the future will be dependent on providing anticipatory services without being freaky and without destroying trust.

# Mobile Usability Challenges

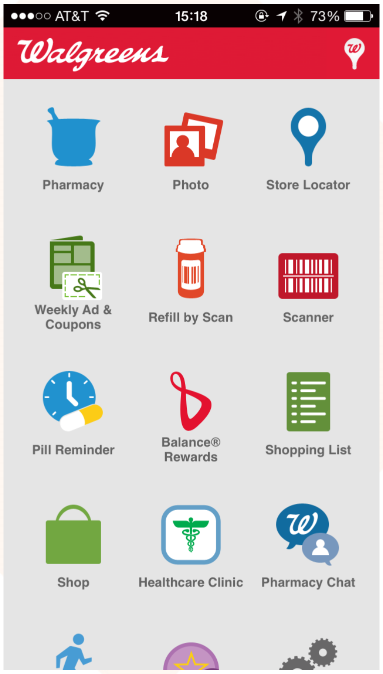
- Limited ***attention*** – people often multi-tasking
- User needs triggered by context
  - Application needs to provide what they want at the “right” time and in a form suitable for current context.
- Require access to personal data, obtained either through web-based services or their personal devices-tight integration
- ***Require sensitivity to the task at hand***

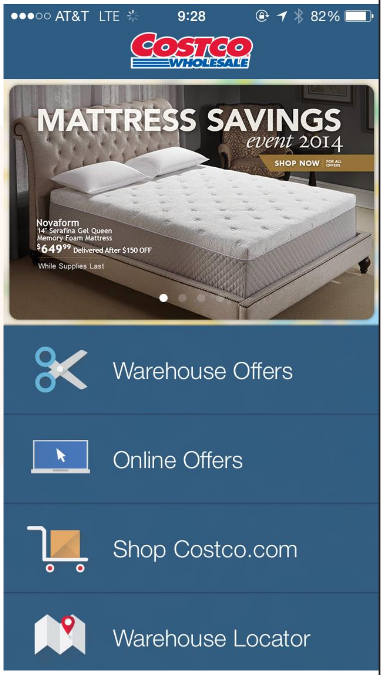
# Mobile Task Considerations

- **Small screen and slow interaction**
  - slow download speed and/or small keypads
- What functionalities of your full app need to be transferred to mobile?
  - More items >> less attention for each of them
  - What order >> what is on top versus bottom?
- ***Tasks on mobile should require just 2-3 clicks***
  - Each click is an opportunity for a dropped connection or slow downloading user experience.

# Task Prioritization

- What order of importance is placed on different tasks?
- How many things can one do? Versus the webapp?





# Tasks Suited For Mobile

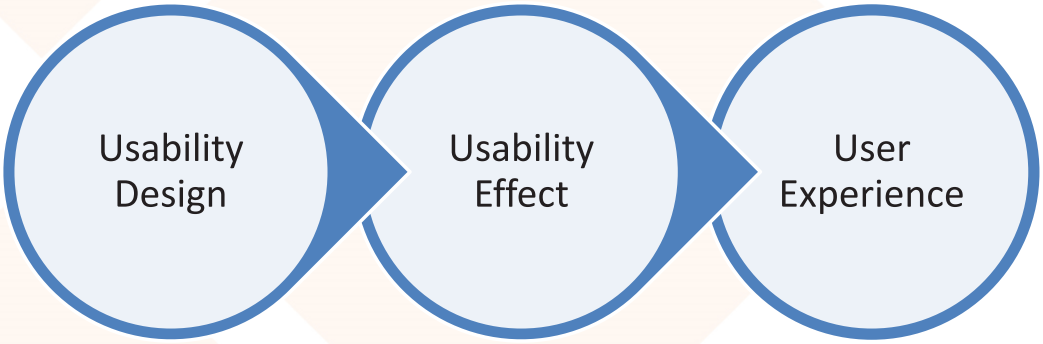
- **Tasks that have a deadline**
  - Buy a gift at the last minute
  - Pay bills during vacation
  - Check bank balances before writing a check
- **Tasks that involve rapidly changing information**
  - Traffic, flight info, movie schedules, directions
- **Tasks that require privacy**
  - Small screens ideal for private activities
  - Check personal email or doing non-work related tasks
  - Social networking

# Tasks Not Suited For Mobile

- Involve a large amount of complexity and/or very time consuming
  – Research
  – Large amounts of reading
  – Comparison of many options
  – Advanced transactions

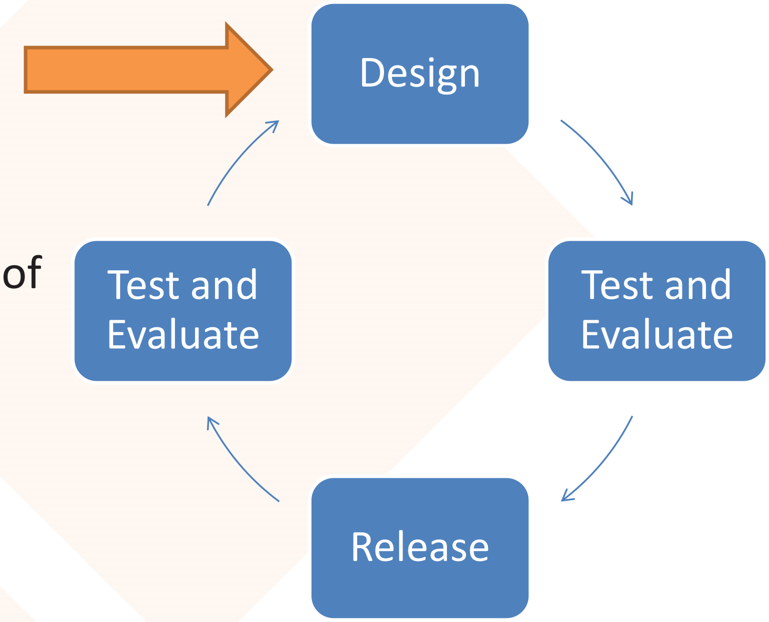
What is your mobile app task and is it suited for mobile?

# Basic Usability Concepts



# Design-Test and Evaluate

- What will the mobileapp do?
  - Big impact on UX
- Is it a conversion of existing app?
- What functions will a user really access?



XBOSOFT
Software Quality Improvement

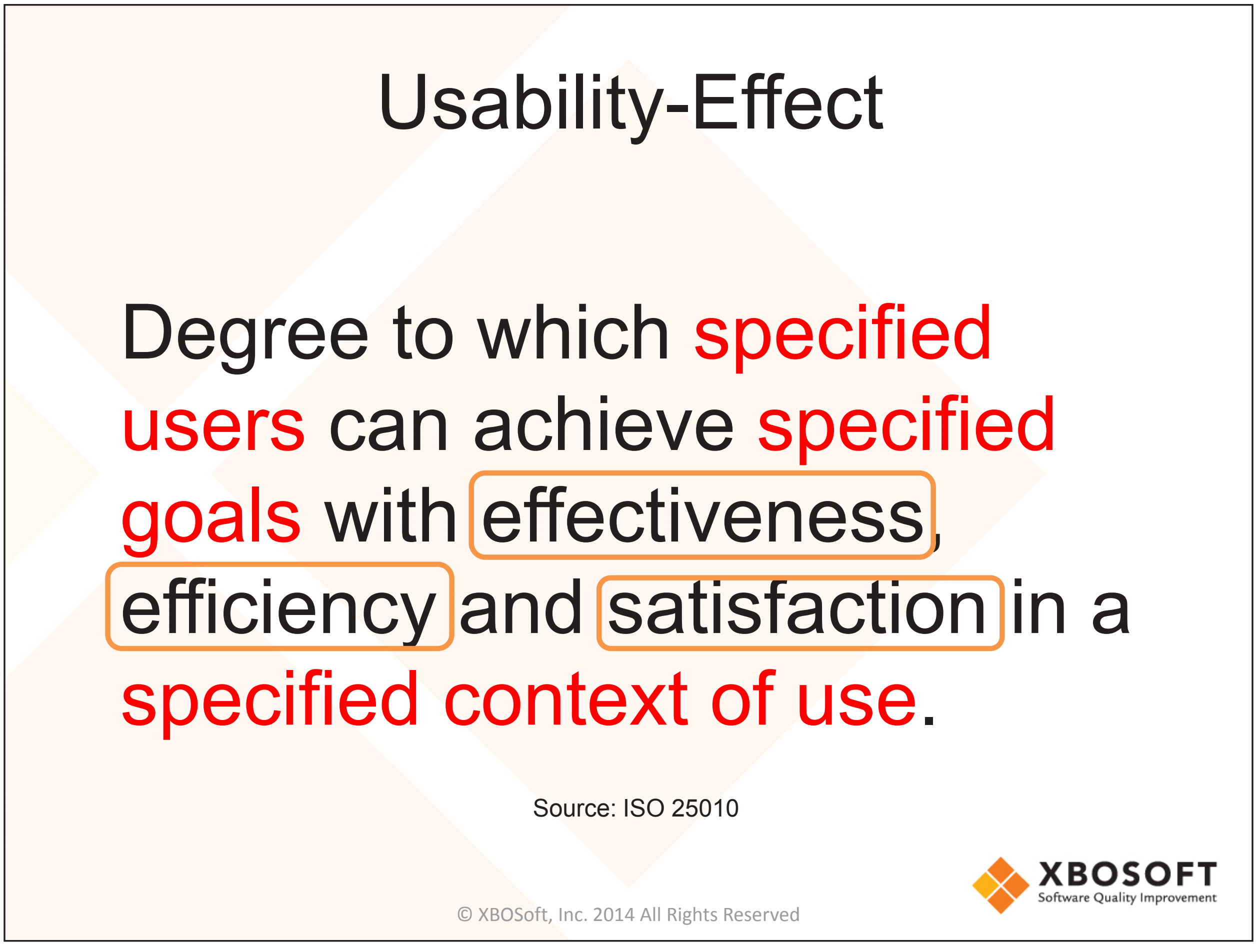
# Usability - Design Perspective

- Understandability
- Learn-ability
- Operability
- Attractiveness
- Navigation
- Responsiveness-performance

*If the user cannot figure it out in 30 seconds, they are gone.*

What else can you think of?

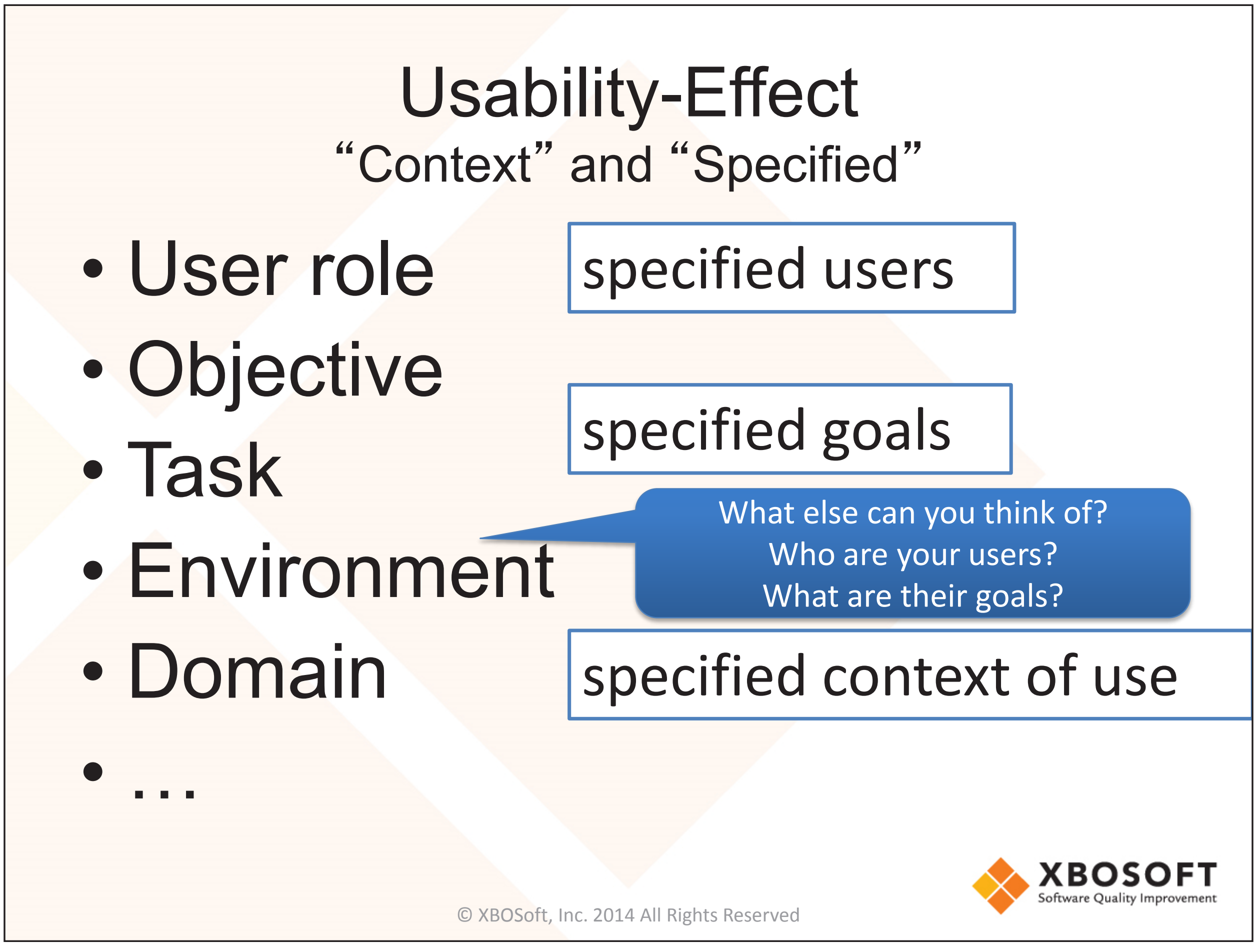
XBOSOFT
Software Quality Improvement



# Usability-Effect

Degree to which specified users can achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

Source: ISO 25010



# Usability-Effect

## “Context” and “Specified”

- User role
- Objective
- Task
- Environment
- Domain
- …

specified users

specified goals

specified context of use

What else can you think of?
Who are your users?
What are their goals?

# Design and Test For the Task

Prioritize tasks - *Majority of app's value is provided by a small number of tasks.*

- **Highly specific**
  - – You want to sell 20 shares of stock for a security you know you own.
- **Directed**, but less specific.
  - – Find the stocks in my account/portfolio.
- **Open-ended**, but restricted to a predetermined site or app.
  - – See where the stock market is today

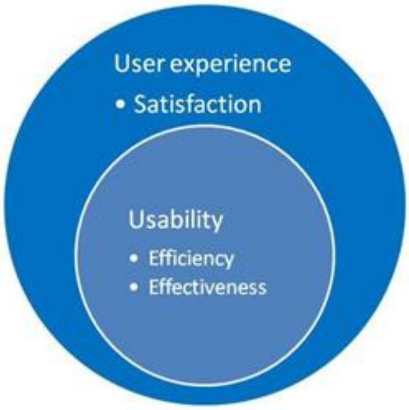
**If you are not involved in the design…. ASK**

# Exercise

1. Work in groups of 2-3
2. Examine one of your applications
   - –Or a common application
3. Determine the objective of the application
4. Develop a list of 3-5 tasks
5. Prioritize the tasks

# Usability-Effect User Experience

- **Satisfaction:** Degree to which users are satisfied in a specified context of use.
  - Likability (cognitive satisfaction)
  - Pleasure (emotional satisfaction)
  - Comfort (physical satisfaction)
  - Trust



Including many other factors experienced over time and other integrated channels and platforms

Convenience
Anticipation

Source: ISO 25010

© XBOSoft, Inc. 2014 All Rights Reserved

# User Experience and Trust

User experience is not just about UI efficiency, it's also about trust and relationship.

© 2014 XBOSoft, Inc.- All Rights Reserved.

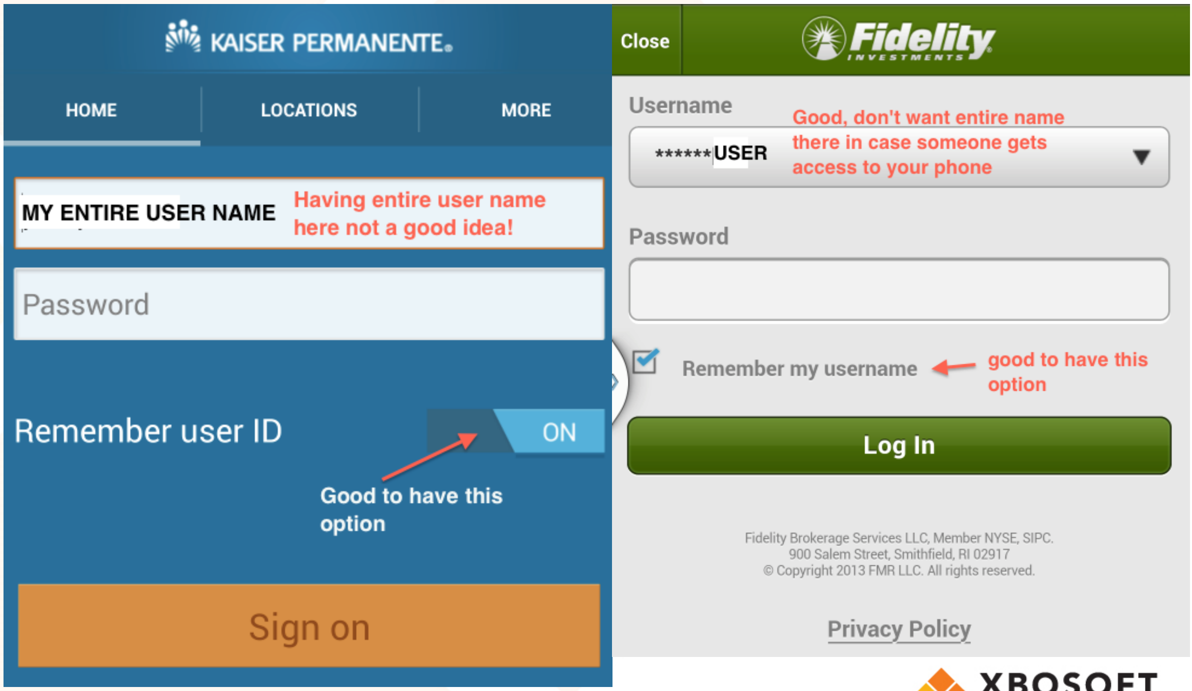
# UX and Trust

- The UX must reflect how people relate to the organization.
- Following usability best practices will ensure that the steps are intuitive and optimal.
- Users are inclined toward an company they can trust.
- Do your customers/users trust ***you*** and ***your company's app?***

XBOSOFT
Software Quality Improvement

***What other methods can create trust or distrust?***

# Apps and Trust--UX



MY ENTIRE USER NAME

Having entire user name here not a good idea!

Remember user ID ON

Good to have this option

Username ******USER

Good, don't want entire name there in case someone gets access to your phone

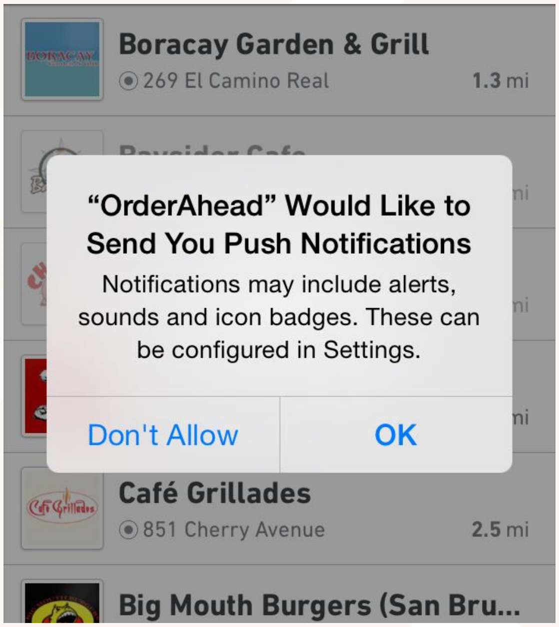
☑ Remember my username

good to have this option

Fidelity Brokerage Services LLC, Member NYSE, SIPC.
900 Salem Street, Smithfield, RI 02917
© Copyright 2013 FMR LLC. All rights reserved.

XBOSOFT
Software Quality Improvement

# Ask Permission



Boracay Garden & Grill

269 El Camino Real 1.3 mi

"OrderAhead" Would Like to Send You Push Notifications

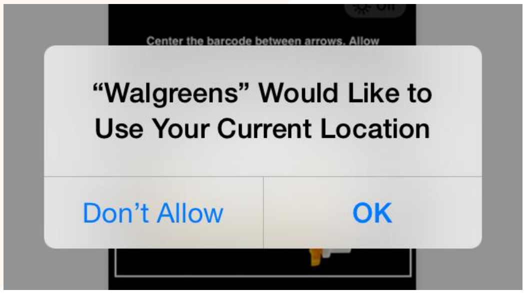
Notifications may include alerts, sounds and icon badges. These can be configured in Settings.

Café Grillades

851 Cherry Avenue 2.5 mi

Big Mouth Burgers (San Bru...

- Create trust
- Don't give users the creeps

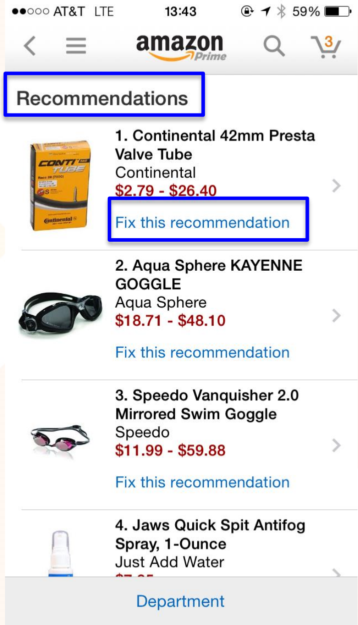


Center the barcode between arrows. Allow

"Walgreens" Would Like to Use Your Current Location

XBOSOFT Software Quality Improvement

# Creating Satisfaction Anticipate, Ask and Listen

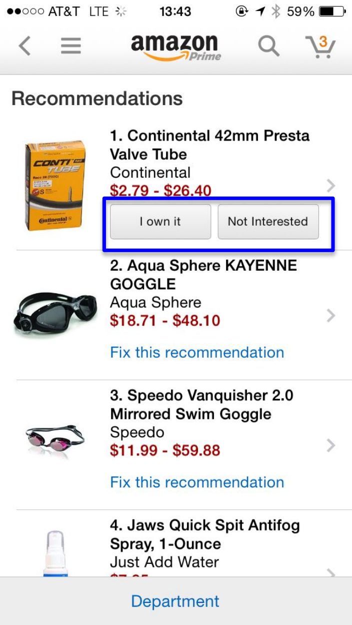


1. Continental 42mm Presta Valve Tube
Continental
$2.79 - $26.40
Fix this recommendation

2. Aqua Sphere KAYENNE GOGGLE
Aqua Sphere
$18.71 - $48.10
Fix this recommendation

3. Speedo Vanquisher 2.0 Mirrored Swim Goggle
Speedo
$11.99 - $59.88
Fix this recommendation

4. Jaws Quick Spit Antifog Spray, 1-Ounce
Just Add Water



1. Continental 42mm Presta Valve Tube
Continental
$2.79 - $26.40

2. Aqua Sphere KAYENNE GOGGLE
Aqua Sphere
$18.71 - $48.10
Fix this recommendation

3. Speedo Vanquisher 2.0 Mirrored Swim Goggle
Speedo
$11.99 - $59.88
Fix this recommendation

4. Jaws Quick Spit Antifog Spray, 1-Ounce
Just Add Water

- Learn about your users
- Let them know you are listening
- And what you want to know…

XBOSOFT Software Quality Improvement

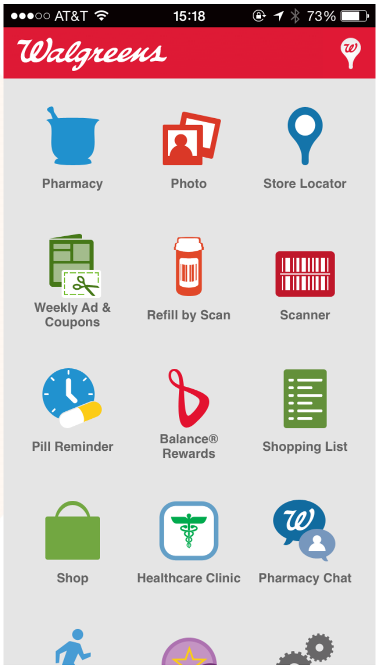
# Creating Satisfaction

# Anticipatory UX

- UX that provides the functions and services for unexpressed needs and wants of your users through use of context and technology
- People are busy – email reminders mostly have little context
- Depending on the circumstances and the customer preferences, use other communication channels
- Works for customers who want this communication and depends on the service/product that the communication is in regards to -- Context

# Walgreens

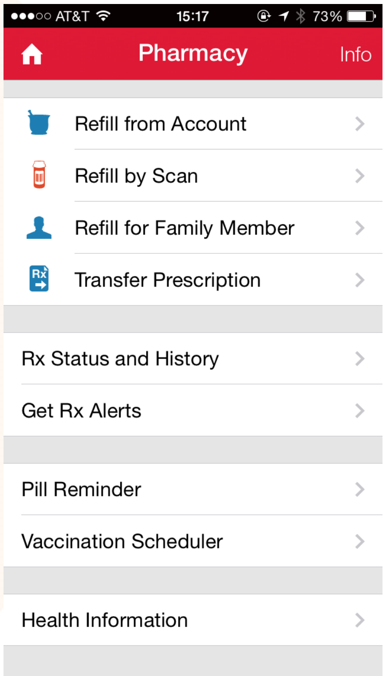
- Refill prescriptions online or mobile. If you forget, they notify you.
- Just reply, and they will refill your prescription.
- When it is ready for pickup, they notify you again.
- If you forget to pick it up they will then notify to remind you that it's ready… multiple times (specified by you).
- Once picked up, they thank you through their rewards program.



# Walgreens

Goal:

- Decrease the amount of time between a customer being eligible for a refill, and actually refilling it.
- Decrease the time between when the refill is ready and the customer collects it.
- Make money only when the prescription is picked up.



# Remind But Don’t Pester

- Fine line between using multiple methods of communication and badgering.
- Provide way for customers to inform you of their preferences.
- This is simple through opt-out links on email communication and the “stop” message for text messaging.

# Anticipate and Satisfy

- For example, minor and regularly scheduled service such as an oil change.
- Don’t know how many pills you have remaining
- Don’t know how many miles the customer has driven since their last oil change.
- However, based on the customer’s history, can be closely approximated.

# Mobile Usability

- What we have learned so far:
  - What is usability/UX
  - Design considerations
- What's next
  - Special considerations for mobile devices
  - Integration
  - Evaluation and testing criteria

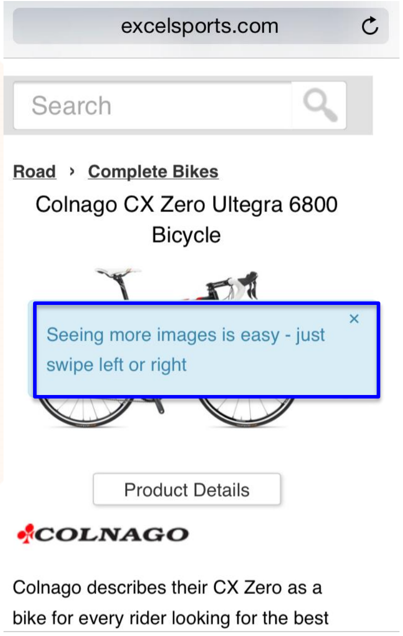
# Mobile Usability Review

- Just having app/website is not enough
- What matters for users
  - Time to load the website
  - User interface
  - Functions available
    - What is a key function?
- Usability design needs to be specific to mobile, else usability effect and UX will suffer

How many of you have an m.companyname .com site?

# Mobile Usability
# Different Platforms Need Different Design

- **Horizontal swiping** now universal
  - Include visible cue
  - Avoid **swipe ambiguity**: don't use same swipe gesture to mean different things
  - Use the same meanings for phones and tablets
  - Tablet users typically *expect* to horizontally swipe while desktop websites avoid horizontal scrolling
- *Mobile sites should perform better than full sites when used on a mobile device.*
- ***Mobile apps should integrate with the desktop version***



excelsports.com

Search

Road › Complete Bikes

Colnago CX Zero Ultegra 6800 Bicycle

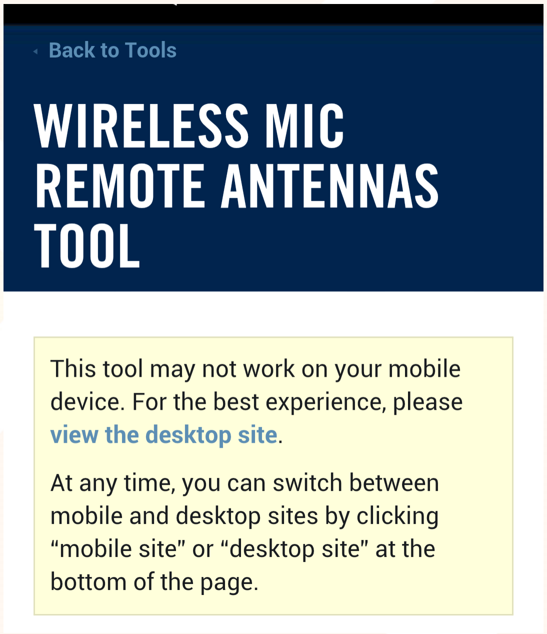
Seeing more images is easy - just swipe left or right ×

Product Details

COLNAGO

Colnago describes their CX Zero as a bike for every rider looking for the best

# Web-Mobile Integration



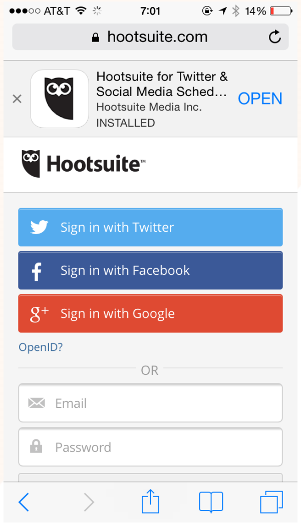
‹ Back to Tools

WIRELESS MIC REMOTE ANTENNAS TOOL

This tool may not work on your mobile device. For the best experience, please view the desktop site.

At any time, you can switch between mobile and desktop sites by clicking “mobile site” or “desktop site” at the bottom of the page.

- Poor integration
- BUT Straightforward about it



●●●○○ AT&T 7:01 14%

hootsuite.com

× Hootsuite for Twitter & Social Media Sched… Hootsuite Media Inc. INSTALLED OPEN

Hootsuite

Sign in with Twitter

Sign in with Facebook

Sign in with Google

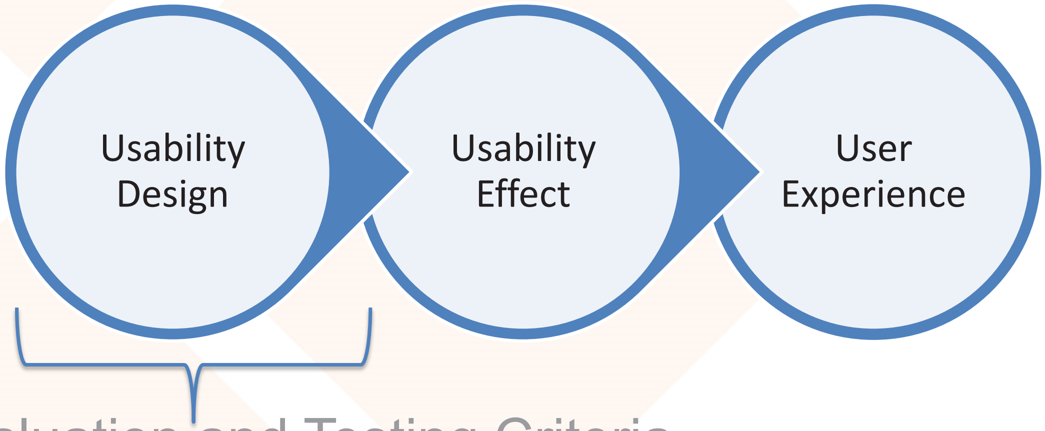
OpenID?

OR

Email

Password

# Mobile Usability Design

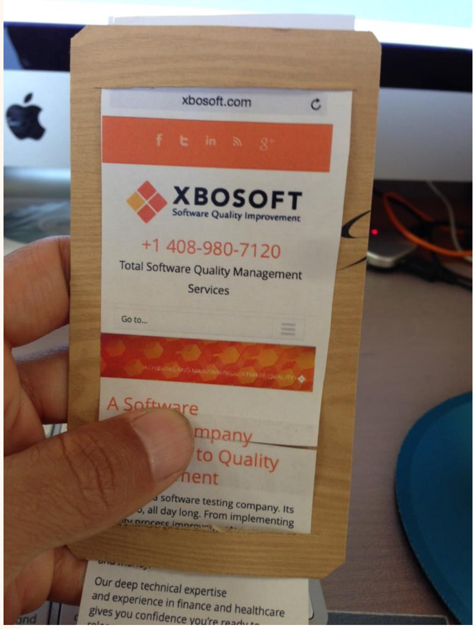


Evaluation and Testing Criteria

# Using Paper Prototypes

- Low tech but very useful
- Use a cardboard template to simulate what the user sees
  – Create your own
  – Simple but easy and fast to find issues

# Sometimes Old fashioned Can Work Fast to Find Issues


xbosoft.com
XBOSOFT
Software Quality Improvement
+1 408-980-7120
Total Software Quality Management Services
Go to...
A Software
mpany
to Quality
nent
Our deep technical expertise
and experience in finance and healthcare

# Evaluation and Testing Criteria

- Typing/Input
- Entry Widgets (Drop downs, links, and lists)
- Sort and Filter
- Menus and Forms
- Registration
- Navigation
- Search implementation
- Errors
- Visibility

What kinds of evaluation criteria would be important here?

## TYPING

# Typing

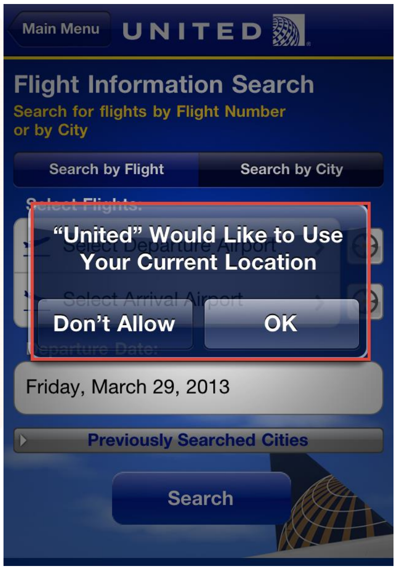
- Typing can be quite difficult for mobile users
  - Reduced keypad
  - Big fingers for big people
  - Lessened dexterity (older users – MORE AND MORE)
- Reduce the cost of typing

# Typing-Use Personalization and History to Provide Defaults

- Defaults can be based on what the user has typed or submitted in the past (e.g., zipcodes, names, addresses)
  - Don't use 0 as the default for a telephone number or zip code.
  - Remember the last value last typed for their zip code and use for subsequent entries.
- Allow Easy Deletion for Defaults

# Compute/Fill In Values

- **Compute and fill in** for the user if possible and appropriate
  - i.e. detect location for location aware apps
  - Zip code → compute other information automatically
  - Adding postage costs to the total costs
- Users often **expect** location information determined automatically.



Main Menu UNITED

Flight Information Search

Search for flights by Flight Number or by City

Search by Flight | Search by City

Select Flights:

Select Departure Airport

Select Arrival Airport

"United" Would Like to Use Your Current Location

Don't Allow | OK

Departure Date:

Friday, March 29, 2013

Previously Searched Cities

Search

# Typing-Input

- Characteristics or criteria would we evaluate/test for UX
  – What would you list here?

# Typing-Input

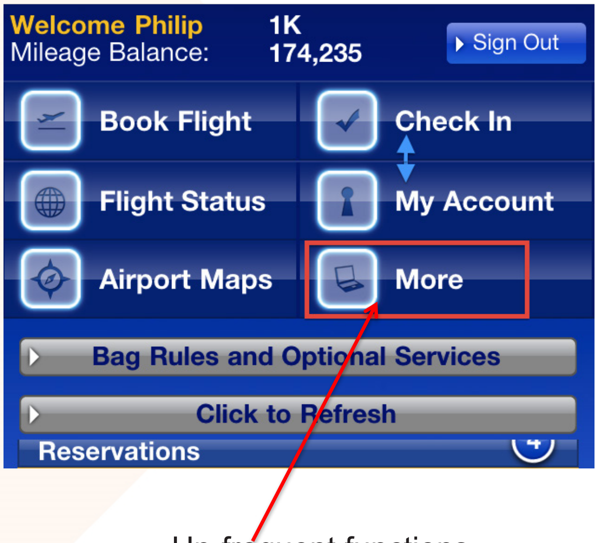
- Characteristics or criteria would we evaluate/test for UX
  – Typing mistake tolerant
  – Defaults provided
  – Deletion of defaults
  – Computed values
  – What others to add?

Dropdowns, buttons, boxes, links, lists, etc…

## WIDGET ENTRY

## Dropdown Boxes, Buttons, and Links

- Build in tolerance for error
  - Leave space around widgets that need to be clicked (buttons, arrows for dropdown boxes, links, scrollbars)
- Be consistent in using space around links and widgets

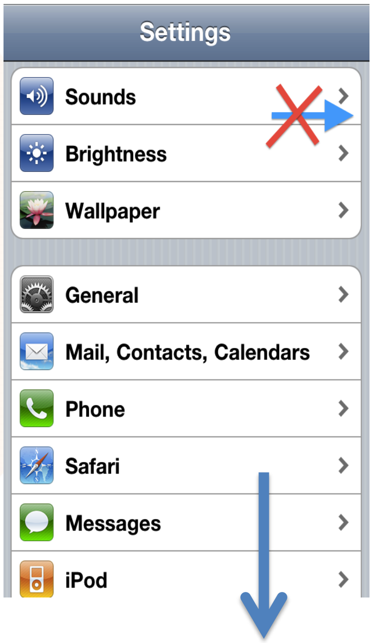


Welcome Philip 1K
Mileage Balance: 174,235

Un-frequent functions

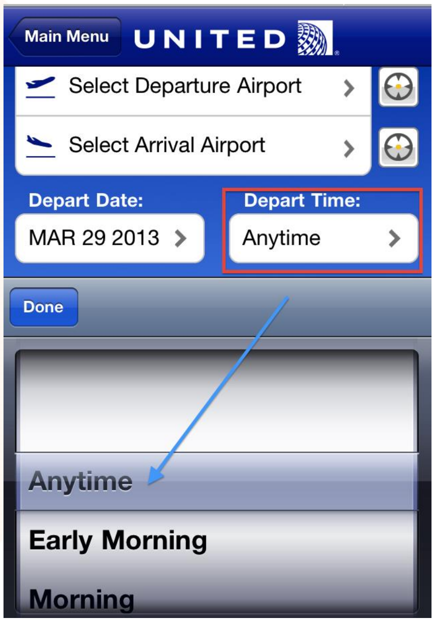
# Lists and Scrolling

- All the items on a list should go on the same page:
  - if the items are text-only, and
  - if they are sorted in an order that matches the needs of the task.
- Users are willing to scroll down a list if they know how far along they are and how much more work they have to do.
  - Sorted alphabetically



# Carousels

- Use simple controls for going back and forward.
  - Manual carousel allows the user to control the carousel **if decides to use it.**

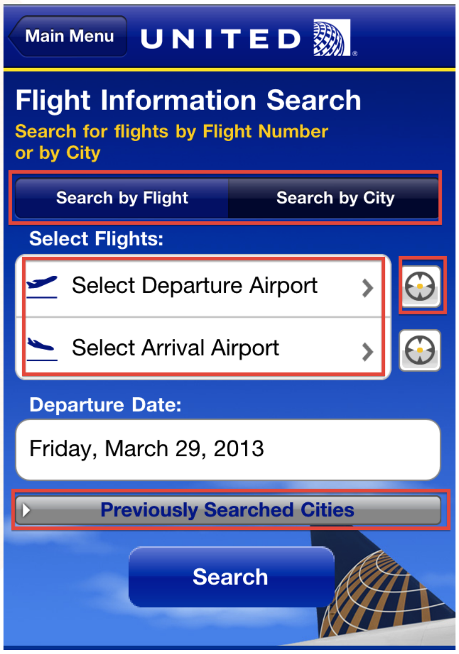


Depart Date: MAR 29 2013

Depart Time: Anytime

# Sorting and Filtering

- Many ways to sort the same list, depending on the user and the task
- For different sort criteria, provide the option to sort that list according to all criteria



Main Menu UNITED

Flight Information Search

Search for flights by Flight Number or by City

Think of the searches your users will do?

# Exercise

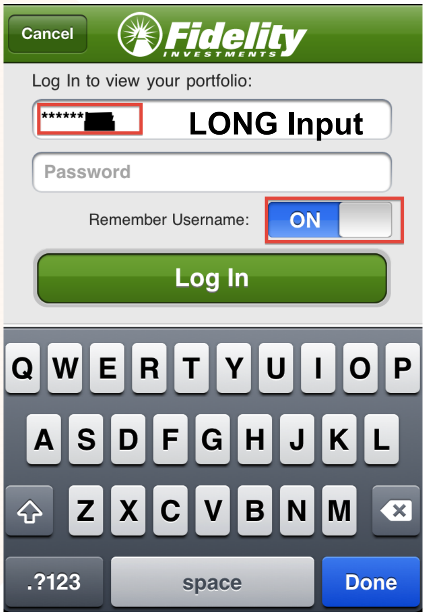
- *Exercise: Find out if a flight from Munich to London is on time for today. Try 2 diff airlines.*
  - Don’t have the number of the flight or the exact time
  - Look through the list of arrivals at London Heathrow Airport
- *Exercise: Do a search on Wine Spectator to see what search criteria you can use.*

Lots of input! How do we handle this?

## FORMS AND REGISTRATION/LOGIN

# Forms-Textboxes

- Textboxes in a form should be long enough so they fit on the screen and accommodate user input.



Log In to view your portfolio:

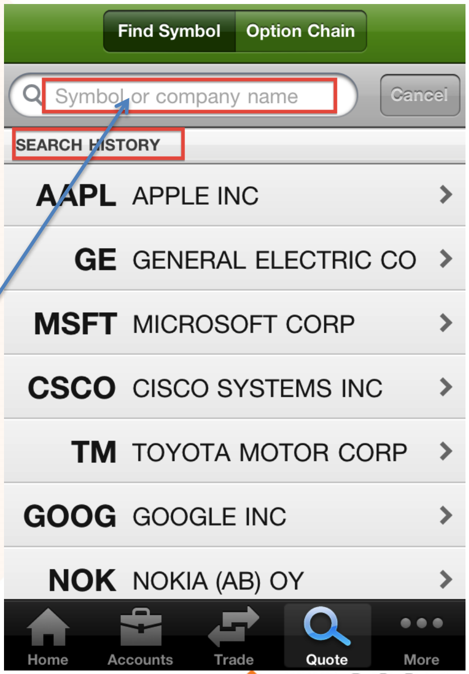
****** [redacted] LONG Input

Password

Remember Username: ON

# Form Field Descriptions

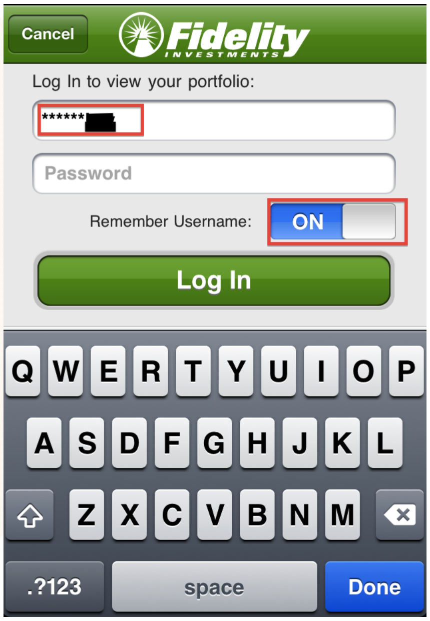
- Small screen – easy to lose context and become disoriented so descriptions are important.
- Description critical in a form, where many fields have to be filled in.
- OR use the technique shown here.



XBOSOFT Software Quality Improvement

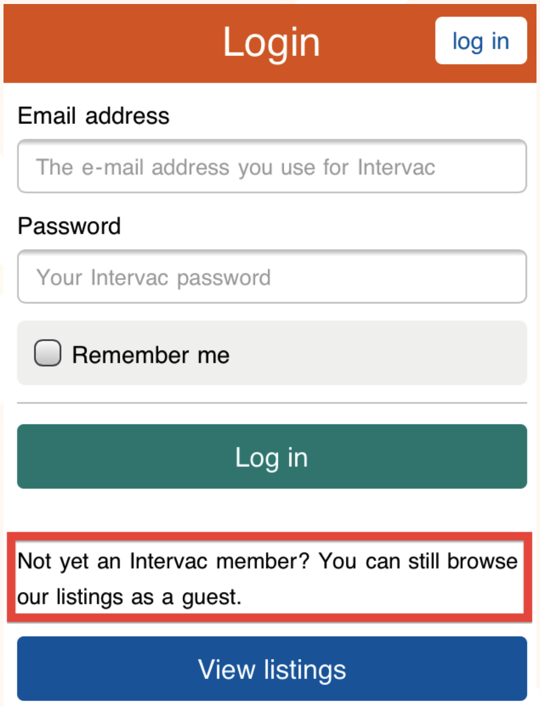
# Login/User Registration

- Passwords and usernames are hard
- Do not ask people to register on a mobile device
  - Sign in is different than registration
- Skipping registration should be the default option, but less features
- Registration incurs an extra time/click cost on mobile devices



XBOSOFT Software Quality Improvement

# User Registration – Cont.



Login

log in

Email address

The e-mail address you use for Intervac

Password

Your Intervac password

□ Remember me

Log in

Not yet an Intervac member? You can still browse our listings as a guest.

View listings

- Offer the option of proceeding without registration or sign in
- What would you give users access to without and ID?

Similar to sort and filter but little different-let's see

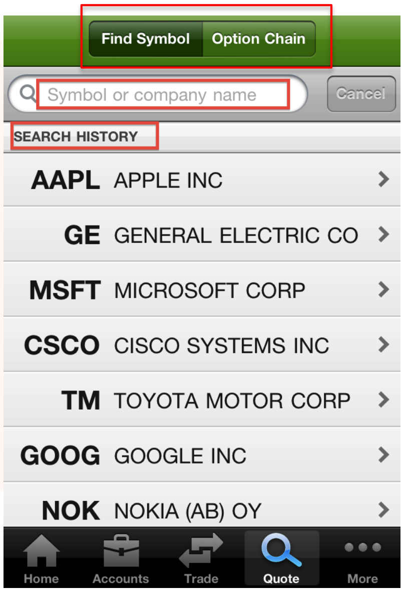
# SEARCH

# Implementing Search Functions

- Search takes space and grabs user's attention, distracting from other tasks
- Think clearly about your users' task at hand
  - Browsing tasks
  - Searching tasks
  - Execution oriented

# Search Boxes

- Length of the search box >= average search string. Use largest possible size that will fit on the screen.
  - Preserve search strings between searches.
  - Use auto-completion and suggestions.
  - Give history-based defaults.
- Do not use several search boxes with different functions on the same page.
  - one for stocks by company symbol
  - one for stocks by company name
  - one for search within their articles

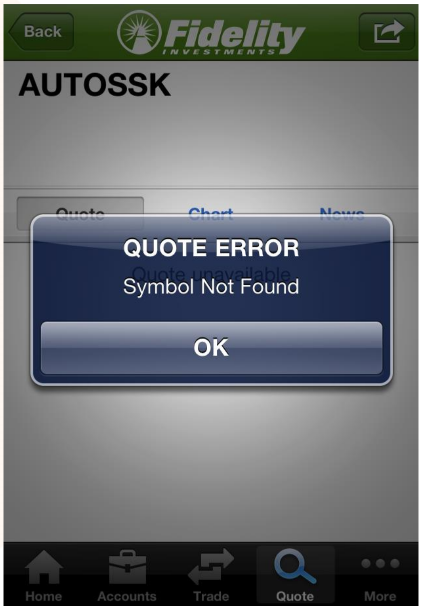


SEARCH HISTORY

| | |
|---|---|
| AAPL | APPLE INC |
| GE | GENERAL ELECTRIC CO |
| MSFT | MICROSOFT CORP |
| CSCO | CISCO SYSTEMS INC |
| TM | TOYOTA MOTOR CORP |
| GOOG | GOOGLE INC |
| NOK | NOKIA (AB) OY |

# Search – No Results

- If the search returns no results, offer alternatives
  - No search results often due to typo in query
  - Inform user of search failure
  - Offer results to alternative searches
- Search by company name ("Autodesk") in the search-by-symbol box → 0 search results.
- Zero search results should have triggered a search by name



QUOTE ERROR

Quote unavailable.

Symbol Not Found

# Search Implementation

| Sub-attribute | Where to apply | Evaluation: 2 – all, 1 – partial, 0 - none |
|---|---|---|
| Search box length appropriateness | | |
| Auto-completion appropriateness | | |
| Error handling | | |
| No Results handling | | |

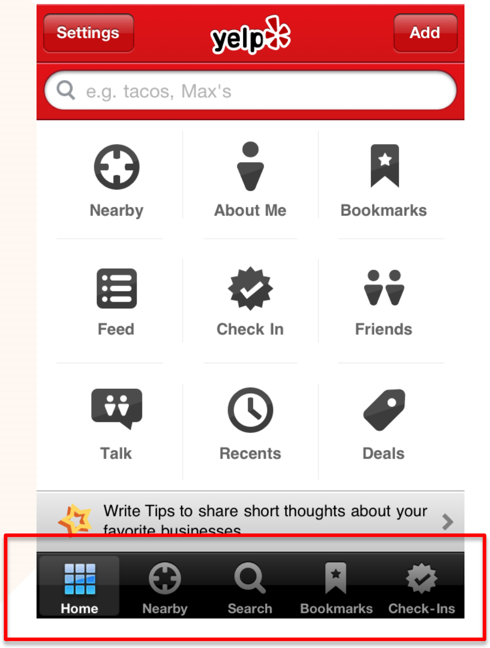
Where do you want to go today?

# NAVIGATION

XBOSOFT
Software Quality Improvement

# Navigation

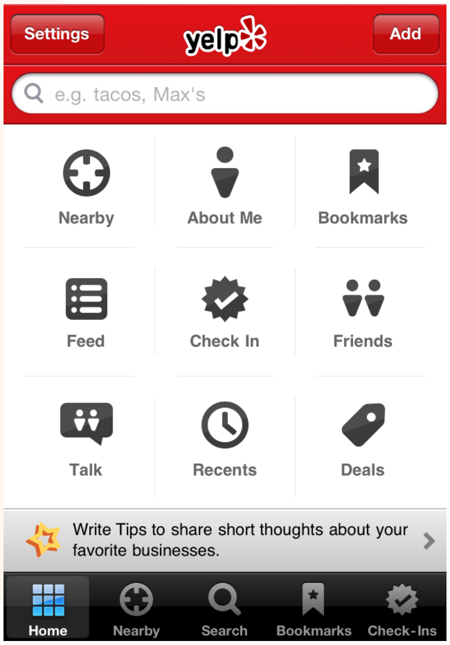
- Users have different goals in mind and use apps in different context
- Must easily find their way.
- Navigation on desktop apps is standardized (by MSFT defacto)
- Navigation should be the same on mobile. But it's NOT
- Due to the small screens, adaptations required



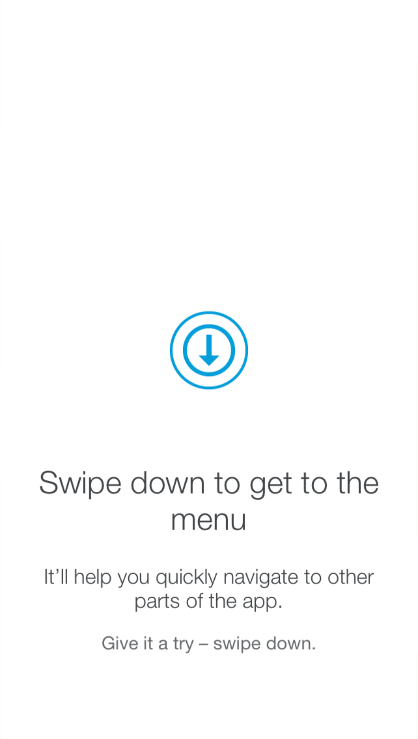
XBOSOFT
Software Quality Improvement

# Required Clicks

- Minimize the clicks that the user needs to go through in order to input information on your site.
- Compress steps together on a single screen if appropriate rather than separate screens.



# Swiping



Swipe down to get to the menu

It'll help you quickly navigate to other parts of the app.

Give it a try – swipe down.

- Let users know what they can and cannot do

# Exercise

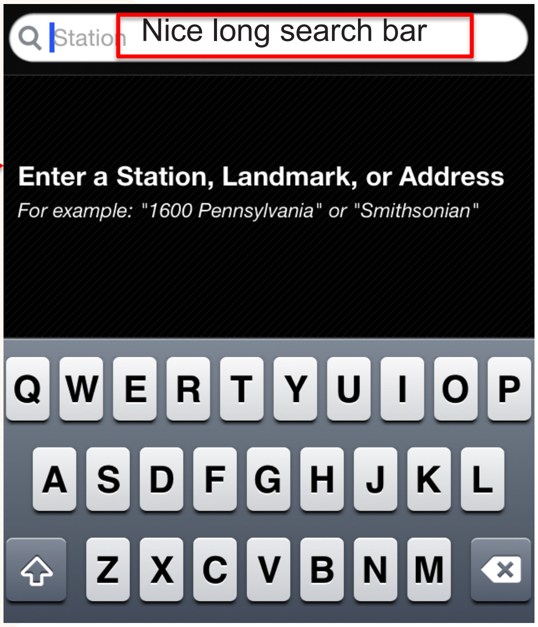
- Break into small teams of 2-3 people
- Complete the task of returning a phone call
  - Start from home screen
- What criteria would you use to rate navigation?
- Tell me what results you got with 2-3 different phones/OS

Keep It Simple Stupid

# ERRORS

# Errors - Prevention

- Prevention is the first thing we want to do!
  - Context sensitive help
  - What other ways can we prevent errors?



Station

Nice long search bar

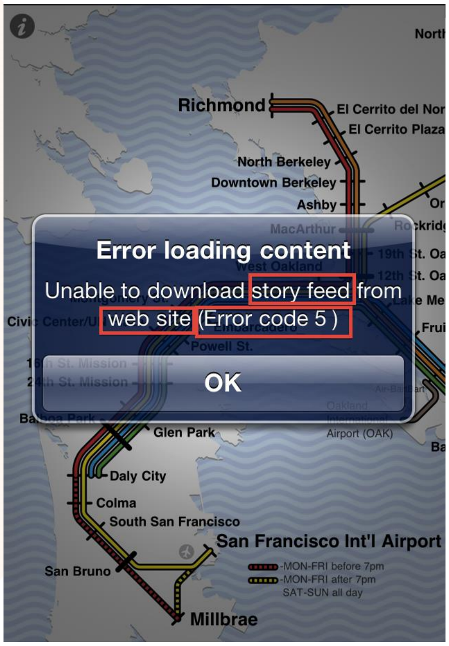
Enter a Station, Landmark, or Address

*For example: "1600 Pennsylvania" or "Smithsonian"*

XBOSOFT
Software Quality Improvement

# Make Error Messages Simple

- Error messages need to be seen and presented simply
- Tell the user where the error happened (browser, website, application, etc.), what does not work, and what the user needs to do.
- Explain where the error is coming from.



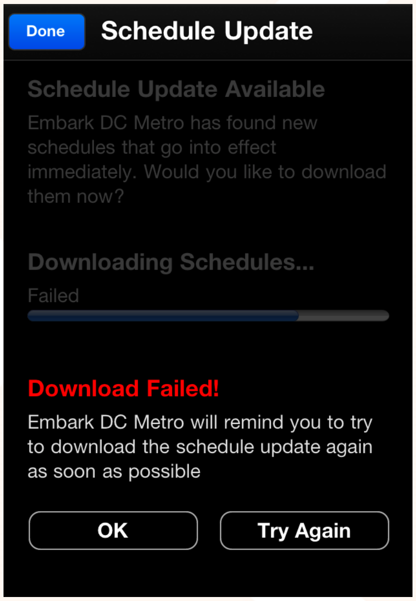
**Error loading content**

Unable to download story feed from web site (Error code 5 )

OK

XBOSOFT
Software Quality Improvement

# Errors-Status


Done

Schedule Update

Schedule Update Available

Embark DC Metro has found new schedules that go into effect immediately. Would you like to download them now?

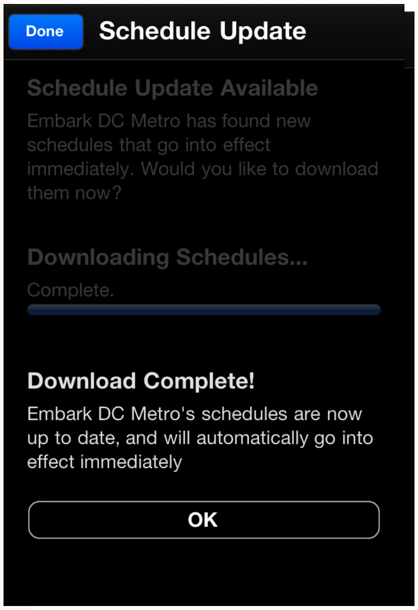
Downloading Schedules...

Failed

Download Failed!

Embark DC Metro will remind you to try to download the schedule update again as soon as possible

OK | Try Again


Done

Schedule Update

Schedule Update Available

Embark DC Metro has found new schedules that go into effect immediately. Would you like to download them now?

Downloading Schedules...

Complete.

Download Complete!

Embark DC Metro's schedules are now up to date, and will automatically go into effect immediately

OK

Let the user know what’s going on! Error or not

(Embark – Metro – Washington DC)

# Error Handling

- Form input error location notification
- Error explanation and status
- Error prevention
- Other

What other Error Handling characteristics would you use for YOUR app to evaluate/test it?

Monkey See Monkey Do

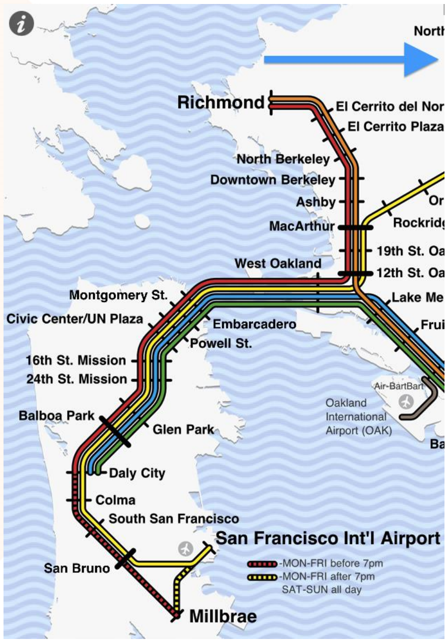
# DISPLAY AND VISIBILITY

# Visibility
# Images, Animation, Videos, Text

- Include images only if they add **meaningful** content
  - Not for decoration: How does the image contribute to the content, task or message?
  - Benefit versus cost - they take space on the page and download slowly for online app.
  - Do thumbnails really add value or just look cool?

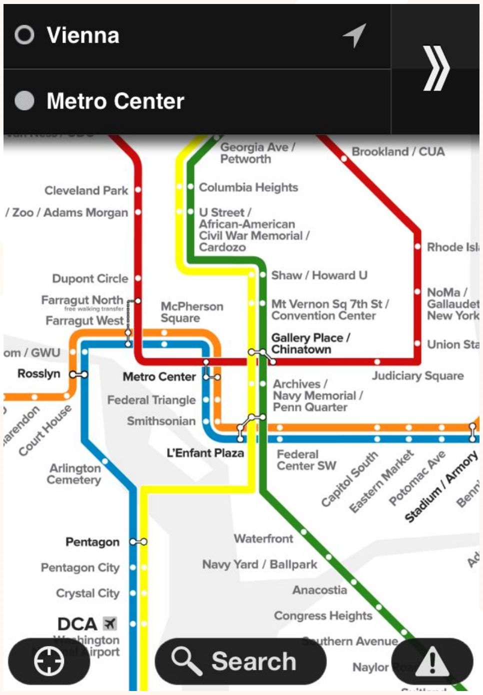
# Image Sizes

- Do not use image sizes that are bigger than the screen → entire image should be viewable with no scrolling.
  - Details are important but only after the user has interest level in the picture.



Bart

# Horizontal Scroll



What does this tell you?

DC Metro

# Displaying Content

***People rarely read all the text***

- For information apps, new content should be given priority.
- Summarize content: When users click on the summary/headline, they are taken to more detail that does not repeat information in the summary.
- Concise, direct language
- Format, use of bullets
- Think about context!

XBOSOFT
Software Quality Improvement

# Special Mobile Application Domain Considerations (Context)

- Location – Aware
- Shopping and Ecommerce
- Banking and Finance

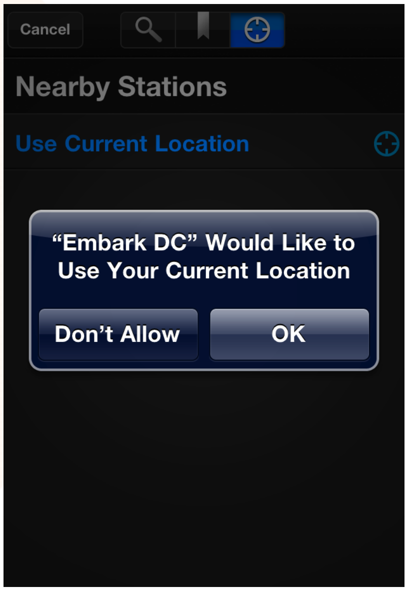
XBOSOFT
Software Quality Improvement

# Location-Aware Applications

- Location aware applications are becoming more prevalent on mobile applications.
- Whenever you have location information in your app, link it to a map and include a way of getting directions.
- Cutting and pasting is inconvenient or impossible.
- Don’t just give users an address without a way to find it on a map

# Location Aware Apps

- Give users a choice!
- Create TRUST



Nearby Stations

Use Current Location

“Embark DC” Would Like to Use Your Current Location

Embark – Metro
Washington DC

# Ecommerce Apps

- With products, use image thumbnails that are big enough so that the user can recognize what it is
- Provide option to email a product to a friend.
- Provide option to save that product and return to it later.
- Provide option to save the product in a wish list.
- Allow users to save the items that they like, so that they could go back to them on a larger screen (integrate with desktop app) and inspect them more closely.

# Ecommerce Apps

**Always Provide Critical Information:**

- Locations and opening hours (if applicable) - checking location information, business hours, and order status are most frequent types of mobile based ecommerce activities
- Shipping costs - Easily accessible and noticeable (trust).
- Phone number - Easily findable so users can call if there is a problem with their order
- Order status

# Banking Apps

- Mobile banking and financial transactions growing
- Provide clear understanding of mobile app security
- Users hesitant to make transactions
  - Enable confirmation by emailing a message to themselves.
  - Confirmations are hard to write down when you are ‘mobile’.
  - Give users comfort they will have later access to that confirmation number.

XBOSOFT
Software Quality Improvement

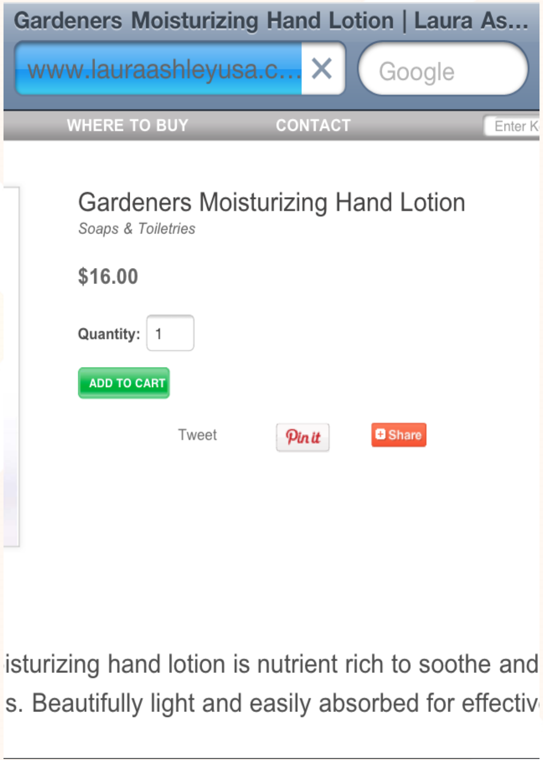
# Let’s Step Through a Few Examples

Usability and UX For Mobile
(time permitting)

XBOSOFT
Software Quality Improvement

# LA

- What is wrong here?



Gardeners Moisturizing Hand Lotion | Laura As...

www.lauraashleyusa.c...

Gardeners Moisturizing Hand Lotion

*Soaps & Toiletries*

$16.00

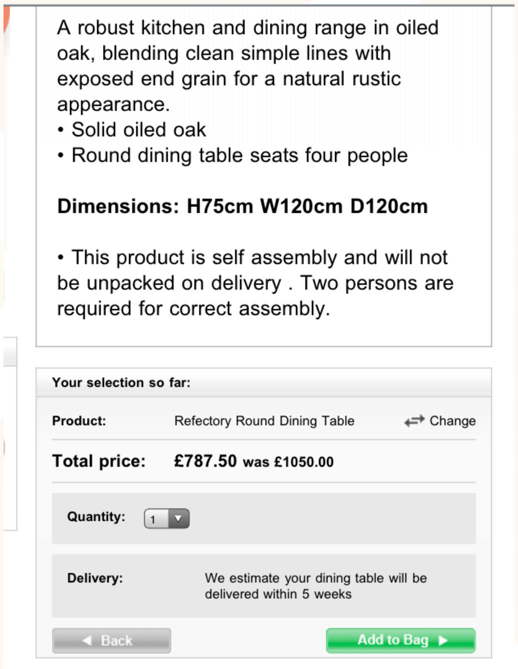
Quantity: 1

isturizing hand lotion is nutrient rich to soothe and
s. Beautifully light and easily absorbed for effectiv

XBOSOFT
Software Quality Improvement

# LA MS

- Consistent green
- Obviously want you to click and add to the bag



A robust kitchen and dining range in oiled oak, blending clean simple lines with exposed end grain for a natural rustic appearance.

- Solid oiled oak
- Round dining table seats four people

**Dimensions: H75cm W120cm D120cm**

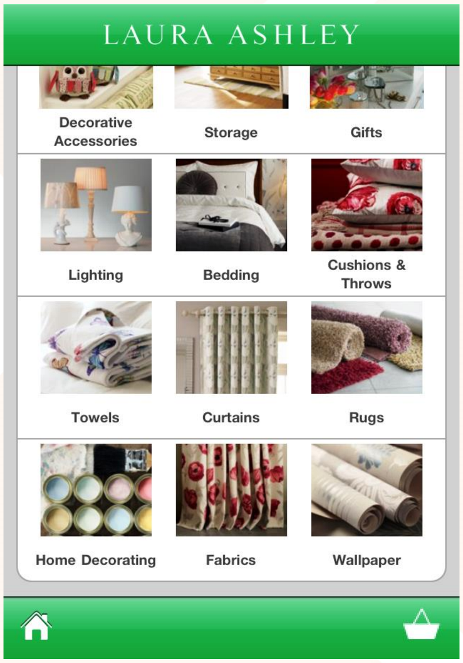
- This product is self assembly and will not be unpacked on delivery . Two persons are required for correct assembly.

**Your selection so far:**

| | | |
|---|---|---|
| **Product:** | Refectory Round Dining Table | ⇄ Change |
| **Total price:** | **£787.50** was £1050.00 | |
| **Quantity:** | 1 | |
| **Delivery:** | We estimate your dining table will be delivered within 5 weeks | |

XBOSOFT
Software Quality Improvement

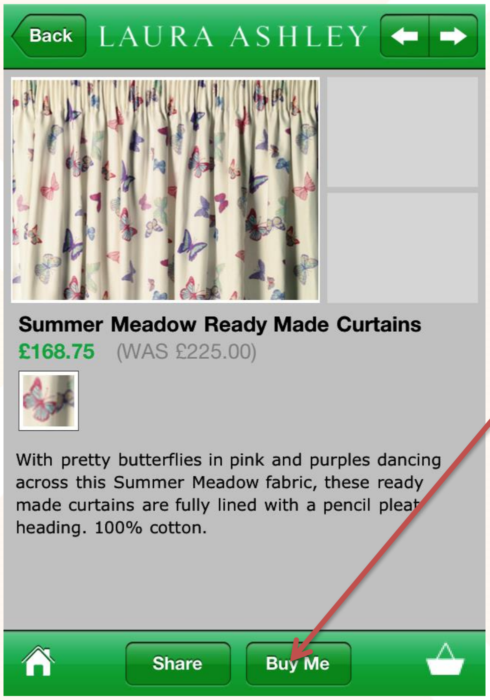
# LA Mobile App



- Big buttons
- Good for big fingers
- Semi-meaningful icons on the bottom
  - Note contextual location

XBOSOFT
Software Quality Improvement

# Buy or Share?



**Summer Meadow Ready Made Curtains**
£168.75 (WAS £225.00)

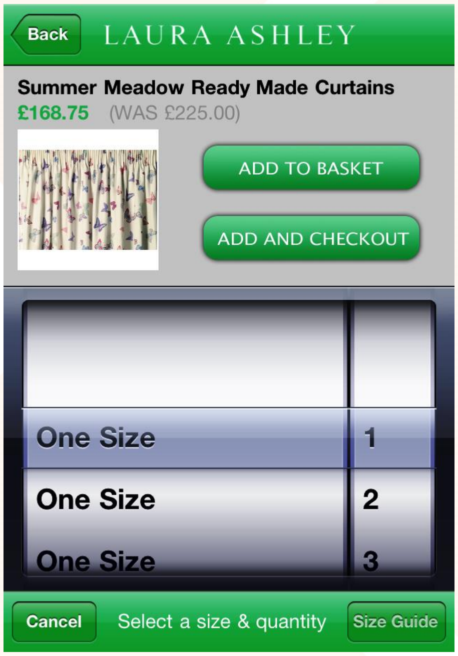
With pretty butterflies in pink and purples dancing across this Summer Meadow fabric, these ready made curtains are fully lined with a pencil pleat heading. 100% cotton.

- No placement of importance or priority

**This is one place where general Usability principles apply!**
**Maybe on purpose?**

XBOSOFT
Software Quality Improvement

# Select Size

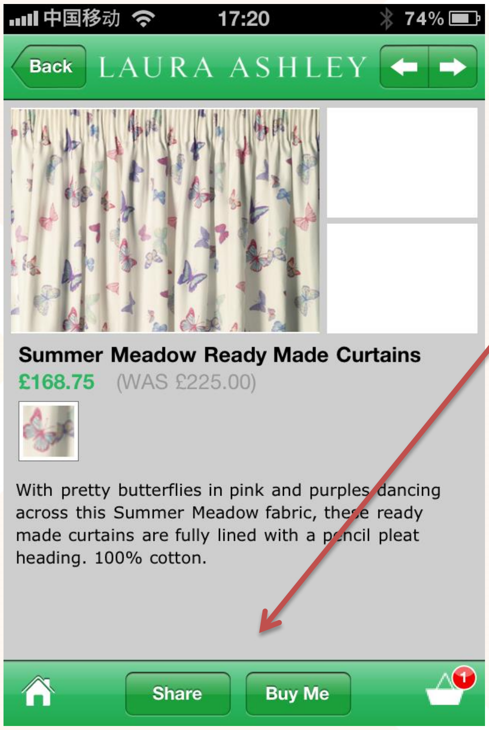


Summer Meadow Ready Made Curtains

£168.75 (WAS £225.00)

- Good navigation
- Good size buttons
- No priority

# In my Basket



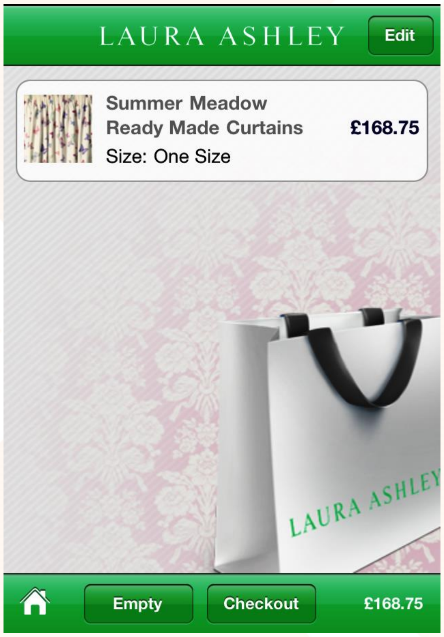
Summer Meadow Ready Made Curtains

£168.75 (WAS £225.00)

With pretty butterflies in pink and purples dancing across this Summer Meadow fabric, these ready made curtains are fully lined with a pencil pleat heading. 100% cotton.

- What should I do now?
- Buy or share?
- Indication of where we are in the process

# Shopping Basket



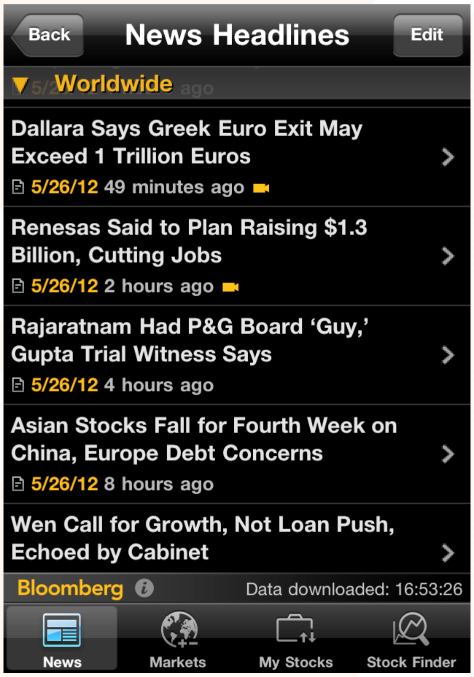
Summer Meadow Ready Made Curtains £168.75
Size: One Size

- Empty or Checkout?

***What would you recommend they do here?***

XBOSOFT
Software Quality Improvement

# Bloomberg



Dallara Says Greek Euro Exit May Exceed 1 Trillion Euros
5/26/12 49 minutes ago

Renesas Said to Plan Raising $1.3 Billion, Cutting Jobs
5/26/12 2 hours ago

Rajaratnam Had P&G Board 'Guy,' Gupta Trial Witness Says
5/26/12 4 hours ago

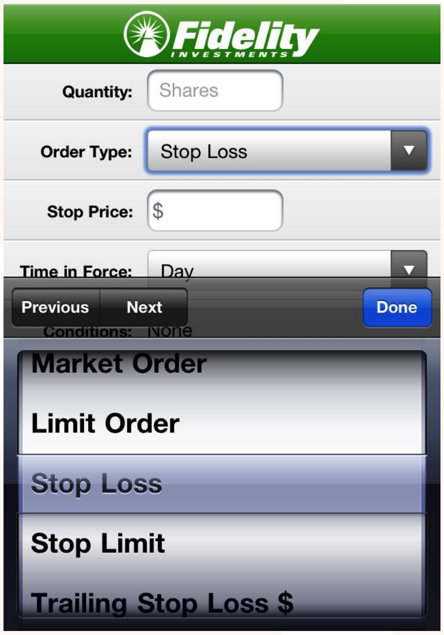
Asian Stocks Fall for Fourth Week on China, Europe Debt Concerns
5/26/12 8 hours ago

Wen Call for Growth, Not Loan Push, Echoed by Cabinet

Bloomberg
Data downloaded: 16:53:26

- Simple Info App
- Simple colors
- Meaningful Icons
- Big enough to see AND click on with big fonts
- Summary going to detail

XBOSOFT
Software Quality Improvement



Quantity: Shares

Order Type: Stop Loss

Stop Price: $

Time in Force: Day

Conditions: None

Market Order

Limit Order

Stop Loss

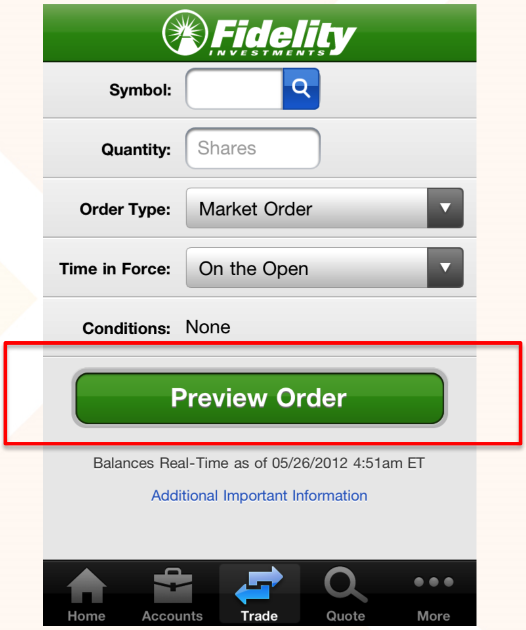
Stop Limit

Trailing Stop Loss $

# Fidelity

- Easy to use scroll
  - Thumb Friendly
- Simple colors
- Very limited task based functionality

XBOSOFT Software Quality Improvement



Symbol:

Quantity: Shares

Order Type: Market Order

Time in Force: On the Open

Conditions: None

Balances Real-Time as of 05/26/2012 4:51am ET

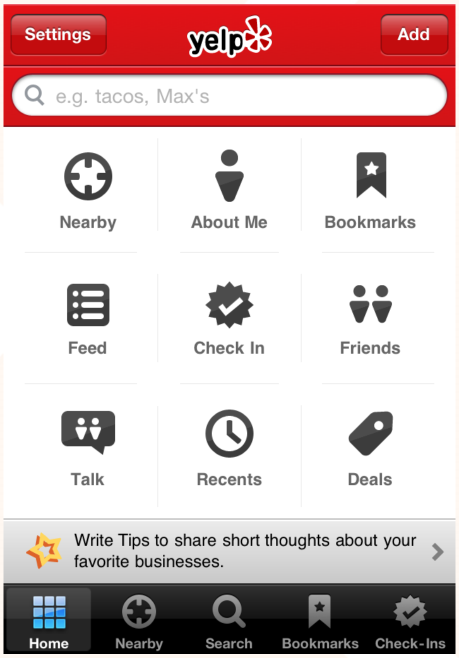
Additional Important Information

# Fidelity

- Size and Importance
- Easy to see what they want you to do
- Consistent icons on bottom
  - Simple
  - Not everything

XBOSOFT Software Quality Improvement

# Yelp

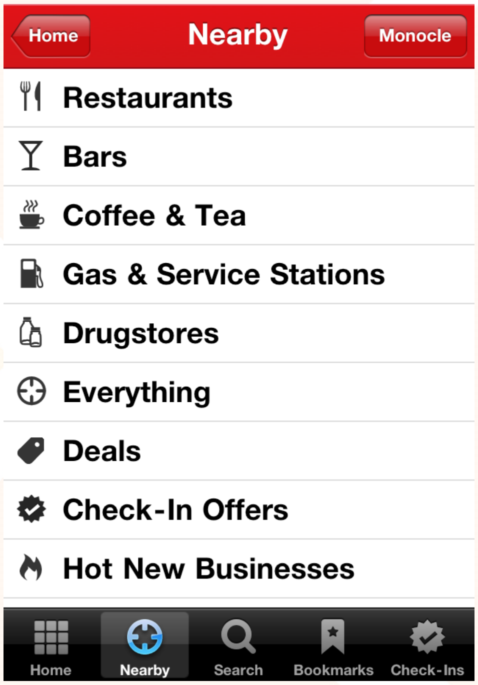


Write Tips to share short thoughts about your favorite businesses.

- Easy on the eyes
- Semi-meaningful icons

Based on what we just learned, what attributes would this app score well on?

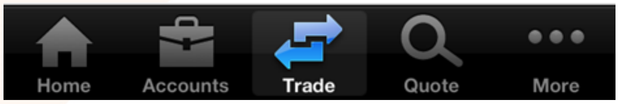
# Yelp



- Consistent
- Finger friendly
- Consistent location of icons on bottom

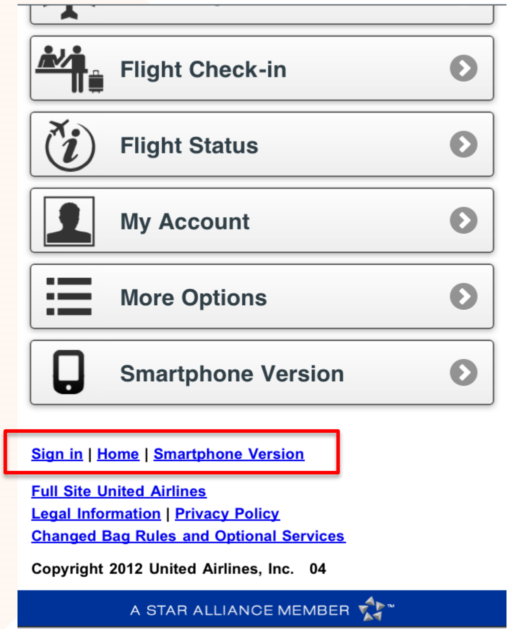
Similar to Fidelity

What does this mean?



# Making Assumptions

- Sense the platform and switch to a ‘mobile site’ to provide mobile users with more efficient web experience.
- Don’t make too many assumptions regarding the users’ expectations.
- For full United.com site, we need to scroll to the bottom to find the link.



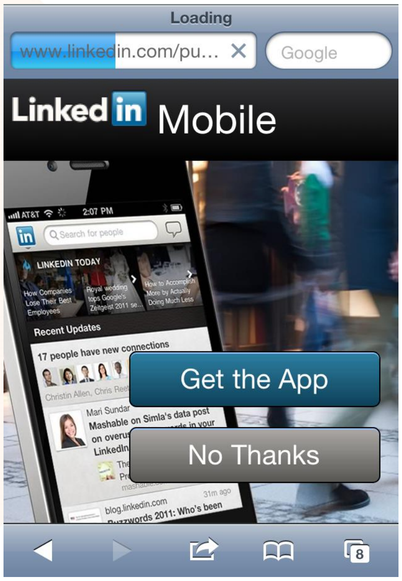
Sign in | Home | Smartphone Version

Full Site United Airlines
Legal Information | Privacy Policy
Changed Bag Rules and Optional Services

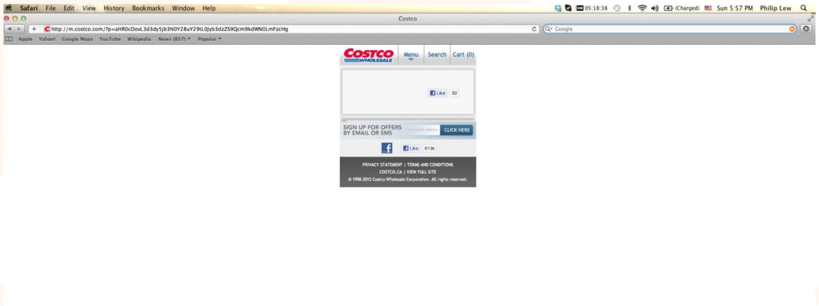
Copyright 2012 United Airlines, Inc. 04

# Give a Choice to Avoid Making the Wrong Assumption

- Managing user expectations of how the application should behave needs to be thought out carefully.
- Context of the user and their expectations is key.
- LinkedIn appears to be one step ahead and gives you a choice when accessing their site; an optimized mobile experience via an mobile-app, or just their ordinary website.



# Web-Mobile Integration



http://m.costco.com/?p=aHR0cDovL3d3dy5jb3N0Y28uY29tL0Jyb3dzZS9Qcm9kdWN0LmFzcHg

SIGN UP FOR OFFERS BY EMAIL OR SMS

PRIVACY STATEMENT | TERMS AND CONDITIONS
COSTCO.CA | VIEW FULL SITE
© 1998-2012 Costco Wholesale Corporation. All rights reserved.

- Overall integration of web-apps needs to be thought out carefully with mobile scenarios in mind.
- Friend wanted to show me a product at www.costco.com.
- He sent me a link from his iPhone. I was at my desktop, and opened the link and got the mobile version.

XBOSOFT
Software Quality Improvement

# What Is Designed, We Test

## Understanding Design Leads to Better Understanding, Evaluation, and Testing

OR SOMETIMES WE MUST DETERMINE THE TEST CRITERIA

XBOSOFT
Software Quality Improvement

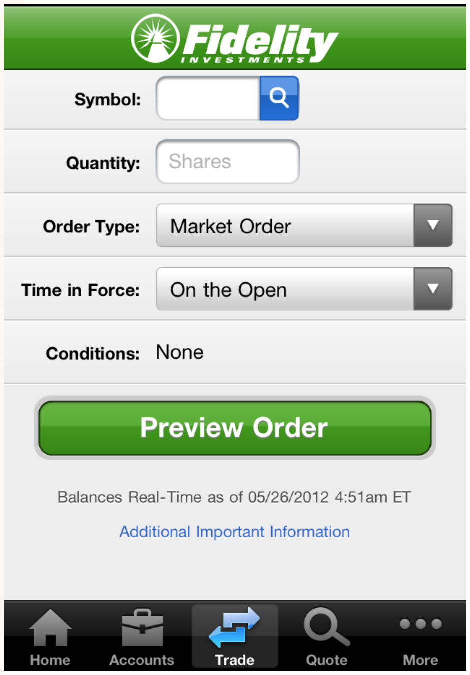
# Determine the Needs at Hand

- What need is the mobile application attempting to mobilize?
- How can the workflow be designed more efficiently to accomplish the task?
- How can the specific features and characteristics of a mobile device improve and complement the experience in contrast to the normal web-based application or other mobile platforms?

XBOSOFT
Software Quality Improvement

# Determine the Needs at Hand

- Implement top scenarios users want and optimizing efficiency for those scenarios ONLY.
  - Filling an order was a scenario optimized just for mobile.
  - Only 4 data items to fill in, 2 being scroll buttons, biggest button indicates precisely the task purpose.



Symbol:

Quantity: Shares

Order Type: Market Order

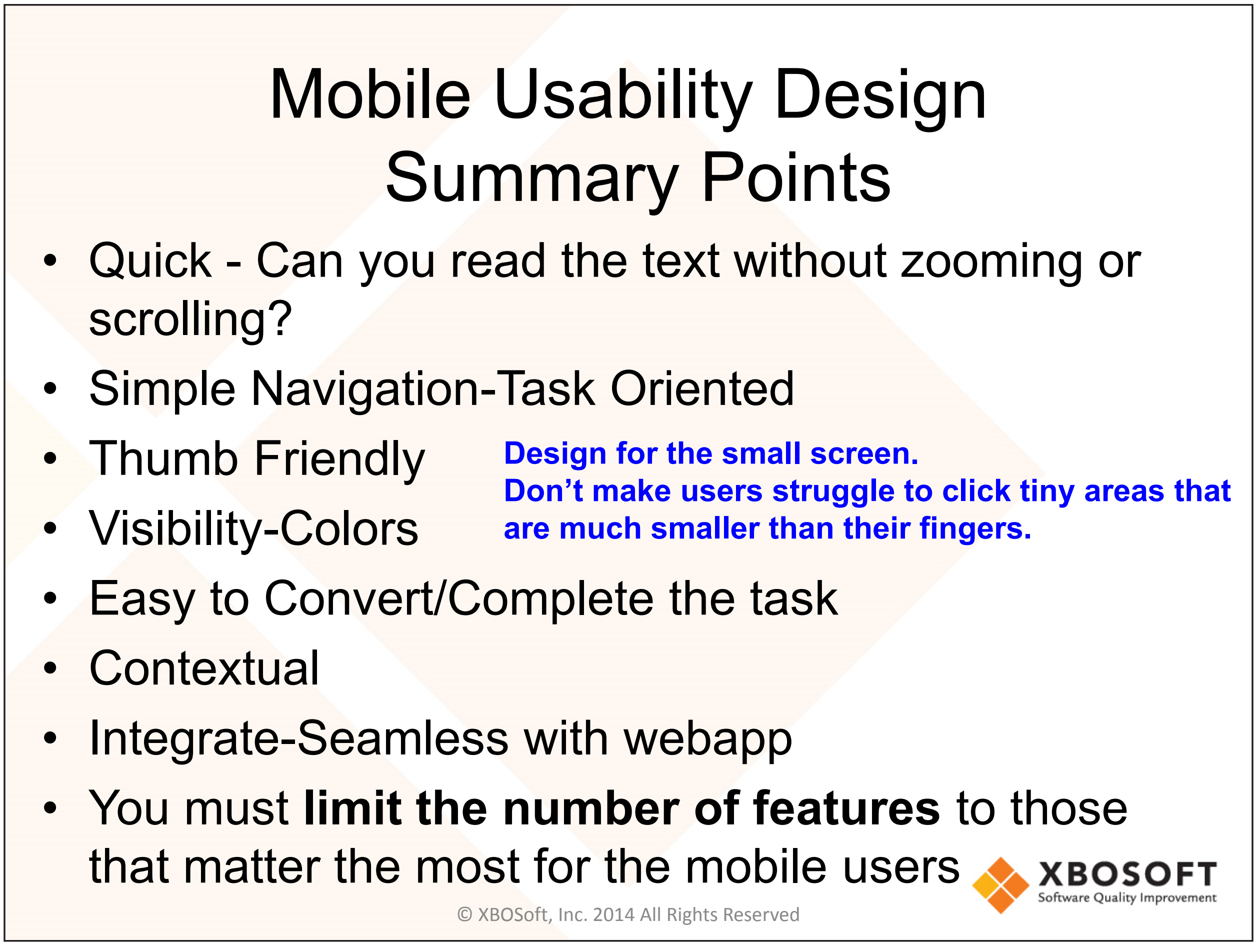
Time in Force: On the Open

Conditions: None

Balances Real-Time as of 05/26/2012 4:51am ET

Additional Important Information

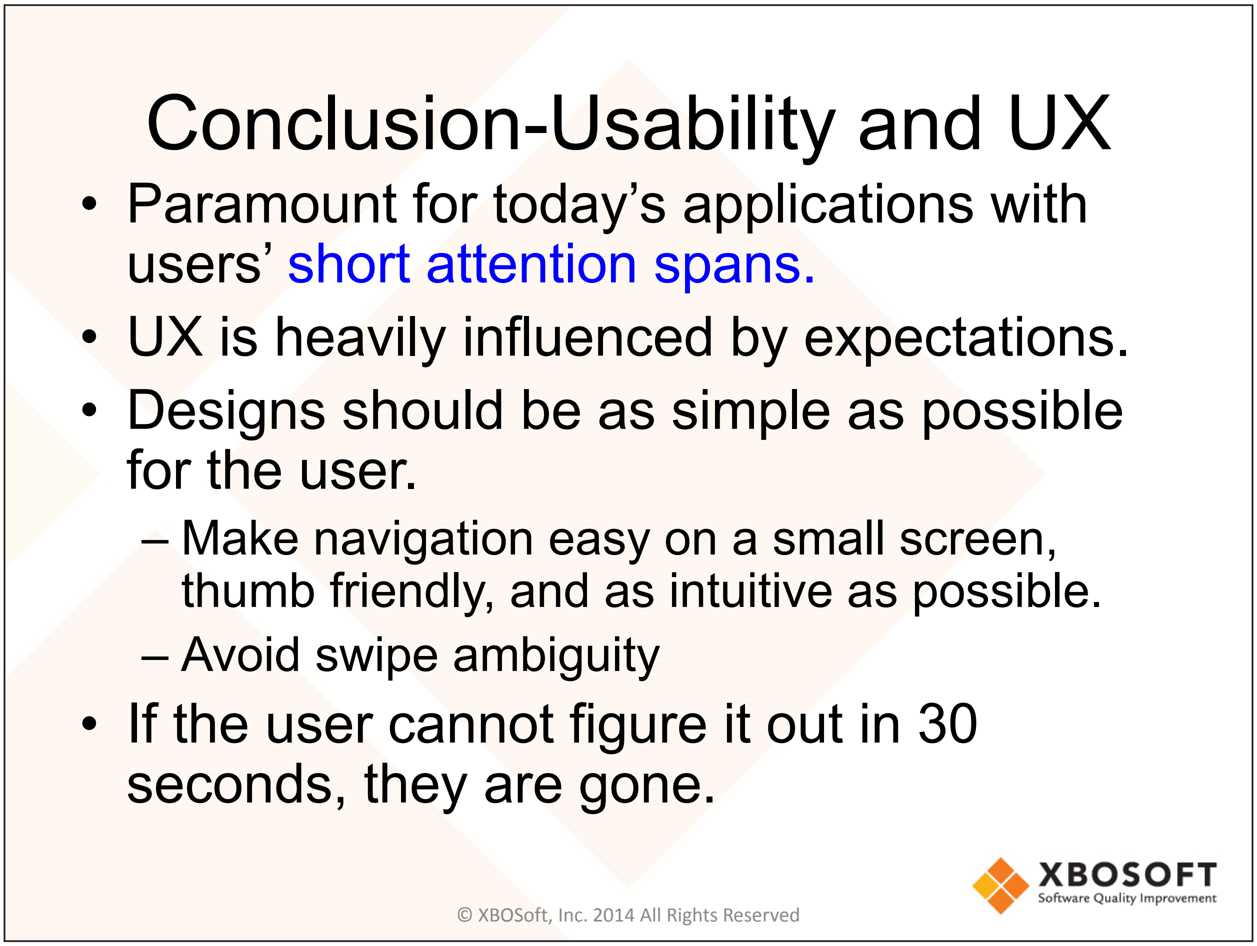
XBOSOFT
Software Quality Improvement



# Mobile Usability Design Summary Points

- Quick - Can you read the text without zooming or scrolling?
- Simple Navigation-Task Oriented
- Thumb Friendly
- Visibility-Colors
- Easy to Convert/Complete the task
- Contextual
- Integrate-Seamless with webapp
- You must **limit the number of features** to those that matter the most for the mobile users

**Design for the small screen. Don't make users struggle to click tiny areas that are much smaller than their fingers.**



# Conclusion-Usability and UX

- Paramount for today's applications with users' short attention spans.
- UX is heavily influenced by expectations.
- Designs should be as simple as possible for the user.
  - Make navigation easy on a small screen, thumb friendly, and as intuitive as possible.
  - Avoid swipe ambiguity
- If the user cannot figure it out in 30 seconds, they are gone.

# Conclusion (last one!!) Don’t Mobilize Everything

- Mobile-app versus your web-app - differentiate and discern access from each platform.
- Determine the features you really need and optimize screens for certain workflows rather than trying to do it all.
- Think about the tasks!!
- Better to have an half of an application that kicks _____ rather than a half ____ application.

# You Can Be Wrong → Usability Testing

Screen recorders

- Unobtrusive and does not get in the way of testing
- Can use the device as in any context or environment

Some issues

- Most don’t record user gestures, especially failed ones
- Limited recording times and capabilities
- Limited to recording mobile web sessions only
- Can’t use the test participant’s own device
- Can’t integrate with usability testing software such as Morae
- Can’t typically record facial expressions and verbal comments

# Usability Testing
# Using Screen Recorders

Some newer solutions

- UX Recorder
  - Records screen activity, captures some gestures, and records user voice and face but can only be used for mobile web apps.
- Magitest
  - Captures tap gestures only, user voice and face, and can record activity of other mobile native apps.

XBOSOFT
Software Quality Improvement

# ***Mobile UX***
# Life or Death
# Make or Break

XBOSOFT
Software Quality Improvement

04/06/2014

# Questions and Answers

*Please fill out an evaluation form and drop it in the collection basket located at the back of the room.*

Philip Lew
@philiplew
philip.lew@xbosoft.com

Some resources: http://www.xbosoft.com/knowledge_center/

www.xbosoft.com

XBOSOFT
Software Quality Improvement

# Thanks

Q&A

www.xbosoft.com
@xbosoft
408-350-0508

/xbosoft

/xbosoft

Philip Lew
@philiplew
philip.lew@xbosoft.com

White Papers: http://www.xbosoft.com/knowledge_center/

Blog: http://blog.xbosoft.com/

XBOSOFT
Software Quality Improvement

# Does DAD Know Best, Is it Better to do LeSS or Just be SAFe? Adapting Scaling Agile Practices into the Enterprise

**Aashish Vaidya**
aashish.vaidya@cambiahealth.com

## Abstract

Organizations, large and small, that are experimenting and succeeding in using agile practices at team level, face their next challenge in scaling these practices across the enterprise. This challenge can come from expanding pilot programs from handful of teams to more teams; or, it comes from agile teams working with non-agile parts of the organization. The process inevitably creates confusion as teams employing different methods interface with each other. Introduction of agile practices challenges existing structures and practices, and brings forth questions regarding traditional roles, responsibilities and expectations. Should an organization continue to hone team level practices or should it try to extend agile practices to other parts of the organization? Are there large scale practices that work in easing the interaction between agile teams, and other non-agile, semi-agile business units? While scaling, can an organization get too process heavy and risk losing the original intent of transitioning to agile practices?

There are primarily three scaling frameworks that try to address scaling agile practices: Disciplined Agile Delivery (DAD), Large-Scale Scrum (LeSS), and Scaled Agile Framework (SAFe). Each of these frameworks draws from variety of agile and lean practices. However, an organization's context matters most in deciding whether to embrace particular framework or only select practices to obtain desired results. In most cases, an organization has to make informed, pragmatic choices and experiment with various practices in order to address its specific needs without losing sight of why it embarked on an agile transition and transformation journey in the first place.

Cambia Health Solutions has for over three years rolled out Scrum and other agile practices across more than 40 development oriented teams. We also keep apprise of scaling models and newer practices to better understand our own implementation. For Cambia, scaling practices, articulated in scaling frameworks, such as enterprise-wide synchronized Sprints, multi-program Quarterly Release Planning, Scrum of Scrums and Communities of Practices, have been effective in organizing our work. Many of these practices can prove useful not only to larger enterprises like Cambia, but also to smaller ones, who are looking to improve and refine their adaption of agile practices.

## Biography

***Aashish Vaidya*** *is a Staff Consultant at Cambia Health Solutions, a not-for-profit, health solutions company. He works on large portfolio projects and serves as an internal Agile and Quality coach. He is a founding member of Cambia's Enterprise Transition Community, Agile Best Practices Exchange and other Communities of Practice. Aashish has over 20+ years of industry experience working in various leadership and management positions for companies such as Compaq, Intel, Kronos Incorporation. Aashish has been a presenter and a panelist for Technology Association of Oregon, AgilePDX, RoseCity SPIN and PNSQC. An agile practitioner for over 7 years, he holds CSP, CSM and SAFe SA certifications. He is a co-author of 2 articles on Agile Atlas on Scrum common practices and maintains an infrequent blog on Agile and Management topics at agilesutra.wordpress.com. Aashish has a Bachelor of Science degree in aerospace engineering from Texas A&M University.*

# 1. Introduction

Almost three years ago, the IT department at Cambia Health Solutions transitioned its Software Development Life Cycle (SDLC) from traditional waterfall methods to agile practices. Cambia Health Solutions is a not-for-profit total health solution company. The Cambia IT department is a shared services group, which provides software development, integration and other IT services to Cambia's two major divisions: Health Insurance Services (HIS) and Direct Health Solutions (DHS). Health Insurance Services is Cambia's mainline, regulated health insurance business. Whereas, DHS is comprised of various companies that provide affiliated healthcare solutions. Just like many health care companies, Cambia is facing rapidly changing market forces including meeting the challenges of ever increasing health care costs, and once in a generation reforms in the form of the Affordable Care Act (ACA). Cambia added two new corporate values of agility and innovation as a reminder of these challenges. Transitioning to agile practices was in a direct response to prepare Cambia for meeting market challenges and achieving its business objectives.

All of Cambia's development teams, as well as a few IT operational and other shared services teams, have adopted Scrum and Kanban for their SDLC about three years ago. Our experiences in leading transition efforts and team level practices were outlined in a 2012 PNSQC paper (Vaidya and others, 2012). Along with Scrum and Kanban team practices, we have employed various other practices that extend beyond teams and help us interact on larger, more complex projects. These practices also help agile teams interface with non-Agile units of the organization.

# 2. Agile Scaling Frameworks

At Cambia, when we changed our SDLC to agile approach, we adopted Scrum as our primary agile method. Many agile methods provide specific practices that inform team level practices. However, Agile methods generally do not provide specific guidance for wider organizational practices such as the front end process of software development such as business case approval, budgeting, project onboarding or the back end of interacting with partners and vendors, having the product ready for deployment to users, and project benefits realization.

For instance, Scrum is the best known and most used agile method in the industry. As defined by Scrum Alliance, Scrum specifies a minimal set of practices. It calls for three roles on a Scrum team – the Product Owner (PO), a ScrumMaster (SM) and a cross-functional team. It specifies each team to manage its work through four artifacts: a Product Backlog, a Sprint Backlog, a Product Increment and Definition of Done. In order to achieve its goals, Scrum specifies five activities for teams: Backlog Refinement, Sprint Planning, Daily Scrum, Sprint Reviews and Sprint Retrospectives (Agile Atlas). Beyond these roles, artifacts and activities, however, Scrum makes no specific recommendations in scaling beyond the team level. It expects an engaged leadership to provide broader vision and goals, and empower self-organizing agile teams. It expects practices to emerge naturally that will help organization continue its journey towards better agility.

Similarly, Kanban is even less specific in its required practices. In most Kanban implementation, generally two practices are specified. One is a visualization of workflow, generally through a Kanban board. Another practice is imposing a work-in-progress limit for every step in the workflow. David Anderson who articulated the Kanban method adds three other practices: measure and manage flow, make process policies explicit and use of models to recognize improvement opportunities (Anderson, 2010, Loc 619 of 4729). Just like Scrum, Kanban relies on self-organized teams and highly involved leadership in devising other practices that helps the adoption of Agile into the organization.

Three years back, Cambia's Enterprise Transition Community was tasked to help rolling out agile practices to the organization. Part of the transition effort involved not just addressing team level practices, but also aligning 40+ agile teams to work together with rest of the organization that still used traditional methods. At the time, our primary scaling framework reference model was Scaled Agile Framework (SAFe). Since then we have also referenced two other scaling frameworks – Disciplined Agile Delivery (DAD) and Large Scale Scrum (LeSS). All three of these frameworks try to address practices beyond the team level. These frameworks use combination of practices based on various Agile and Lean concepts. Many proponents claim that an organization must adopt a scaling framework in its entirety to reap the benefits. At Cambia, instead, we have taken a more pragmatic approach of adopting, blending, and modifying practices that suit our context, and help us meet our business objectives. We reference these scaling frameworks to understand our current implementation and examine whether adoption of additional practices will help us make continuous improvements and solve specific issues.

Before I outline Cambia's scaling practices beyond the team level, let us review these three scaling frameworks and their approaches to roles, processes, and other salient features. This will provide some context on practices we follow at Cambia as compared to those advocated by the scaling frameworks. To provide a frame of reference, I will also compare and contrast the key features against standard Scrum. Also, note that these frameworks are continually updating their practices and as such, the information below may have changed since the time of this writing.

## 2.1. Disciplined Agile Delivery

Disciplined Agile Delivery is developed by Scott Ambler and is an attempt to "fill in the process gaps that Scrum purposely ignores." DAD is a "hybrid approach which extends Scrum with proven strategies from Agile Modeling (AM), Extreme Programming (XP), Unified Process (UP), Kanban, Lean Software Development, Outside In Development (OID) and several other methods" (Ambler, 2013). DAD teams "focus on producing repeatable results, such as delivering high-quality software…[but] do not strive to follow repeatable processes" (Ambler, 2012).

### 2.1.1 Roles

The DAD framework starts with Product Owner and primary Team Members roles, just like Scrum. In addition, DAD introduces a role called Team Lead, which is akin to ScrumMaster in Scrum and an Architecture Owner role. There are other secondary roles like Specialist, Independent Tester, Domain Expert, Technical Expert and Integrator. These secondary roles are introduced to address scaling issues and can be deployed on a temporary basis. The Product Owner and primary Team Members roles are similar to Scrum (Disciplined Agile Delivery).

### 2.1.2 Practices

The DAD framework employs four distinct lifecycles and expects an organization to employ these lifecycles to suit its needs (Ambler, 2013).

The first lifecycle, called Agile/Basic, illustrated in Diagram 1, has three phases called the Inception, Construction and Transition.
The Inception phase is meant for "lightweight visioning activities to properly frame the project", meant to be brief for one or more short iterations where initial requirements and Release Plan are worked out. This may include initial modeling and architectural visioning. The Construction phase follows the contours of typical Scrum iterations. Similarly, the Transition phase could be one or more short iterations and is used for deploying the solution into production. Daily stand-ups, iteration planning, review and retrospective remain the same as Scrum. In DAD, a work item list is slightly different from a Scrum Product Backlog, since it includes not just requirements and defects, but also, "other non-functionality oriented work such as training, vacations, and assisting other teams" (Disciplined Agile Delivery).

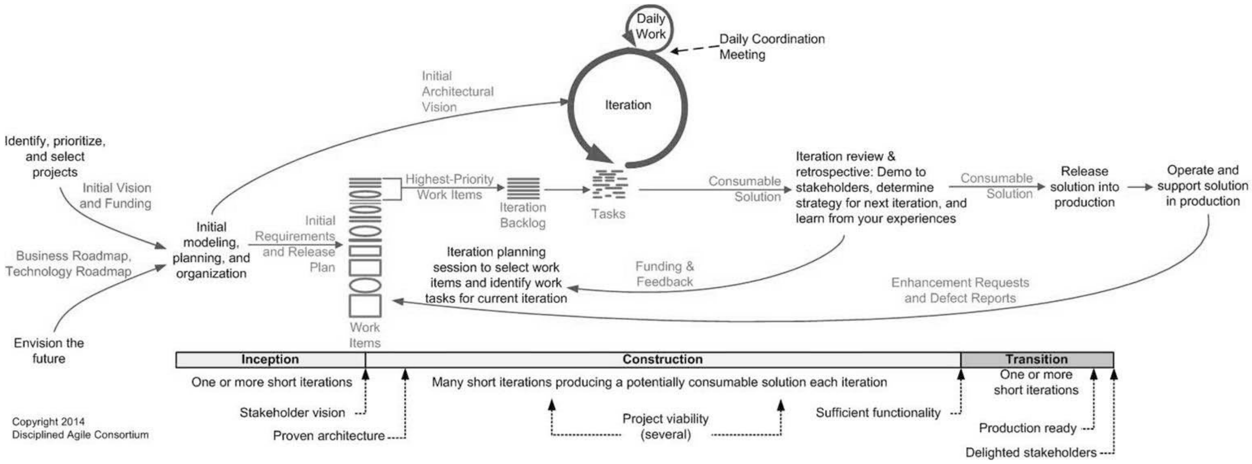

Daily Work
Daily Coordination Meeting
Initial Architectural Vision
Iteration
Identify, prioritize, and select projects
Initial Vision and Funding
Highest-Priority Work Items
Iteration Backlog
Tasks
Consumable Solution
Iteration review & retrospective: Demo to stakeholders, determine strategy for next iteration, and learn from your experiences
Consumable Solution
Release solution into production
Operate and support solution in production
Initial modeling, planning, and organization
Initial Requirements and Release Plan
Business Roadmap, Technology Roadmap
Iteration planning session to select work items and identify work tasks for current iteration
Funding & Feedback
Enhancement Requests and Defect Reports
Work Items
Envision the future
Inception
Construction
Transition
One or more short iterations
Many short iterations producing a potentially consumable solution each iteration
One or more short iterations
Stakeholder vision
Proven architecture
Project viability (several)
Sufficient functionality
Production ready
Delighted stakeholders
Copyright 2014 Disciplined Agile Consortium

**Diagram 1: Agile/Basic Lifecycle (Credit: Disciplined Agile Consortium)**

The second lifecycle, called Advanced/Lean, as depicted in Diagram 2, is closer to Kanban in its Construction phase. Just like the Agile/Basic lifecycle, the Inception phase is used to stock a work item pool. This work item pools are organized along business value, fixed delivery date, expedited or some other intangible categories. Planning, retrospectives, demos, stand-ups and other activities are performed as needed rather than following a scheduled cadence as followed in the Agile/Basic lifecycle. This includes deployment readiness during the Transition phase. Some upfront architectural modeling and visioning similar to Agile/Basic lifecycle is done during the Inception phase.

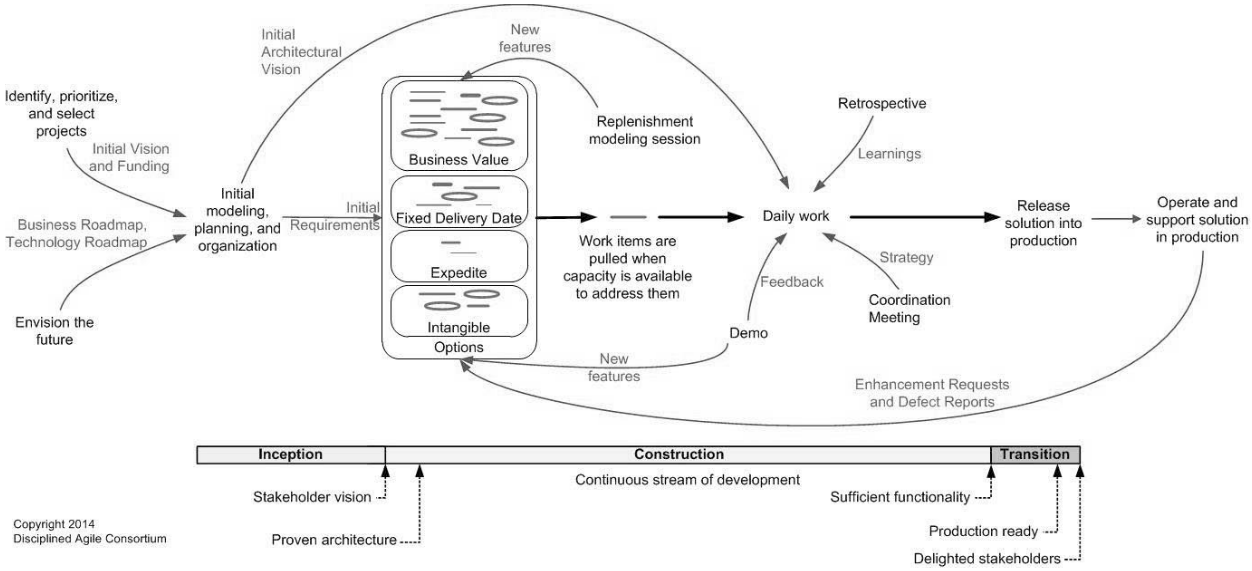

Initial Architectural Vision
New features
Identify, prioritize, and select projects
Initial Vision and Funding
Replenishment modeling session
Retrospective
Learnings
Business Value
Fixed Delivery Date
Expedite
Intangible Options
Initial modeling, planning, and organization
Initial Requirements
Business Roadmap, Technology Roadmap
Work items are pulled when capacity is available to address them
Daily work
Release solution into production
Operate and support solution in production
Feedback
Strategy
Coordination Meeting
Demo
Envision the future
New features
Enhancement Requests and Defect Reports
Inception
Construction
Transition
Continuous stream of development
Stakeholder vision
Proven architecture
Sufficient functionality
Production ready
Delighted stakeholders
Copyright 2014 Disciplined Agile Consortium

**Diagram 2: Advanced Lean Lifecycle (Credit: Disciplined Agile Consortium)**

The third lifecycle is called Continuous Delivery lifecycle, which is a “leaner” version of Advanced/Lean lifecycle, illustrated in Diagram 3, below. It does away with an explicit Inception phase and has very short Transition period. In this lifecycle, a “product is shopped into production or the marketplace on a very regular basis…as often as daily,” but weekly and monthly is common too (Disciplined Agile Delivery).

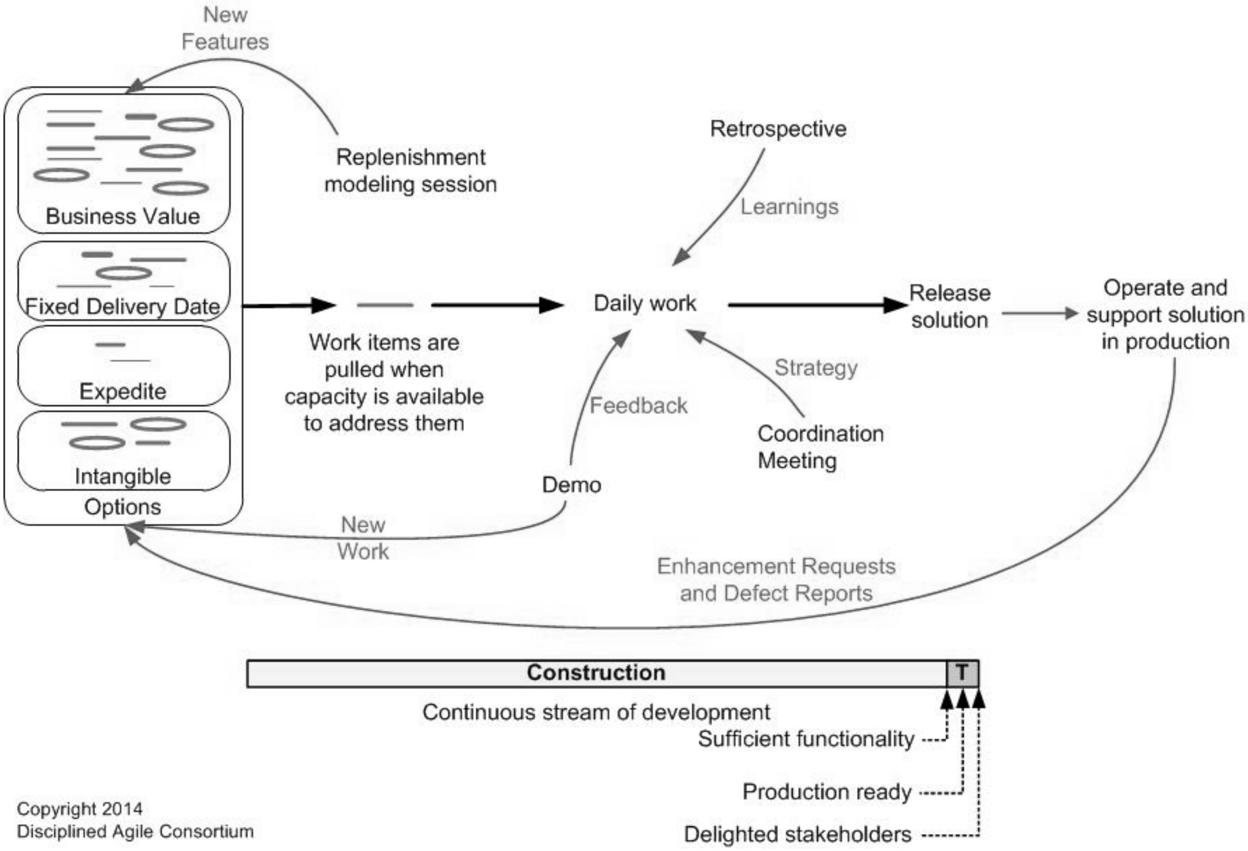


**Diagram 3: Continuous Delivery Lifecycle (Credit: Disciplined Agile Consortium)**

The last or the fourth lifecycle is called the Exploratory lifecycle, shown in Diagram 4, which is meant for “agile or lean teams that find themselves in startup or research situations where their stakeholders have an idea for a new product but they do not yet understand what is actually needed by their user base.” This lifecycle consists of six activities such as envisioning, prototyping, deploying, observe and measuring and based on the feedback canceling or productizing the idea (Disciplined Agile Delivery).

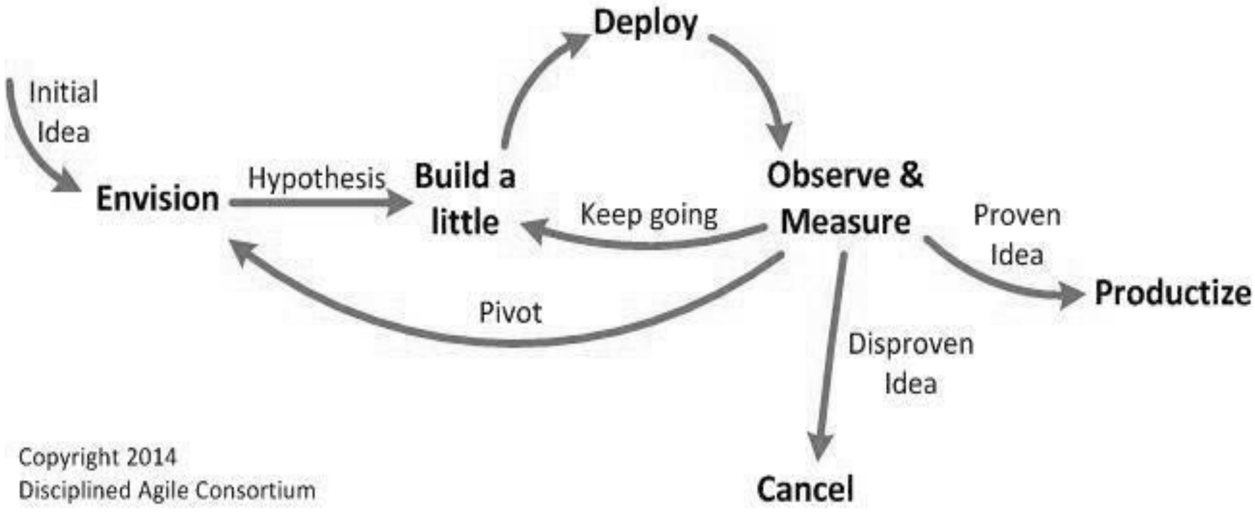


**Diagram 4: Exploratory Lifecycle (Credit: Disciplined Agile Consortium)**

## 2.2. Large Scale Scrum (LeSS)

Craig Larman and Bas Vodde articulated Large Scale Scrum “to apply Scrum to very large, multisite, and offshore product development.” They define LeSS to apply to a median implementation that covers, “around 800 people on one product at 5 sites, with about 15 million lines of source code” (Larman & Vodde, 2013). LeSS framework specifies organizational changes, which isn’t directly addressed in standard Scrum. LeSS specifies cross-functional, cross-component, end-to-end feature teams through the elimination of traditional team lead and project manager roles. In an article outlining a case study at J.P. Morgan, the feature teams each had a “blend of domain, technical and functional skills.” (Larman & Winn, 2014)

The LeSS framework recommends two frameworks, one that covers up to ten Scrum teams (called LeSS framework-1) and another that has more than ten teams (called LeSS framework-2). This suggests that framework-1 should be used for up to 100 people, since it also recommends a team size of up to ten team members. In framework-2, scaling is accomplished using sets of framework-1 groups.

### 2.2.1 Roles

In LeSS framework-1, a single Product Owner is common to all ten teams, whereas, in framework-2, an Area Product Owner role is introduced that covers a specific product area. An Area PO covers 3+ Scrum teams and specializes on one requirement area (Larman & Vodde, 2013). No other special roles are specified compared to standard Scrum.

### 2.2.2 Practices in Framework 1

Sprint Planning meeting changes in framework-1, illustrating in Diagram 5. For Sprint Planning, each of the Scrum teams sends two members per team (as opposed to entire Scrum team participating in standard Scrum) plus the one overall Product Owner to decide on which chunk of Product Backlog items to work on. Product Owner arbitrates between teams when a backlog item is in contention. Similarly, Sprint Review changes to a single meeting for all Scrum teams, but is limited to two team members per each Scrum team (Larman and Vodde, 2013).

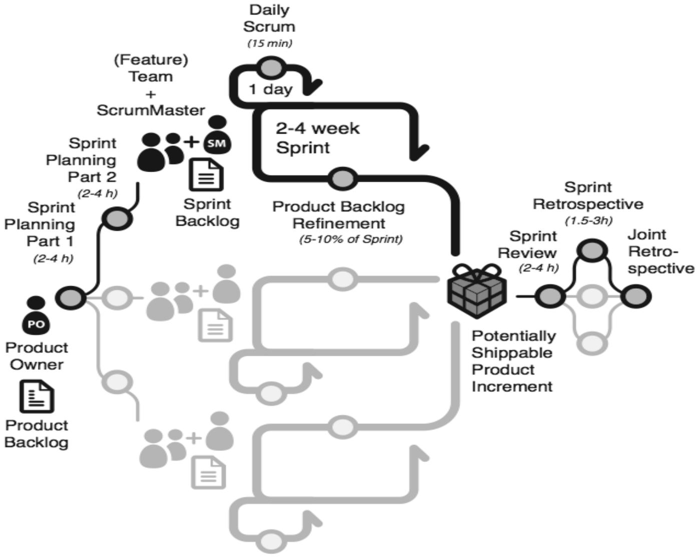


**Diagram 5: LeSS framework-1 (Larman and Vodde, 2013)**

In addition to these changes, LeSS introduces three more practices: Inter-team coordination meeting, a Joint Light Product Backlog Refinement and Joint Retrospective meeting. The Inter-team coordination meeting can be held frequently during the week and it can use various format, including an “Open Space, Town Hall Meeting, Multi-Team Daily Scrum or Scrum of Scrums, to increase information sharing and coordination.” The Joint Light Product Backlog Refinement has a maximum duration not to exceed 5% of the Sprint duration. It is also restricted to two team representatives. In this meeting, the team looks to refine product backlog items for upcoming Sprint. Lastly, a Joint Retrospective is added, which is attended by team Scrum Masters and one representative from each team. They jointly “identify and plan improvement experiments for the overall product or organization” (Larman and Vodde, 2013).

LeSS framework-1 also has an optional practice called In-Sprint Item Inspection where the teams “informally seek out early feedback from the PO or other stakeholders on finished Product Backlog items as soon as possible during the Sprint.”

### 2.2.3 Practices in Framework 2

All framework-1 meetings continue for each of the product area as outlined in framework-1 practices. In framework-2, as shown in Diagram 5, a new Pre-Sprint Product Team Meeting is introduced. This meeting is held prior to Area Sprint Planning meeting outlined in framework-2 practices, with all Area POs and the overall Product Owner, in order to focus on “product-level rather than area-level optimization”. (Larman and Vodde, 2013) In addition, an overall Sprint Review meeting at the product level in introduced which only focuses “on a subset of interest to the overall PO or to many Area POs.” Lastly, an Overall Sprint Retrospective meeting at the product level is introduced. This meeting “happens earlier in the subsequent Sprint, after area-level Joint Retrospectives.” (Larman and Vodde, 2013).

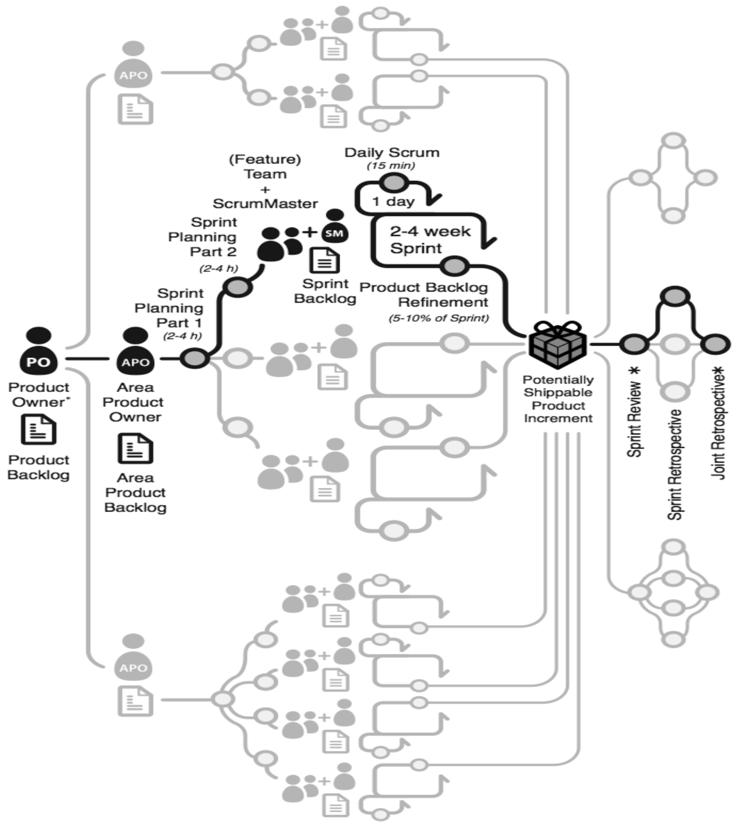


**Diagram 6: LeSS Framework-2 (Larman and Vodde, 2013)**

## 2.3. Scaled Agile Framework (SAFe)

The Scaled Agile Framework is created by Dean Leffingwell along with other collaborators, is illustrated in Diagram 7. SAFe framework articulates three levels of organization: Team, Program and Portfolio. Each level has its own activities and all levels are tied together. SAFe incorporates agile and lean practices at all three levels (Leffingwell, 2011, loc 1261-1315). SAFe provides team and program size patterns, which can then be used for scaling across larger organization. SAFe specifies standard Scrum team size consisting of five-nine team members. A program is defined as consisting of persistent five-twelve agile teams or 50-125 individuals that are dedicated to the program and capable of delivering business capability or value.

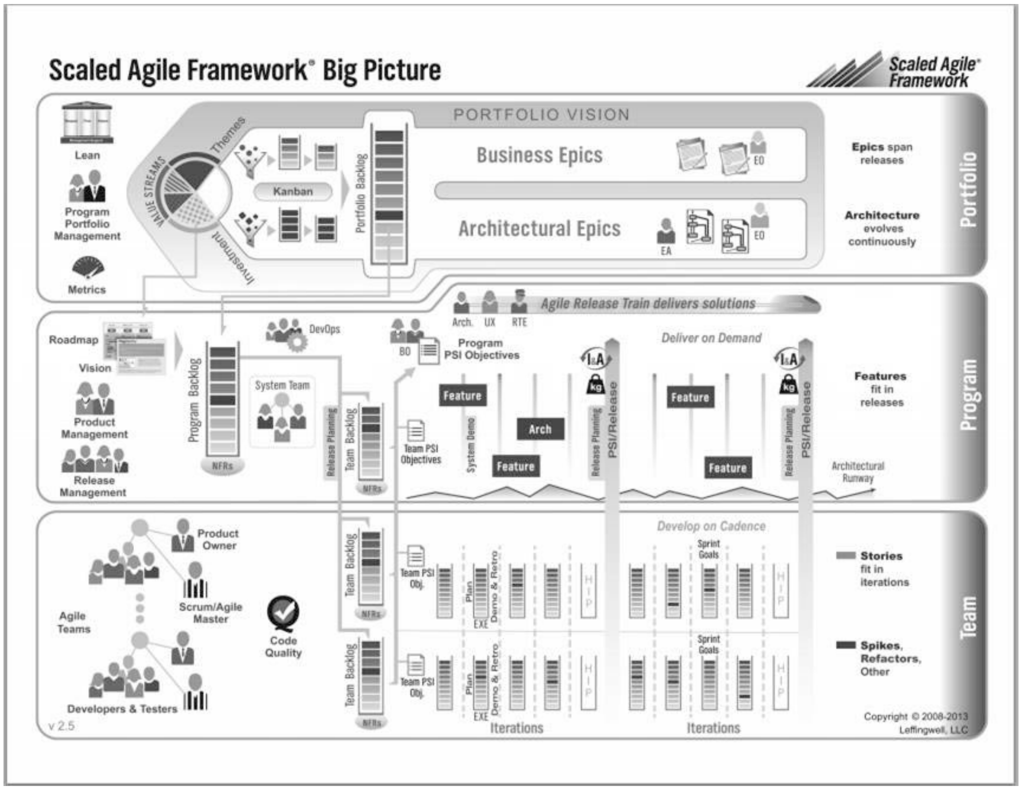


**Diagram 7: Scaled Agile Framework Big Picture (Scaled Agile Framework)**

### 2.3.1 Roles

At the team level, an agile team continues to resemble a typical Scrum team with some variations. A team has a ScrumMaster, which could “be part-time role for a team member (25-50%), or a single ScrumMaster may be shared across 2-3 teams” (Scaled Agile Framework). The team, referred to as ScrumXP, continues to have Product Owner and a team of five-nine team members, just like standard Scrum. A ScrumXP team can be a specialized component team and does not have to be broadly cross-functional or a feature team. The collection of ScrumXP teams coordinates with each other to develop and deliver cohesive end-user value. However, a ScrumXP team must have capability to design, build and test its own work.

At the Program level, several new roles and teams are created. A Product Manager role serves “as the ‘content authority’ for the release train and is responsible for defining the prioritizing the Program Backlog, and working with Product Owners to optimize Feature delivery” (Scaled Agile Framework). Product Manager directs the work of Product Owners at the team levels. A dedicated System Architect role is present to perform some up-front architecture and guide the emergent architecture for all program teams. Just as a Product Manager is a “chief Product Owner” for the program, a Release Train Engineer role serves as a “chief ScrumMaster”. The Release Train Engineer “facilitates program level processes and program execution, escalates impediments, manages risk, and helps drive program-level continuous improvement” (Scaled Agile Framework). A User Experience or UX designer provides “cross-program design guidance so as to provide a consistent user experience across the components and systems of the larger solution.”

SAFe also has additional program level teams, apart from the individual roles mentioned above. A Business Owner team comprising of three-five stakeholders have “the ultimate fiduciary, governance, efficacy and ROI responsibility for the value delivered by a specific release train” (Scaled Agile Framework). A Release Management Team (RMT) is generally manages across one or many products lines, the responsibility for scheduling, managing and governing of synchronized releases (Scaled Agile Framework). A DevOps team “provides tighter integration of development and operations” and maintains deployment readiness for the program. A System Team is “responsible for providing assistance in building and using the development environment infrastructure – including Continuous Integration, build environments, testing platforms and Test Automation frameworks – as well as integrating code from Agile Teams, performing end-to-end system testing, and demonstrating solutions to stakeholders at each iteration” (Scaled Agile Framework).

At Portfolio level, there is a Program Portfolio Management team that “represents the highest-level fiduciary (investment and return) and content authority (what gets built)….” This team comprises of “business managers and executives who understand the enterprise business strategy, technology, and financial constraints….” This team has the responsibility of providing portfolio vision, strategic and investment funding and overall portfolio governance (Scaled Agile Framework).

### 2.3.2 Practices

At the team level, SAFe specifies a blend of Scrum and Extreme Programming (XP) practices. The most significant departure from Scrum is that SAFe specifies practices surrounding code quality or agile software engineering practices mainly derived from XP. The code practices include Agile Architecture, Continuous Integration, Test-First, Code Refactoring, Pair Work, and Collective Code Ownership. The other significant change from Scrum is that SAFe does not expect teams will produce Potentially Shippable Increment (PSI) every Sprint, but rather over a quarterly cadence. At the program level provides features, which the teams deconstruct and size to fit into iterations.

SAFe employs an Agile Release Train (ART), at the program level, in order to develop large-scale systems. An ART comprises of four two-week iterations followed by a three-week HIP (Hardening, Innovation and Planning) iteration. Teams are dedicated to the ART and they all synchronously develop and release a PSI on the same quarterly cadence. A system team generally operates iteration behind the individual ART teams. The system team integrates code from various ART teams and performs “end-to-end and system performance testing” of features, either manually or through test automation. The system team also helps stage a “system Sprint demo”, where teams showcase the whole system to the stakeholders. At the end of every quarter, during the HIP iteration, all teams within the ART plan the next PSI during an all-hands Release Planning meeting. During execution of a PSI, the ART also organizes a twice a week Scrum of Scrum for all the dedicated ART teams.

At the Portfolio level, SAFe introduces concepts of investment themes and values streams that align the ARTs at the program level. A value stream is defined as a “long-lived series of system definition, development and deployment process steps used to build and deploy systems that provide a continuous flow of value to the business, customer or end user” (Scaled Agile Framework). In SAFe parlance, a value stream is realized through an Agile Release Train at the program level. Investment themes “reflect how the portfolio allocates budget to the release trains that implement the portfolio business strategy” (Scaled Agile Framework). These themes in turn act as a funnel, seeding a portfolio backlog with business or technical epics. There are two types of Epics in SAFe – Business and Architectural. Business Epics are “large, typically cross-cutting customer-facing initiatives that encapsulate the new development necessary to realize certain business benefits” (Scaled Agile Framework). And Architectural

Epics are “cross-cutting technology initiatives that are necessary to evolve portfolio solutions to support current and future business needs” (Scaled Agile Framework).

# 3. Program and Portfolio Level Practices at Cambia

Agile methods such as Scrum, Extreme Programming (XP), Kanban, offer practices that generally start with the team. Many of the Agile methods expect that through self-organization, truly cross-functional, cross-component teams will be able to delivery business value quickly on a cadence, with high quality. They expect these teams to respond to changes and work at a sustainable pace over a long duration. So, for companies that have transitioned to agile practices for their SDLC but still largely employ traditional PDLC and portfolio management techniques, these methods don’t provide much concrete guidance. In many smaller organizations, where formal structures and processes do not exist, or where structures are less hierarchical, adoption of agile practices can happen quickly and with relative ease. However, in larger organizations where there are Enterprise Program Management Office or Portfolio Management structures, introduction of practices at team levels leads to inevitable issues with existing norms, and processes.

Jim Highsmith, writing for the Agile Alliance, anticipated these culture issues at the initial adoption of the Agile Manifesto over a decade ago:

> *But while the Manifesto provides some specific ideas, there is a deeper theme that….people who held a set of compatible values, a set of values based on trust and respect for each other and promoting organizational models based on people, collaboration, and building the types of organizational communities in which we would want to work. At the core, I believe Agile Methodologists are really about "mushy" stuff about delivering good products to customers by operating in an environment that does more than talk about "people as our most important asset" but actually "acts" as if people were the most important, and lose the word "asset". So in the final analysis, the meteoric rise of interest in and sometimes tremendous criticism of Agile Methodologies is about the mushy stuff of values and culture (Highsmith, 2001).*

For transitioning or transforming organizations such as Cambia, Agile practices at team levels inevitably leads to larger questions about organizational design, command-and-control hierarchical management, HR and management practices and overall company culture. The three scaling frameworks described above provide approaches that attempt to address some of the issues that an organization faces and offer solutions to address these gaps. However, each framework provides some benefits, but they have shortcomings as well.

For example, DAD creates four distinct lifecycles, each of which an organization can adapt to fit its context. However, it also specifies an overly complicated work items pool, which an organization can address in much simpler ways. Further, DAD introduces several temporary secondary roles, which can hamper agile teams from becoming cohesive, self-organizing units capable of producing end-to-end features.

LeSS starts where Scrum leaves off when it comes to scaling agile practices in large organization. However, in the process, it makes recommendations that are problematic, like having a single Product Owner for up to ten teams. For a transitional organization, the business units are generally not accustomed to interacting with software development from the inception of the project to production delivery. With a Product Owner responsible for so many teams, in turn shortchanges the agile teams on the much-needed interaction with their business users and partners on a regular basis, throughout the project cycle.

Similarly, SAFe organizes its practices into three levels: team, program and portfolio, which is quite useful for larger organization. At a team level, it embraces certain XP practices, which standard Scrum does not. However, the framework has myriad of issues, including being overtly process heavy.

At Cambia, we have not embraced any particular scaling framework on a wholesale basis. None of the scaling frameworks completely aligns with our business needs. Instead, we have chosen few practices to help meet our needs. We also look for opportunities to refine and experiment with new approaches, make continuous adjustments as we obtain more information about the effects of these changes on the organization. This has allowed us to realize incremental benefits without putting the organization through excessive change management churn.

At a high level, Diagram 8 shows Cambia's approach to scaled Agile and program level organization.

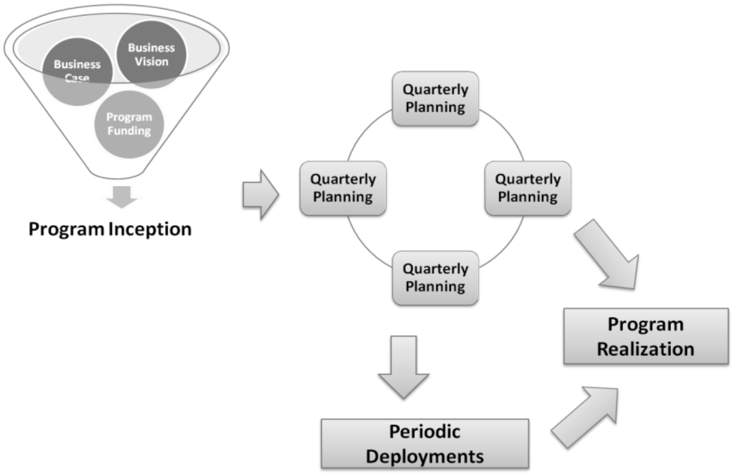


**Diagram 8: High Level View of Cambia's Scaled Agile Approach**

Multiple projects and multiple agile teams stay aligned and coordinated with the use of synchronized quarterly planning and Sprint schedules (see Diagram 9 for visual representation of the process).

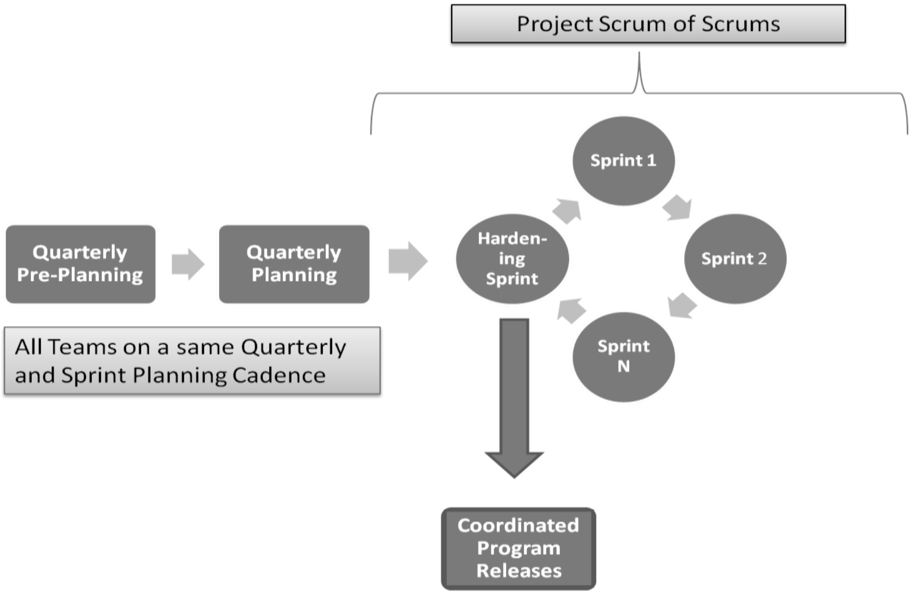


**Diagram 9: Cambia Synchronized Quarterly and Sprint Process for Program and Team Alignment**

Below is a discussion of some of these practices and how they are currently practiced at Cambia. Wherever, appropriate, I will highlight practices that the scaling frameworks specify as well.

## 3.1. Synchronized Enterprise Sprint Schedule

With the change in SDLC to agile approach, we broke away from traditional IT functional silos and re-organized into cross-functional agile teams, with representation from at least development, testing and analysis functions. However, most of the teams are not cross-feature oriented, neither are they dedicated to a single portfolio project. The predominance of components teams over features teams, also means that most of our teams have dependencies and need a way for them to stay coordinated and synchronized. Using an enterprise-wide synchronized Sprint schedule and Sprint duration is one good way to address this need. Since our transition, our Sprint duration at three weeks for all teams.

SAFe and LeSS recommend synchronized sprint as a scaling pattern. Mike Cohn also mentions synchronized sprint as an effective way to coordinate and synchronize several teams (Cohn, 2009, 343-345). Although Cohn does not stipulate that, each team needs to have the same Sprint duration.

At Cambia, synchronized enterprise sprints have helped team stay coordinated on work that requires multiple teams. Since all teams are in Sprint Planning on the same day, the Product Owners are able to coordinate dependencies, and if necessary, re-order their respective Sprint Backlogs. In addition, as Cambia does not deploy specialized project or program level Systems team as SAFe recommends, synchronized sprint schedule helps in reducing risks of untested or unintegrated code for longer periods. This in turn helps reduce hardening time before production deployment.

Since Cambia has offices in four states, many agile teams are distributed over multiple locations, having Synchronized enterprise Sprint schedule also helps in planning and budgeting for occasional co-location of these distributed teams for face-to-face Sprint planning or other major planning events. This has led to better planning sessions, and in many cases it has helped in making distributed teams more cohesive.

## 3.2. Scrum of Scrum Meetings

Synchronized Sprints allows Cambia agile teams to coordinate and synchronize their work during sprint planning. However, we also needed a way to stay aligned during the Sprints as well track and monitor project progress. In order to address this issue, we use Scrum of Scrum meetings. This practice is recommended in SAFe as well as LeSS. Cambia utilizes various types of Scrum of Scrums depending on the need of the teams and projects:

- **Project Scrum of Scrums** – Each portfolio project generally runs a Project Scrum of Scrum. This meeting invites representatives from Agile teams that have work for that particular project. Build and deployment engineers, operations and other business units are also represented on this meeting.
- **Testing-Specific Project Scrum of Scrums**: Many projects also organize testing specific Scrum of Scrums. During this meeting, testing concerns that are not critical enough to be raised during regular Scrum of Scrums, but are important to Agile teams and to User Acceptance Testing (UAT) group, can be discussed. It also serves as a way for teams to coordinate their system integration and testing work. Generally, a team member with QA expertise from each of the Agile team attends this meeting, along with UAT representatives and test environment coordinators. This meeting also helps in coordinating integration and testing work with vendors or external partners.
- **Organization Scrum of Scrum** – This meeting is usually organized for “like” teams to share any common concerns. For example, at Cambia, several Agile teams work on Electronic Data Interchange (EDI) processes. These EDI teams meet to discuss common issues, concerns, and development approaches for their domain.

The frequency of these meetings varies. Typically, each project Scrum of Scrums meets at least twice a week. Project Scrum of Scrums has improved interaction between the agile teams, operations teams and other business units. With the use of project dashboards, most projects also use the Scrum of Scrums meeting to monitor the progress of feature development, surface any issues regarding integration of code, deployment readiness, vendor and partner coordination and other needs arising out of inter-team interaction, on a set cadence. Other major benefit of various Scrum of Scrums is a marked reduction of effort on the part of the Project Management Office (PMO) in collecting project status from each individual teams (as it used to happen before), as well as less overhead and distractions for the agile teams. Similarly, both the Project and Testing-specific Scrum of Scrums has allowed to better alignment on systems integration and testing between agile teams. These meetings have provided a way for the UAT testers to monitor the progress of feature development, engage in early testing and refine their test plans for the formal user acceptance testing.

## 3.3. Hardening Sprints

As, many Cambia teams are component teams, there is not enough Agile maturity or fluency to produce potentially shippable code at the end of every Sprint. Moreover many Product Owners and teams are not adept at deconstructing features, and stories into vertical, thin, business value oriented, requirement slices. TO address this issue, we use hardening Sprints to account for any remaining work that could not be completed during the Sprints, to make potentially shippable code in product-ready solution.

Hardening Sprint before the end of quarterly Release process is a feature of SAFe framework. Hardening Sprint at Cambia typically comprised of systems performance testing, formal user acceptance tests and other end-to-end testing including final integration work with external vendors and partners. The number of Hardening Sprints is largely dependent on the project size, complexity and number of releases each project has planned to complete its delivery. For a typical project which generally plans quarterly releases, a Sprint (and sometimes even less time) is scheduled for hardening. Projects that have longer release cycles and those that require complicated vendor coordination, or require integration with outside entities usually plan higher number of Hardening Sprints. By shifting systems integration work as part of feature complete definition, we are find many teams have to reserve less time for hardening.

## 3.4. Multi-team, Multi-Project Quarterly Planning Meeting

As Scrum only specifies two levels of planning – Sprint and Daily Scrums, use of Scrum alone was not sufficient for Cambia to understand the overall progress of our portfolio implementation during the course of the year. Due to various constraints including meeting regulatory compliance, for many projects and teams just-in-time planning is not always sufficient. For many portfolio projects that have mandated deadlines, Cambia also needed a way to predict our work demand and understand our resource capacity.

Over two years ago, Cambia started organizing Quarterly Planning events to start addressing some of these needs. The initial Quarterly Planning meeting was attended by a handful of teams who were contributing to a federally mandated compliance project called ICD-10 (WHO approved 10th revision of the International Classification of Diseases). Centers for Medicare and Medicaid (CMS) at that time had mandated all health care organizations in the US to implement ICD-10 by October 2012 (since then the deadline has moved twice). Every quarter since, Cambia has planned a multi-team, multi-project, quarterly planning meeting to align our work over the mid-term horizon, and to get insight into potential deadline and resource contentions.

This practice is similar to LeSS recommended framework-2 Pre-Sprint Product Backlog meeting, in intent. However, it is differs significantly. It is perhaps closer to SAFe specified Release Planning meeting, with couple of major differences. First, the meeting is not an all teams, all-hands Release Planning meeting as SAFe specifies it. Generally, two to three representatives from each team (Scrum Master, Product Owner and a key Team Member) attend this meeting. Apart from team level participation, project managers, select representatives from IT operations and members from key business units, also attend this meeting. The second difference is that as Cambia teams are not organized along SAFe-like Value Streams, we do not perform a rigorous planning of a Potentially Shippable Increment (PSI) or planning of Agile Release Trains.

For the Quarterly Planning meetings, the teams define their quarterly team backlog with project specific deliverables, non-project backlog items in form of enhancements, product support and other maintenance work. Many teams also set aside team capacity for training or other specific continuous improvement items. Although, many teams and POs find it challenging to maintain a healthy balance between all the competing project priorities and ongoing improvements.

The general format of this meeting is:

1) At team level, team members refine their quarterly backlog of known work with the PO.
2) Team representative attend the quarterly planning meeting. During the first part of the meeting, teams meet, discuss, and coordinate work. This may lead to re-ordering of their quarterly backlogs including identification of additional work, or coordinating systems integration and testing.
3) Teams identify questions, impediments or issues affecting their teams' ability in committing to an achievable quarterly plan. This includes planning for any Hardening Sprints for project deliverables.
4) Teams report out their confidence level.
5) Projects collate information from teams' backlogs and report out their confidence at a project level.

Quarterly Planning meeting has allowed teams and projects to understand potential gaps in getting feature and story definitions completed, recognize potential schedule risks, as well as recognize resourcing issues. The Quarterly Planning meeting has also provided us a way to perform backlog leveling between certain under- and over-allocated teams. The meeting provides insight for the PMO team to work with Product Owners in leveling and maintaining desired portfolio proportions between strategic and maintenance work allocation. However, it should be noted that based on our current maturity level, backlog and portfolio leveling is successful only on a limited basis.

## 3.5. Communities of Practice

Communities of Practice (CoP) are another important scaling practice that is widely used at Cambia. Communities of Practice are generally "a group of people who share a craft or profession." A Center of Excellence (CoE) refers to a "team, a shared facility or an entity that provides leadership, evangelization, best practices, research, support and/or training for a focus area. At Cambia, CoPs are generally a combination of Communities of Practice and Centers of Excellence. They are called Best Practice Exchanges (BPEs). Cambia BPE normally doesn't have a dedicated team that provides leadership and determines best practices. Instead, a volunteer core team generally forms as the nucleus of a CoP. The volunteer core team member generally dedicates about 10-20% of her time towards CoP activity. This core team engages in activities typical of a CoE, as well as helps with organizing learning and knowledge sharing events.

Two of the more active CoPs at Cambia are Agile Best Practice Exchange and Software Quality Best Practice Exchange. Last year, Agile Best Practice Exchange developed twenty best practice primers on agile practices. It also organized 36 Agile Overview Training, Agile Online Spotlight sessions and User Story writing workshops. It also organized two internal Open Space events. Within the Agile BPE, there is also an agile coaching and training community that provides coaching and mentoring help to teams and projects.

Similarly, Software Quality BPE organizes quarterly town halls, weekly online discussion events, bi-weekly quality practices meetings, QA certification learning forums, and practices regarding test automation.

Apart from these CoPs, there are other BPEs such as Secure Application Coding, Architecture, and Software Development. The activity level of each of these BPEs ebbs and flows, depending on the engagement of the core team and the community at-large as well as delivery pressures.

As we know, the Product Owners represent the "entrepreneurial" aspect of the company, whereas, role-based and function-based CoPs represent the craft or profession. Product Owners generally focus on meeting project delivery and time to market considerations. Cambia BPEs serve as a counter-balance by sharing and improving craft-based sustainable practices. BPEs work to enhance skills in team members which can continue to provide sufficient fuel to deliver high quality working software, on time, perpetually into the future. Both of these functions are needed and complement each other. LeSS explicitly calls for the use of CoPs. SAFe doesn't directly specify the needs of CoPs. However, many Agilists recommend this practice including Mike Cohn (Cohn, 2009, 347-352). Henrik Kniberg documented a similar structure to Cambia comprising of Agile teams and CoPs at Spotify. At Spotify, agile teams are called Squads, and CoPs are called Guilds (Kniberg and Ivarsson, 2012).

### 3.6. Benefits

Many of the team and enterprise level practices have led to significant improvement in our delivery and quality practices. On the delivery front, our agile teams have achieved steady flow of business value. For example, vast majority of our teams generally have a 90+ percentage of commit versus accepted story ratio. Through the Quarterly Planning meetings, we have achieved good visibility one quarter out on gaps in our planning, risks, issues and capacity needs. The practices of last three years have led to significant improvement in our overall agility and adaptability. For example, in 2013, during the first year of Affordable Care Act implementation, Cambia was prepared with viable solution for each of the four state-based health exchanges that we operate in (Oregon, Washington, Utah and Idaho). This included working with challenging partners such as Federal Facilitated Marketplace and Cover Oregon, which had significant issues in their implementation and did not provide timely specifications and requirements to insurance carrier or engaged in any meaningful integration testing. Another federally mandated ICD-10 project has now gone through two delays. Largely due to the implementation of Scrum, and enterprise level scaling practices, Cambia has achieved a level of agility to meet these types of changing market needs. In the past, these type of changes introduced significant stress in the organization.

Similarly, we have managed to improve our quality practices, with performing systems integration and user acceptance testing much earlier in the project cycle. Vast majority of the projects at any given time manage to keep their defects backlog to under double digits. This is also a significant improvement, these backlogs used to be significantly higher, in the tens of defects, d. Another instance of improvement is the number of change requests (CRs) we introduce into production. For instance, in the first seven months of 2014, we deployed 466 CRs to production. Following these CRs, there were 23 CRs that "caused harm" (production incidents attributed and root caused to a specific change request). Over the same period in 2013, we had 22 CRs that caused harm with 365 CRs deployed. Thus, we introduced 27% more changes into production, while maintaining the number of production incidents. It should be noted that this improvements are correlational as opposed to causal, as we cannot attribute them to any specific practice improvement.

## 4. Challenges

There continue to be significant challenges for us in our agile transition and transformation process. To highlight a few:

- **Availability of a full-time Product Owner role on each of the agile teams**: The need for a Product Owner is a crucial need that continues to be a gap on some teams. Many agile teams represent several business areas, and many Product Owners are only available part-time to them as they perform other non-product management duties as well. In many instances, a Product Owner is either not empowered or is not comfortable representing other business areas. In other instances, even if a Product Owner is empowered to represent various business units, but they do not have ready access to subject matter experts (SMEs) in other business units.
- **Organizing work through traditional projects means work allocation is not evenly distributed on agile teams:** We organize work and budget using traditional projects through a yearly cycle. So depending on the type of work needed for any giving year, many agile teams get capacity constrained. Adding capacity to existing teams or spinning up new teams requires lead-time, even if budget is available. The practice of adding or changing team members to existing team is not always effective as it leads to team reforming and storming. This tends to slow teams down exactly when they need to be more productive.
- **Absence of true feature teams and value streams:** Most Cambia teams are organized around technology stacks and are only "partial" feature teams. Despite the addition of enterprise-wide

practices, Cambia agile teams require a fair amount of coordination, alignment and synchronization. Lack of feature teams also means costly handoffs and heavier reliance on processes. We have achieved a fair amount of stability by dedicating team members to a team. However, we don't have the next level of alignment of having dedicated teams to a program or value streams, as both LeSS and SAFe specify.

- **Slow uptick of adding agile engineering practices:** A fair amount of early efforts were concentrated on booting up agile teams, acclimating team members and affiliated team members on agile methods such as Scrum or Kanban. However, penetration of agile engineering practices such as Continuous Integration, Continuous Delivery, robust automated testing regiment to decrease reliance on Hardening Sprints, development in business oriented vertical stacks, have not been uniform. The good news is that through the use of CoPs, many teams are getting enabled and improving on these practices. In many cases, shoring up development practices and reducing existing technical debt are trumped due to time to market considerations, especially meeting mandated compliance needs.
- **Onboarding of strategic projects:** As we go through annual budget cycles, many projects get funded at the beginning of the calendar year. Thus, the first quarter becomes a significant strain on the agile teams as well as the entire organization as many projects require onboarding work. However, this year, Cambia's PMO has been staging and staggering the onboarding of projects, which has led to some relief in this area.

# 5. Planned Practices

In our agile transition and transformation journey, we have noticed that there is a significant gap between the approval of a business case, which has a very high-level program deliverables and the type of information needed at the agile team levels. To bridge this gap, we are engaging in a two-tier planning process: at the program level and at the team level. As much of our business capability and requirements flow through programs, we have started maintaining work in features, a coarser grain of functionality, into program backlogs. These backlogs expressed in the form of features are shared with teams to be deconstructed into Sprint-able stories. Just as the team Product Backlogs is refined every quarter for the Quarterly Planning event, we are doing the same of refining features on a quarterly basis. We are hoping this will start laying the groundwork on adding the practice of specifying a PSI on a quarterly basis and launching SAFe type ARTs. We think this will be a good improvement over the current practice of large, all-encompassing product solutions, rather than defining and deploying more manageable, but useable chunks of the product backlog, early.

At Cambia, it is possible to have 20-25 projects onboarding or in delivery phases, simultaneously. We recognize that we have a complex process of overlaying these projects on 40+ agile teams. We know through various feedback mechanisms that teams find it overwhelming to deal with multiple projects (for some teams it can be as high as 4-6 projects), and juggle with enhancements, maintenance requests and drawdown of technical debt. One potential solution is to launch a SAFe value stream and dedicate teams to a single program or project. We have identified couple of persistent value streams that we expect to exist and funded on a multi-year basis. This SAFe pilot will allow us to learn and understand practices that may be suitable for our context. The pilot may also provide an impetus to collapse our program level into a handful of persistent value streams, rather than the current practice of launching multiple, traditional projects on an annual basis. We expect this may provide us an impetus to start moving away from projects and move towards enabling product and business capability based development approach.

# 6. Conclusion

Each of the three Agile scaling frameworks discussed here have their pluses and minuses. Each get some things right and each fall short on some of the agile values and principles. There has been a fair amount of discussion and consternation in the agile community on best way to scale agile practices into large organizations. These discussions usually tend to break into two camps. One camp maintains that organizations must accept transformative (and sometimes radical) changes to its organizational structure to achieve the true goals of achieving true agility.  This means changes to organizational design, to HR practices, moving away from Taylorian Scientific Management approach, away from command-and-control decision-making, and away from traditional project management techniques.  The other camp tends to advocate a more gradual approach by introducing transitory practices, and realizing tangible increment benefits, which then in turn can generate much needed transformative changes.

However, Ron Jeffries (and many other Agilists) through his criticism of SAFe warns that these flawed scaling frameworks might actually end up doing more harm than good, as it may give an organization a false sense of security.  A sense that by implementing these scaling practices, they have truly transformed the way they do business, but they may never achieve true agility, hyper-productivity, and a more innovative and truly engaged work force (Jeffries, 2014).  This is a valid criticism and it represents a true risk for an organization, which is seeking to obtain a competitive edge in meeting the challenges of rapidly changing market conditions.

At Cambia, we continue to experiment with practices that help us meet our corporate goals and approach the level of agile fluency required by the organization.  We need to continue to hone our agile transition journey, continue to use practices from existing agile methods as well look to scaling models for practices that can truly provide a competitive and innovative edge.  Just like any other organization, Cambia will have to maintain focus on our original intent of adaption of agile practices so that we can thrive as an organization in a post-health reform era, maintain our not-for-profit ethos, and provide affordable and transformative health care services to our members.

# References:

Vaidya, Aashish, Catherine Row and Mark Jackson, 2012, *On the Way to Meeting a Mandate: Transitioning to Large Scale Agile, PNSQC.org, http://www.uploads.pnsqc.org/2012/papers/t-50_Vaidya_paper.pdf (accessed May 22, 2014).*

Agile Atlas, Core Scrum, http://agileatlas.org/atlas/scrum (accessed July 22, 2014).

Anderson, David J., 2010, *Kanban: Successful Evolutionary Change for Your Technology Business*, *Kindle edition*, Blue Hole Press.

Ambler, Scott, 2013. "Going Beyond Scrum Disciplined Agile Delivery", http://disciplinedagileconsortium.org/Resources/Documents/BeyondScrum.pdf (accessed June 25, 2014).

Ambler, Scott*, 2012, "*Repeatable results over repeatable processes*,"* *http://disciplinedagiledelivery.wordpress.com/2012/05/17/repeatable-results-over-repeatable-processes-2/ (accessed May 27, 2014)*

Disciplined Agile Delivery, "Roles on DAD Teams", http://wp.me/P1ODzT-sc (accessed June 25, 2014).

Larman, Craig, and Bas Vodde. 2013. "Large Scale Scrum – More with LeSS." *Entry posted December 30, http://agileatlas.org/articles/item/large-scale-scrum-more-with-less, (accessed June 10, 2014)*

Larman, Craig, and Matt Winn*, 2014.* "Large Scale Scrum (LeSS) @ J.P. Morgan"*. Entry posted April 14. http://www.infoq.com/articles/large-scale-scrum-jomorgan, (accessed June 10, 2014)*

Leffingwell, Dean, 2011, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, Kindle edition,* Addison-Wesley Professional.

Scaled Agile Framework. Various Web Pages. http://scaledagileframework.com/ (accessed June 25, 2014). (Note: To reference additional information on quotes and paraphrasing that appear in the paper, click the various icons on the landing page of the site).

Highsmith, Jim. 2001. "History: The Agile Manifesto". *http://agilemanifesto.org/history.html (accessed June 25, 2014)*

Cohn, Mike, 2009, *Succeeding with Agile: Software Development Using Scrum,* Addison Wesley Professional.

Kniberg, Henrik and Anders Ivarsson. "Scaling Agile @ Spotify with Tribes, Squads, Chapters and Guilds", Entry posted November 12. http://blog.crisp.se/2012/11/14/henrikkniberg/scaling-agile-at-spotify (accessed June 25, 2014).

Jeffries, Ron, 2014. "SAFe – Good But Not Good Enough". *Entry posted February 27.* http://xprogramming.com/articles/safe-good-but-not-good-enough/ , (accessed July 11, 2014)

*Additional Reference:*

Lines, Mark. 2012. "Introduction to DAD," *http://disciplinedagiledelivery.com,* entry posted November 11, http://wp.me/P1ODzT-dx (accessed June 25, 2014)

# Quality Engineering For DevOps Customers

**Dwayne Thomas**

thomas.dwayne@gmail.com

## Abstract

The TV software company (the company) of this experience report creates interactive software that overlay TV program feeds. The company's DevOps professionals mainly perform software deployment activities and IT operations. (The combination of these two activities form the acronym DevOps.) DevOps help to more frequently scale software solutions to the business market. DevOps professionals also perform production testing and scale interactive software experiences to TV viewers. Testers at the company recently started supporting the company DevOps team more actively, since the DevOps team now consumes the most of the products of testing and development activities. The company quality engineering (testing) team is faced with a question: how do testers improve software quality for DevOps customers?

Recently, many learning opportunities for DevOps professionals have arisen. Due to the 24/7 nature of the TV industry, the company DevOps professionals are on call to monitor the software deployments. The question of how to improve software quality engineering for DevOps customers is fertile for exploration because the DevOps role has only formally existed for 5 years (Wikipedia, 2013). As Gene Kim (2014) says, "DevOps is more like a philosophical movement, and not yet a precise collection of practices, descriptive or prescriptive." Because it has existed informally for 10 years other experienced professionals in the software world assert that there are rich practices to be mined to support the DevOps role (Kowolowski, May 7 2014).

This paper provides relevant test processes and tools for quality engineering for DevOps customers. This paper analyzes author's own most fruitful quality engineering strategies and those of his colleagues. The company pivoted from marketing software solutions offering the software tools to DevOps personnel. Case studies of testing practices of a few celebrated software development companies serve as reference for this paper. Oregon's vibrant community meetups provide supplementary information. This paper is framed to help agile software testers to support software testing in a DevOps focused software development environments.

Biography

*Dwayne was a Senior Quality Engineer for Cloud Services and IP devices of Ensequence, a TV engagement solution company(2013-2014). Dwayne whet his appetite to write about quality engineering in his UO Applied Information Management capstone paper titled "The Role of Testers in an Agile Software Development Life Cycle within B2B Companies." Dwayne previously taught middle school math for several years and was a tax/engineering analyst. Dwayne enjoys combining software disciplines to solve software quality challenges. Currently, Dwayne is quality engineer for CrowdCompass by Cvent.*

# 1. Introduction

The TV software company (the company) of this report offers a solutions-based approach to television program providers. The company DevOps professionals maintain the software and hardware infrastructure for the enhanced TV viewer experience. The company has a deep understanding of software development lifecycles. The approaches offered in this paper offer the most benefits when combined with transparent continuous improvement practices. Organization-wide, the company developed production releases which ensure the DevOps resources are protected and improved. The company leaders supported many of these practices that complemented existing approaches. This paper's sections include redefining testing environment; expanding testing roles; software tools selection processes; lessons learned; industry trends.

Testers observed the DevOps workflow early and often at the company. DevOps routed their operation issues through the testing team as well, which focused on documenting the issues, tracking them, and testing the development team fixes. Most of the tools natural to testers described in this paper are used in conjunction with our established browser-based functional testing tools, such as Google's Chrome development tools. The development team helped execute many of the initiatives mentioned in this paper. Developers and testers were keen to document their procedures in order to maximize the efficiency of new processes. Developers provided testers secure shell (SSH) permissions for servers and git code repositories. Gitlab ™ enabled testers to contribute to development discussions on software code discussions.

Testers started testing each development effort in order to validate it as shown in the diagram below, rather than individual pieces of software produced. The company more broadly defined the Microsoft (2014) identified development effort to include working software, application program interfaces, hardware, and database configurations, see Figure 1. The company also defined stakeholders to include internal and external customers.

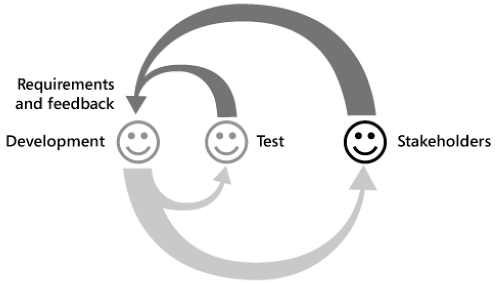


Figure 1. Quality verification of development effort

# 2. Redefining Testing Environment

Because scaling the company software environment configurations was often rocky, testers corralled the testing environment as an important factor for the development process.

## 2.1. Deployment And Monitoring the Environment

Testers and development agreed on a testing infrastructure similar to the company production (DevOps controlled) environment. Testers used Rackspace™ (the cloud server provider) tools to deploy new testing servers, databases, and load balancers. Modeling production servers required understanding which services and databases should be paired. Testers used the command-line bash programming language and Unix servers to deploy new branches of software code into servers, mimicking the DevOps professionals setup. Testers support DevOps professionals monitoring tools for detecting and fixing hardware issues. Solarwinds™ complements our internal monitoring tools and is a model for our own internal monitoring software. The company investigated how to test the integrated system of software devices. Testers mocked more of the infrastructure in many-to-one system tests. After testing cycles the software code and environment variables were both migrated to the company a staging environment. Testers and development provide more robust documentation and intuitive software for the DevOps team but still are ready with answers to DevOps questions.

### 2.2. Testing The Larger Environment

Deploying the larger testing environment meant longer test cycles in the short run, as more components were vetted in more combinations. A software ecosystem is made up of software solutions that facilitate the activities in associated social or business (Bosch, 2009). Testers prepared for longer test cycles by carving out software project time for integrating all the products and communicated this request to organization stakeholders. But experiencing the integration cycle still unsettled the testing team as a whole. Some testers shone with their debugging and application programming interface (API) skills. Fiddler an API desktop trafficking tool helped to understand the data pipelines of the software stack. Still, the new infrastructure debugging brought the Postman browser tool to the forefront as a more user-friendly interface. Testers of various experience levels with API testing helped isolate development issues. The API testing also helped DevOps workarounds before the graphical user interface (GUI) software was completely stable.

## 3. Expanding Testing Roles

Testers sometimes performed a small but significant series of responsibilities that are typically reserved for technology specialists in other software development life cycles: primarily in technical writing review, peer code review, database administration, and software configuration.

### 3.1. Development Documentation Verification

Testers reviewed existing development documentation and used it to complete deployments. API documentation helped testers identify development issues more specifically and quickly. By a series of comparisons, testers identified when API procedures and documentation were not consistent. For example, when developers inadvertently changed Javascript Object Notation (JSON) objects structures, testers suggested restoring the expected state of the objects. Even in cases where development documentation was robust, testing discoveries helped prompt development to deliver more mature graphic user interface tools.

### 3.2. Code Level Understanding

Testing the company technology infrastructure required in depth knowledge of it and—likely—its alternative systems. The development team encouraged testers to understand and to peer review unit tests. Unit tests are building blocks for assuring pieces of business knowledge were always available. Unit tests also delivered snapshots of the pieces of software function to testers, helping to form mental models of the technology infrastructure. Quality engineers used their access to Git (code version management software) to understand the specific nature of every code commit. It also helped suggest when modules might be especially susceptible to bugs. Tracking down buggy areas of the software was occasionally as simple as finding unreadable javascript code. One javascript comparison of 3 values was more complicated than it needed to be and provided unreliable software. (The comparison should have read like the mathematical arrangement of three variables $x<y<z$.)

### 3.3. Database Observations

The testing team used database administration access to experiment in short cycles in development - shorter cycles than permissible in production settings. Therefore it was important for testers to configure the databases. For example, DevOps professionals collected relational table formatted database production metrics on 15-minute intervals using the MySQL Workbench ™ tool. However, testers mainly shortened those intervals to 1 minute in testing using Mongo database tool. Testers also imported production data into the testing databases to get snapshots of television viewing data. Database observations helped detect whether representational state transfer (REST) methods of data were successful. Mongo™ Nosql (nonrelational document formatted) database tools helped testers get authorization for the user information quickly. In one instance the database should have helped testers detect when a service was out of date. Database observations revealed that a datapoint that should have been reported in the software disappeared due to the wrong time stamp. Understanding differences between relational database unique identifiers (longer integers) and non-relational identifiers again helped speed the fixing of a few development issues.

### 3.4. Load Testing

The company considered launching its product suite during some of the most celebrated live TV events. Such events might have included the Grammy's. The company executives needed load testing proof that the TV application infrastructure was scalable in order to confirm business contracts with external customers. The load testing initiative helped justify that the company offerings could withstand large increases in the requests of TV viewers. Amazon web services supports the "Bees with machine guns" approach to load testing their own servers. Netflix ™ offers the "Simian Army" suite of tests for maintaining the availability and reliability of their cloud services. Testing also helped establish the financial cost to the company in the case of spikes in the use of the application.

### 3.5. Verification of Third Party Services

The company interactive experience leverages common third party services such as Facebook ™ and Twitter. As these services created their own ecosystems testers inadvertently grew to understand them better. On occasion when a service did not run for testers, it was due to the social networks changing their authentication protocols. This realization helps DevOps and development anticipate shortcomings in the social services, which should shorten the fix cycle on the production environment. Twilio ™, a telephone messaging, third party service offers extensive documentation of their offerings online. When the company monitoring tools did not report on text failures to landlines, testers used the Twilio documentation as the basis of feature requests to developers. The company projects also harness open-source software packages such Node Package Manager (NPM). Software configuration managers might be the ones to police compatible versions of software, however testers were sometimes on the front lines of compatibility issues. NPM made unexpected production changes to their packages that the company testers helped shield from the company technology architecture.

## 4. Software Tools

Testers added to their browser-based testing toolboxes throughout this quality engineering effort. The main testing considerations for selecting tools were usability and development support. For example testers preferred Postman to Fiddler, because Postman has a more intuitive user interface.

### 4.1. Development Support

More often than not, other colleagues at the company were already using the tools to some degree but testers incorporated them into testing workflow. The DevOps team needed Microsoft Windows ™ desktop computers for maintaining some legacy hardware of business customers. The development team needed to work on Mac computers. The computer operating systems determine the testing tools that are supported. Testers at the company use both of these two operating systems. The Gitbash tool helps windows machines SSH into Unix servers. Computer sharing software Real VCN ™ help testers access their Mac computers remotely from their windows ones. Testers might have wanted to use Microsoft Word for all their documentation but the hypertext markup language Markdown was most expedient for both the development and testing teams. Testers were versed in the Python language for their own efforts but read Javascript in order to understand development code. Foundational understandings of relational (MySQL) and non-relational databases (Mongo™) helped testers reconcile the databases in the testing environment.

### 4.2. Testing-Specific Coding Tools

Testers already maintained the use of two languages in their own workflow. Python automation scripts help setup the testing environment. Object oriented programming also help make the automation scripts easy to maintain. The Python scripts facilitated data imports into the television software authoring tools. Windows batch scripts facilitated deploying the new software builds and making limited choices about the environment. The scripts also enabled testers with various levels of experience to support the deployment activities. Traditionally batch files take more time to understand because scripts more directly communicate with computers. These tools helped testers made expand the testing toolset.

# 5. Software Industry Advanced Test Practices

The software industry leaders Linked-in, Facebook, and Google suggest practices to help the company testers to deploy higher quality software even faster.

### 5.1. Linked-in

Linked-in's test strategies are quite mature and have a diagram, see Figure 2 (Mohapatra, 2014). While the company makes contextual decisions, Linked-in has a strategy diagram that helps determine which products are shipped. The TV software solutions company would benefit from test plans that reliably identify test strategies for release. Further, continuous integration automation tests would likely improve the quality of our software. The company built an automation framework using Selenium and Google Chrome but needs to insist on that framework as part of its integration processes. The tests should be tracked on a dashboard and compared to production issues to make sure that the tests are reliable and robust.

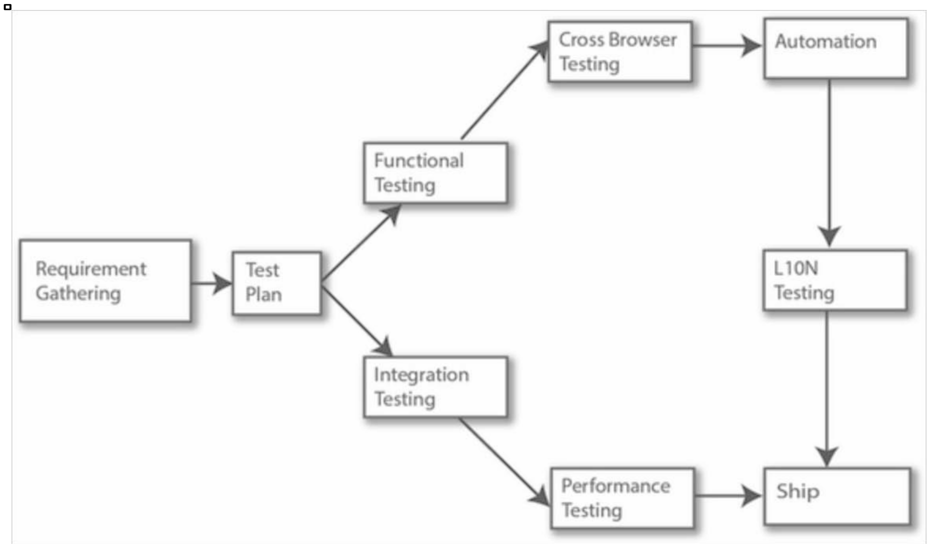


Figure 2: Linked-in test strategy diagram.

### 5.2. Facebook

The company could apply Facebook's small random sample of testing on its television viewers (Boz, 2014). The TV software solutions company treats its published advertisements as the most effective ones, when the software solutions company could be learning from television viewers to increase the interaction rate of its advertisement assets. External tests can reduce some of the internal debates among the development team and increase the investment return of the products.

### 5.3. Google

Google has specialized its code-based testing practices among testers into two roles (Whitaker, et al 2012). The first role focuses on programming practices throughout the organization. The second role focuses on testing as a feature, which impacts users and the software product. Additionally, the second role writes production code that focuses on increasing the verification of external inputs. By including quality features in the product itself, the features become more resilient. Test engineers might lead exploratory testing sessions or manage a beta testing effort. They might be focused on programming user scenarios. The TV software solutions company might need to similarly define its testing roles in order to make more scalable software. The company might also need more customized testing tools similar to ones that Google created.

# 6. Lessons Learned

Regular preparation and verification of all development changes improved the product development flow to one that facilitated the company success and an enhanced television viewer experience. At the time of this paper the company had passed several market trials and had amassed one of the largest footprint of interactive TV advertising in the U.S.

### 6.1. Bridges To Higher Quality Software

Full stack technology testing decreases risks to internal DevOps and external customers. Many of the consequences of this additional testing were not predicted, but all were beneficial to the company's software quality goals. The products constantly received helpful feedback on its quality from testers. The supporting software tools were readily available to testers for download. Documentation of the software is available online. Software courses by MongoDB and the javascript course on Code School helped fill some information gaps as well. Software quality success shortens deployment cycles and helps increase external demand for the company software solutions. Yet there are also quality of life benefits to quality software for DevOps professionals who experience more predictable deployments and operations. The benefits to testers include increased learning and collaboration (Novak, June 2014).

### 6.2. Testing New And Legacy Technologies

Testing engineers were not thorough about integrating new and old technology infrastructure. Specifically, database compatibility between Mysql and Nosql integrations were blindspots. Test engineers did not mock the real usage of the software solution. DevOps professionals were wary that test engineers only tested the latest versions of the services. In cases like this one, testers relied on development to more explicitly support the DevOps team.

### 6.3. Areas For Further Exploration: Security Testing

The Open SSL heartbleed virus hamstrung much of the software community earlier this year (Wikipedia, July 2014). Testing precautions might have helped with this condition. SQL injection attacks are popular among testers. However, XML injection attacks are also feasible and need to be more thoroughly vetted by testers (Owasp.org, July 2014). Increasingly, the company will hold more personal customer information and will need to be sure that passwords or other confidential information are always guarded.

# 7. Conclusion

The TV software solutions testers found that DevOps-focused product development benefit internal customers and believed that these same focused development practices would also benefit external customers. Prior to important product releases, testers reduced the DevOps questions from a few dozen to several focused readily answered ones. The testing initiative in this paper took place on technology stacks with various histories. Some of the software and hardware tested had existed for months before they were incorporated into the revamped testing process.  On the other hand, many of the assets tested were brand new to the company product suite. Implementing targeted testing and verification achieved more software quality goals sooner for development, DevOps teams, and external customers.

# References

Bosch, Jan. August 2009. From Software Product Lines to Software Ecosystems. Accepted for SPLC 2009 (13th International Software Product Line Conference)

Boz. August 10, 2014. Building and testing at Facebook. https://www.facebook.com/notes/facebook-engineering/building-and-testing-at-facebook/10151004157328920

Gene Kim. DevOps Cookbook.  June 2014. http://www.realgenekim.me/DevOps-cookbook/

Gene Kim. April 21, 2014. DevOps Patterns Distilled -  Implementing The Needed Practices In Four Practical Steps. Meetup: FutureTalk with Gene Kim.

Louis Kowolowski.  May 7, 2014. "What is DevOps and How do I become one?" Meetup: Hack the People.

Microsoft Development Network. August 10, 2014. Testing in the Software Lifecycle. http://msdn.microsoft.com/en-us/library/jj159342.aspx

Mohapatra Sony. August 10, 2014. Linkedin’s Testing Methodology. http://engineering.linkedin.com/testing/quality-control-linkedins-testing-methodology

New Relic. June 2014. The Benefits of DevOps: What’s in It for You? http://blog.newrelic.com/2014/06/06/DevOpsbenefits/

Owasp.org. July 29, 2014.  Testing for XML Injection (OWASP-DV-008) https://www.owasp.org/index.php/Testing_for_XML_Injection_%28OWASP-DV-008%29

The Top 11 Things You Need To Know About DevOps. Gene Kim. IT Revolution Press. June 2014. http://www.thinkhdi.com/~/media/HDICorp/Files/White-Papers/whtppr-1112-DevOps-kim.pdf

James Whitaker, Jason Arbon, and Jeff Carollo. 2012. How Google Tests Software. http://it-ebooks.info/read/1200/

Wikipedia. July 29, 2014. Heart Bleed. http://en.wikipedia.org/wiki/Heartbleed

# Scrum Adoption in our Experience: TGCFO Approach

**Vadiraj R Thayur**

Vadiraj_Thayur@McAfee.com

## Abstract

Agile is a buzzword in the software industry and most companies are investing a huge amount in adopting Agile practices. Scrum, being one of the most prevalent agile methodologies is not very easy to adopt. There is no single way of adopting Scrum and that is why it is important to also learn from the experiences of teams who have adopted Scrum successfully.

Scrum is very flexible and this makes adoption tough and confusing. There are a huge percentage of teams worldwide who have been unsuccessful in adopting scrum, resulting in undesirable outcomes. An approach that works for one team might not work for another, even within the same company. If the project involves geographically distributed teams, the challenge gets tougher. So, adoption should be a mix of recommended practices and some real-world learning.

The basic intent of this paper is to share the scrum adoption experience of a Distributed Agile team in McAfee. This approach also conforms to the Scaled Agile Framework adopted by McAfee for Distributed Projects. In McAfee, as part of the Business Platform Services (BPS) team, we adopted Scrum.

Because there is no best way for adopting Scrum, along with basic scrum principles, experiences of teams also come in handy. This paper shows the actual steps the team followed to successfully adopt scrum and its results. It could be a good case study for teams planning to adopt scrum in their projects.

## Biography

*Vadiraj Thayur is a Software QA Manager at McAfee, currently working in the McAfee India Center in Bangalore. He has been working for the past 10+ years in different QA roles on Enterprise as well as SaaS products. He is a Certified Scrum Master and has played the role of a Scrum Master for more than a year.*

*Vadiraj is a Bachelor of Engineering in Information Science from VTU, Karnataka, India. He also holds an M.S. in Quality Management from BITS Pilani, India.*

# 1. Introduction

A long time ago when the software industry was in its **infancy**, projects were limited and market requirements were very much stable. There used to be market trends, which were relevant for long period of time. Companies had adequate time to build products to suit the market requirements. They could afford to have projects running for several quarters and in many cases a few years. Another factor helping them was the lack of competition. This was the age of the **Waterfall Model** and its variants. Though there were other models existing at that time, Waterfall was the most widely used.

There were people **collecting requirements** from the field, and feed them into another team which **analyzed** them. They would then forward a set of streamlined requirements to a set of architects. The architects would prepare a **design**, first a high level view and then a detailed one. This design would get into the hands of focused team of Engineers. They would put their heads down and **implement** the detailed design to working software. A team of testers would then pick up the software and **test** it, log bugs, verify the fixes given and then certify the software for release. The **documentation** team would prepare product documents. It was then the job of the **Implementation** team to deploy the software into the customer's environment. These were the steps normally followed as part of the Waterfall approach.

As years passed by, the industry matured. Customer's needs became complex. It was the age of new inventions. Trends were no longer relevant for years together. Many players emerged and competition spiced up. Hence, requirements started changing more often. This was when the **iterative model** of software development evolved. Projects were broken down into iterations and at the end of all the iterations; the software was released to the customer. This was the **childhood** of the software industry.

Now it is **teenaged**. It is characterized by very frequent changes in technology. Customers many times have vague requirements, which might change often. Trends sometimes last for a few months. If projects take several months to complete, the output might no longer appeal the customer. The competition is hot. Each space is clogged by many players. The more adaptive approach is at advantage. This is when **Agile** took birth. Agile could also be classified as a refined iterative development approach.

The motto of Agile is to **Inspect and Adapt.** Project cycles are broken down into smaller **sprints.** All stages of the old-fashioned waterfall model are fit into each sprint. Teams are cross functional; they have representatives from each area. At the end of each sprint, the software is ready to be delivered to the Product Owner. In some companies, at the end of each sprint, the software is released to Production. In fact, I know of a sustaining team in McAfee who release software to production at the end of each sprint (15 days). Some companies have gone to the optimal level of Agile, delivering builds on a daily basis to be deployed onto the live site.

There were some compelling reasons why McAfee chose to move towards Scrum. The market was becoming more and more dynamic. Competition was getting tougher. The security landscape was also changing very quickly. The PC business was going flat, and new devices were coming up in quick succession. Security threats were also coming up for the new devices. So, McAfee had to get new products to the market faster, they had to deliver features in increments and do that consistently. Whenever something new came up, their priorities had to change to keep up with the trend. Scrum was a perfect fit to handle this situation.

# 2. Problem Statement

The principles of Agile are very close to a person's life. In both, the changes are constant—the future is unpredictable. Competition is fierce. Both are likely to fail at times, but the key is to learn from failures and move ahead. The key is again **Inspect and Adapt.**

More and more companies are embracing agile. It has become a buzzword in the industry.

If Agile is a Software Development Model, following are some popular Agile Development methodologies:

i. Scrum

ii. Lean and Kanban

iii. Extreme Programming (XP)

iv. Crystal

v. Dynamic Systems Development Model (DSDM)

vi. Feature-Driven Development (FDD)

**Scrum** has emerged as a popular approach among these, mainly due to its simplicity, improved productivity, and ability to act as a wrapper for various engineering practices promoted by other agile methodologies. The following references provide some insight into the popularity of Scrum:

- https://www.techwell.com/2013/02/why-scrum-so-popular
- http://scrummethodology.com/has-scrum-become-the-face-of-agile/

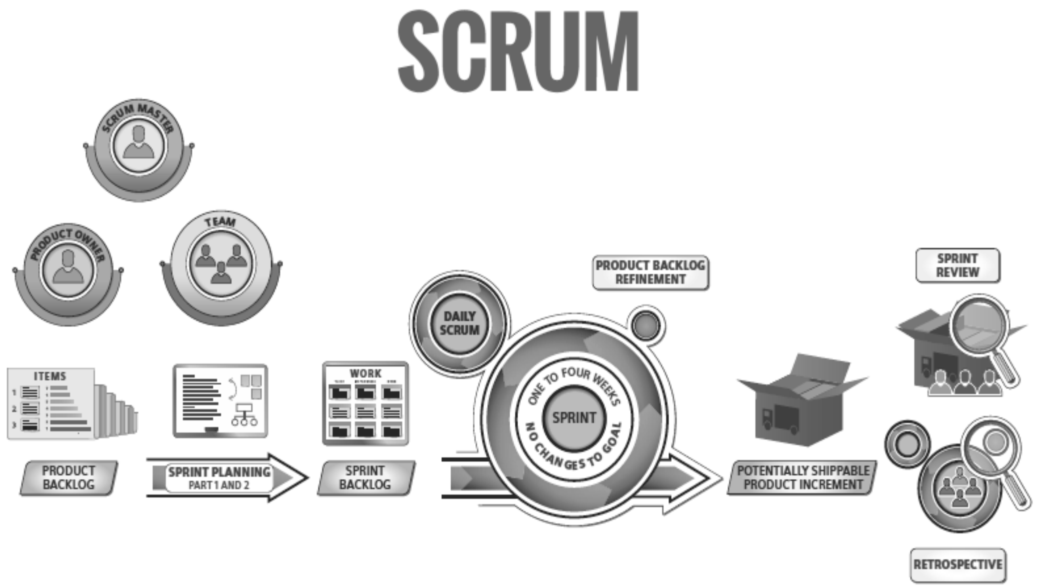
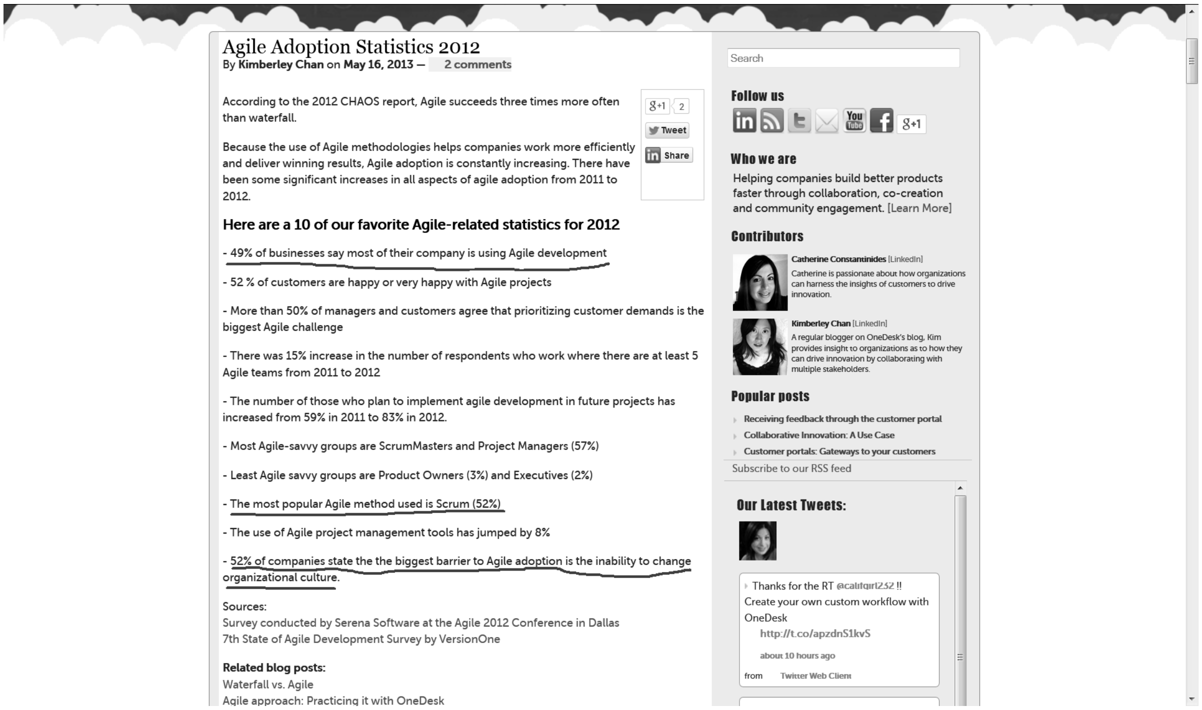


*Fig 2.1: Scrum Overview (Source: Scrum Primer, A Lightweight Guide to the Theory and Practice of Scrum. Authors: Pete Deemer, Gabrielle Benefield, Craig Larman and Bas Vodde)*

In Scrum, the "Product Owner" works closely with the team to identify and prioritize system functionality in form of a "Product Backlog". The Product Backlog consists of features, bug fixes, non-functional requirements, etc. - whatever needs to be done in order to successfully deliver a working software system. With priorities driven by the Product Owner, cross-functional teams estimate and sign-up to deliver "potentially shippable increments" of software during successive Sprints, typically lasting 30 days. Once a Sprint's Product Backlog is committed, no additional functionality can be added to the Sprint except by the team. After a Sprint has been delivered, the Product Backlog is analyzed and reprioritized, if necessary, and the next set of functionality is selected for the next Sprint.

Many companies are adopting Scrum for their teams. They feel they can deliver more quickly and also better accommodate frequently changing requirements.



Agile Adoption Statistics 2012

By **Kimberley Chan** on **May 16, 2013** — **2 comments**

According to the 2012 CHAOS report, Agile succeeds three times more often than waterfall.

Because the use of Agile methodologies helps companies work more efficiently and deliver winning results, Agile adoption is constantly increasing. There have been some significant increases in all aspects of agile adoption from 2011 to 2012.

**Here are a 10 of our favorite Agile-related statistics for 2012**

- 49% of businesses say most of their company is using Agile development

- 52 % of customers are happy or very happy with Agile projects

- More than 50% of managers and customers agree that prioritizing customer demands is the biggest Agile challenge

- There was 15% increase in the number of respondents who work where there are at least 5 Agile teams from 2011 to 2012

- The number of those who plan to implement agile development in future projects has increased from 59% in 2011 to 83% in 2012.

- Most Agile-savvy groups are ScrumMasters and Project Managers (57%)

- Least Agile savvy groups are Product Owners (3%) and Executives (2%)

- The most popular Agile method used is Scrum (52%)

- The use of Agile project management tools has jumped by 8%

- 52% of companies state the the biggest barrier to Agile adoption is the inability to change organizational culture.

Sources:
Survey conducted by Serena Software at the Agile 2012 Conference in Dallas
7th State of Agile Development Survey by VersionOne

**Related blog posts:**
Waterfall vs. Agile
Agile approach: Practicing it with OneDesk

**Follow us**

**Who we are**

Helping companies build better products faster through collaboration, co-creation and community engagement. [Learn More]

**Contributors**

**Catherine Constantinides** [LinkedIn]
Catherine is passionate about how organizations can harness the insights of customers to drive innovation.

**Kimberley Chan** [LinkedIn]
A regular blogger on OneDesk's blog, Kim provides insight to organizations as to how they can drive innovation by collaborating with multiple stakeholders.

**Popular posts**

- **Receiving feedback through the customer portal**
- **Collaborative Innovation: A Use Case**
- **Customer portals: Gateways to your customers**

Subscribe to our RSS feed

**Our Latest Tweets:**

Thanks for the RT @califgirl232 !! Create your own custom workflow with OneDesk
http://t.co/apzdnS1kvS
about 10 hours ago
from Twitter Web Client

*Fig 2.2: Agile Adoption Statistics (Source: http://www.onedesk.com/2013/05/agile-adoption-statistics-2012/)*

The adoption of scrum has not been easy. The process appears to be simple but it is just a framework to build upon. It is not a panacea. Each team will have to customize the approach based on their requirements. Scrum just points at the problems but the resolution is still left to the implementing team. This acts as an advantage and disadvantage. It makes the process flexible but it is also one of the main reasons for failure.

In an interview, Ken Schwaber, the iconic leader of the scrum community said "I estimate that 75% of those organizations using Scrum will not succeed in getting the benefits that they hope for from it." *Source: http://www.netobjectives.com/blogs/scrumbutters-scrumdamentalists-were-mad-hell-and-arent-going-take-it-anymore*

Hence, the approach taken to adopt scrum is quite important. Since there is no prescribed approach, studying and learning from scrum adoption experiences of teams comes in very handy. Hence this paper.

This paper describes the approach followed by the team and highlights some simple recommendations. It includes things which went well and things which went wrong. Some interviews with team members would also be part of this paper. The entire approach was for a team, which had two scrum teams in different sites. So, the Distributed Agile concept is also covered in the paper.

# 3. The Approach

This approach is based on the basic **Scrum Principles** and the **Scaled Agile Framework** adopted by McAfee for Distributed projects. It is called **"TGCFO Approach"** as an enumeration of the 5 phases involved.

The basic Scrum principles are:

- Empirical Process Control
- Self-Organization

- Collaboration
- Value based Prioritization
- Time-boxing
- Iterative Development

(Source: http://www.scrumstudy.com/scrum-principles.asp)

The Scaled Agile Framework adopted by McAfee has some enhancements to the traditional agile framework. It has been designed to support geographically distributed Scrum teams working on a single project. Mainly, a new role of **Product Owner Proxy** has been introduced. This person would be responsible for interacting with the Product Owner and obtain requirements and clarifications. This would eliminate the dependency of the Scrum team on the Product Owner who would usually be in a remote site. **Scrum of Scrums** is another process which involves Scrum Masters, Product Owner and Product Owner Proxies interacting on a regular basis to discuss progress and issues.

The TGCFO Approach which is based on these two comprises of the following phases:

- **Train**
- **Gel**
- **Coach**
- **Follow**
- **Own**

This approach is based on the success story of one of the enterprise teams in McAfee which adopted Scrum successfully. The team had representation across two sites: India and US. Both teams were working on the same set of requirements. Both teams were new to Scrum and the approach is based on the experience of the India team in their journey of Scrum Adoption.

## 3.1 Train

To begin with, the **Scrum Team** should be formed before the team attends the training. The Product Owner, Scrum Master and Proxy **roles** should already be identified. This will enable the team and the individuals to visualize their roles throughout the training. They would have a better understanding of their role and it would help them execute better when they actually start implementing Scrum.

Training needs to be done before introducing something new to the team. The team needs to know at least the theoretical concepts of scrum before plunging into it. They need to know the basic scrum principles, scrum ceremonies, scrum artifacts, the members of scrum team, the advantages of scrum and the disadvantages as well.

The main shift that has to happen is with the mindset of people. In Scrum, the roles of Manager, Lead etc. diminish. It is the team, which is fully empowered and responsible for success and failure. So, the shift from a hierarchical mindset to a horizontal structure mindset is not that easy. Here, training plays a very important role in helping the team understand their roles and responsibilities, understand the process and also helps the management understand their roles.

It is recommended that the entire scrum team (rather the future scrum team) attend the **training** together. In this way, they get to know each other (if they don't know already) and they could experience the entire training together. It would serve as a great kick-start to their scrum journey. After all, it's people who make processes successful.

Choice of the **Trainer** is equally important. The trainer should be a **Certified Agile Coach** and a **Scrum Practitioner.** It would be great if the trainer could continue with the team as a coach in their learning phase. In that way, the trainer can practically demonstrate what was covered in the training sessions.

The content of the training would be equally important. It is recommended that **training content** be reviewed with concerned people in the company so that they can ensure that the training would also cover aspects, which they might want to emphasize. The training should not be a mere presentation. It should involve live exercises for the team at every step, like a mock Release Planning, a mock Sprint Planning, demonstration of a Daily Standup; Exercises on how to write Stories and Tasks, Estimation exercise and so on. This would keep the training interesting and would also give the team a first shot at all the different aspects of Scrum.

### 3.1.1 In Our Experience

The market was becoming more and more dynamic. Competition was getting tougher. The security landscape was also changing very quickly. PC business was going flat; new devices were coming up in quick succession. Security threats were also coming up for the new devices. So, McAfee had to get new products to the market earlier, they had to deliver features in increments and do that consistently. Whenever something new came up, their priorities had to change to keep up with the trend. Scrum was a perfect fit to handle this situation.

When McAfee started investing in Scrum, the goal was to slowly move the company towards Agile Development. Our team members already had experience with Quarterly Delivery in the previous product. So, when work began on the new product (Business Platform Services), they were the natural choice for experimenting Scrum in our Business Unit.

The team was divided into multiple teams, each team working on a specific part of the product. There was another part of the team in United States. A **Scrum Master** was identified for each team   and since the **Product Owner** (in our case, the Product Manager) was in a remote location (United States) and a **Product Owner Proxy** was identified.

A **Scrum Coach** was identified by our company for each Engineering site. Each of them were experienced Agile Practitioners with multiple years' experience. The coach from US was flown in to do the training. It was two day offsite training.

Before the training, Scrum Masters had a session with the Product Owner Proxy and they identified some real epics and stories to carry into the training. They also had discussions to come up with some challenges / questions that they could think of so that they could discuss during the training.

The agenda of the training was a mix of concepts and simulation. The first day was mostly on **concepts**: What is Scrum; The approach; Roles and Responsibilities; Scrum Ceremonies; What is Epic, Story, Task and so on. This was followed by some **practical simulation,** which continued onto the second day. Some activities were like Breaking down an Epic into Stories, Story into Tasks, a 30-minute Release Planning; a 30-min Sprint planning and so on. Then, the art of estimating using Story points was introduced. This was followed by a game of **Planning Poker.** And finally, the training ended with an extensive Q&A session.

It was a great fun-filled experience for the team. However, there was quite a bit of confusion in their minds. It was a totally new process and it required a totally different mindset. Everyone had apprehensions.

### 3.1.2 Comments

Here are some interesting comments from McAfee team members about the training:

Shyamsunder said, “Scrum training was quite useful. The training helped in many doubts on scrum, agile getting clarified.  At the end of two day training everyone not only got to know of various terminology but most of methodology, it’s advantageous over waterfall. It provided the solid foundation and platform to embrace scrum. It helped in getting clarity on roles, responsibility of scrum members.”

Shruthi said, “Scrum training was really useful as we got a chance to understand agile methodology very well. It was very interactive, knowledge imparting, fun filled training. The training has acted as a strong foundation for the Scrum process being followed.”

## 3.2 Gel

This is a very important phase. It has nothing to do with the Scrum Methodology but this is the key to success. Scrum methodology relies heavily on people and their mindset. Scrum lays a lot of emphasis on collective thinking and decision-making. The entire team is responsible for any success or failure. A cohesive team can work wonders whereas a loosely knit team can cause disasters. So, it is very important to ensure the team remains together.

One key step towards achieving this is to **co-locate** the team. Co-locating the team will enable frequent verbal communication. Verbal communication is by far the best way of communicating as it is real-time and either parties get instant feedback. Verbal communication often helps resolve issues that might take more time and effort otherwise. Co-location also enables collective thinking, which is obviously better than individual thinking. Another advantage of co-location is that entire team is always aware of what is happening and they are always in sync. This reduces the need for many discussions and eliminates many conflicts.

All members of the team should be seated together. In many companies, all of them are seated in a single room. In some companies, they are seated in a circle. In companies where there is a cubicle structure, they are located in adjacent cubicles.

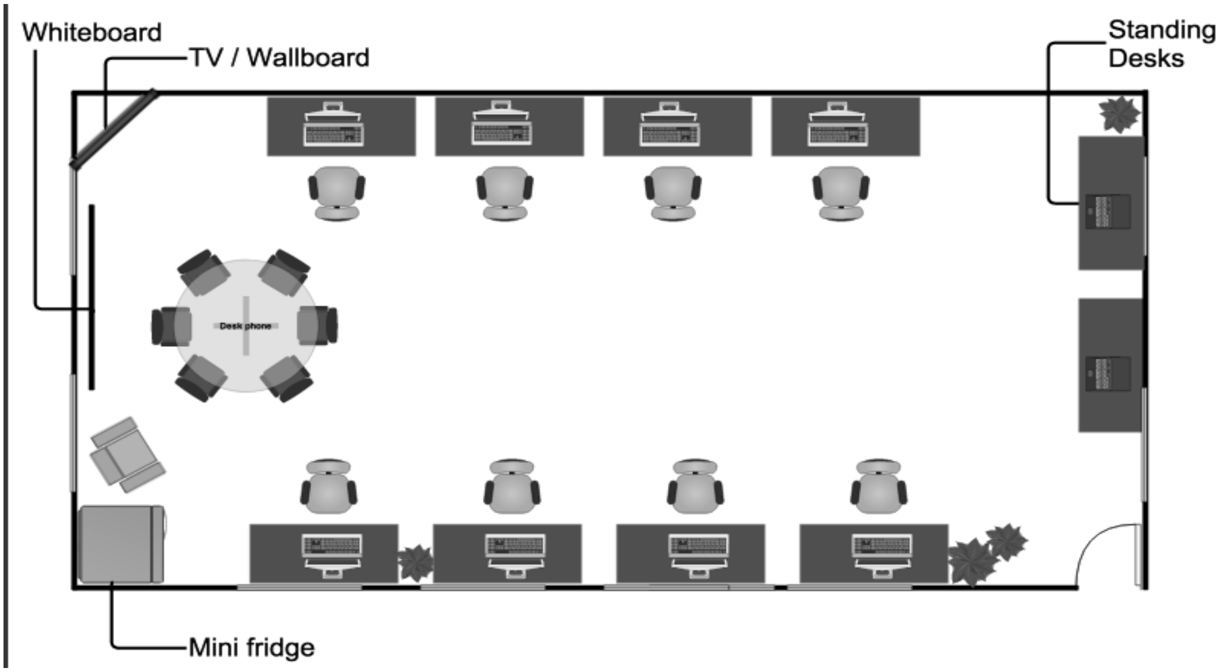


*Fig 3.1: A sample seating arrangement for a Scrum Team (Source: http://www.velocitycounts.com/2014/01/building-best-agile-team-workspace/)*

Seating the team together can facilitate lot of **informal discussions** and help build **strong relationship** between the team members. Initially, there might be conflicts but gradually, the team will start to learn to work together. This is very important in building a successful scrum team. This would also enable the team to participate in various scrum ceremonies.

Another way of ensuring the team **gels** well is to organize team building activities and outings. These would help people understand each other and that would go a long way in ensuring greater collaboration within the team.

### 3.2.1 In our experience

Prior to Scrum adoption, our team had been seated separately, Development on one floor and QA on the other. Bringing the entire team to a common location was tough considering space constraints. But the management put in a lot of effort to seat us all together.

There were lot of apprehensions initially about seating Development and QA together and there was fear of constant conflict. But, the experimentation began. Initially, there was not much communication; people were still moving along with their old team mates. But, as scrum ceremonies started and the project gathered steam, they started interacting more.

The other thing that helped was team activities. In the past, the teams went separately for team outings. Now, they were made to go out together, be it a team outing or some team celebration or lunch. This helped build some bonding between team members and it also helped them communicate better.

### 3.2.2 Comments

Some comments from McAfee team members about team co-location:

Shruthi said “Positives: Lots of time is saved. Now we don’t have to send out mails and wait for responses, we can just walk down to scrum team member’s desk and get doubts cleared. Standups are consuming lesser time as we can gather quickly. Negatives: Sometimes we (both Dev and QA) end up consuming each other’s time when it is really not required. “

Asha said “In Beginning when the team was co-located I felt uncomfortable, felt like QA cannot share openly to DEV. But later on felt it was good sitting together as could get clarifications without any delay from DEV and vice versa.”

## 3.3 Coach

Creating the Scrum environment, training and ensuring team collaboration is the first step. The next step would naturally be to start following Scrum practices. But, it is easier said than done. The team would have to change their mindset and that would take time. So, it would be apt to have the Agile Coach to stay with the team for some time.

The teams would be accustomed to a different way of thinking before adopting Scrum. They would follow the **Big Bang Approach (Some explanation on Big Bang Approach at: http://www.slideshare.net/zumbara2009/big-bang-or-bit-by-bitdoc)** where given a set of requirements, they would sit together for days and brainstorm and plan how to implement the requirements. They would come up with the detailed design upfront and start planning the development effort. They would have fixed a release date and they would start deducing milestones moving backwards.

With Scrum, the mindset has to change. Everyone has to start thinking in a **bit-by-bit** fashion. They should do a very high-level design of the solution using the requirements. Then, they should prioritize the requirements and decide on what to work in the first sprint. Detailed design of the first sprint requirements should be planned and accommodated in the first sprint along with development based on the design. On completion of the first sprint, they should start thinking about requirements of the next sprint and so on. This requires the team to be able to logically divide the existing requirements and prioritize among them and decide on the implementation plan. They should start working with a mindset of going in deep for only the upcoming sprint's requirements and think of the other requirements when the corresponding sprint is coming up. The mindset should be to ship a Potentially Shippable Increment to the customer (if not really ship, the Quality should be at that level). To adopt this iterative thinking approach, the team needs to have a good idea of the overall solution and its requirements. They should be able to discuss efficiently and come up with a detailed execution plan.

This change cannot happen without an expert's constant guidance and feedback. Here, the presence of an **Agile Coach** becomes crucial. The Agile Coach should be actively involved with the team, at least in these initial few sprints. The coach should be part of all scrum ceremonies in this period. The Coach should handhold the team through each ceremony in the initial sprints and later, continues to be actively involved after few initial sprints. This serves one main purpose: The coach can observe the way the team is adopting the methodology and give early and effective feedback. This helps in correcting their mistakes in the learning phase itself so that they do not end up adopting a **"Scum-but"** methodology. In communicating the feedback, the coach should be careful not to over communicate / under communicate. At times, this could de-motivate the team. In this aspect, the Scrum Master can lend a hand by helping the coach communicate effectively to the team and also work as a proxy-coach and help the team start following the processes. The patience, expertise and people skills of the coach come in handy here.

### 3.3.1 In our Experience

Once our training was complete, a coach was appointed for our team. He was an external agile practitioner and had many years of experience in Scrum and coaching.

To begin with, he started attending the planning meetings and the daily scrum meetings. For the first few days, he was just a keen observer, talking only when any questions were directed at him. After those initial days, he started giving feedback to **Product Owner Proxy** and **Scrum Master** about what was not going well and how we could improve.

Our earlier daily standup meetings used to be prolonged. While giving updates, some people used to get stuck in the issues they were facing and that would trigger a long discussion. Sometimes, team members who were not part of the discussion would see that as a waste of their time. Another risk was that the focus of the standup would get lost and people might forget to bring up essential points in the standup. The coach suggested that standup should be focused and not last more than a few minutes. The agenda should be to only discuss **"What was done yesterday", "What is planned for today" and "What the impediments are".** All other detailed discussions should be taken up post standup with only relevant people.

Another instance was during the Sprint planning meeting. Since the product was new to the team and there was no product owner or Architect presence locally. The coach observed during the first couple of Sprint Planning meetings that there was lack of technical decision making in the team. As a result, most of the planning meetings ended with quite a lot of ambiguity. The coach suggested to have a local technical point of contact to help things proceed more gracefully.

### 3.3.2 Comments

Some comments from McAfee team members:

Manish said “Coach gave us good tips to leverage on the scrum practices and its importance. His admonitions during Scrum meetings help us cope with the learning curve quickly.”

Shyamsunder said “Coach attended few meetings as an observer, he noted down how team is responding to scrum adoption. He shared observations, which helped us to rectify the issues with daily standup, attention to process, streamline story writing, address gaps between Dev and QA and make the process more effective. “

## 3.4 Follow

This is the actual implementation phase of Scrum. The team would have gone through the training phase and would also have the guidance of a coach. Now, it is time for some action. The key in this phase is to follow the scrum ceremonies and scrum principles religiously. There might be some amount of tweak in the processes to suit the dynamics of the team but the basic essence of Scrum should remain.

There are various ceremonies in Scrum, which need to be followed by the team. There might be some sort of resistance within the team, they might question the need for some processes, and there might be some reluctance. But, it’s good to stick around for some time to start seeing the results. The very people who felt the processes were meaningless might start seeing the value once they start following them and also understand the intention behind them. The coach and scrum master would come in very handy in achieving this. They need to ensure the team remains focused, motivated and together.

Following are some important ceremonies:

- **Release Planning:** The team runs through the requirements that come from the **Product Owner**. They prioritize those requirements and also look at the dependencies with other requirements. They also do a high-level capacity planning analyzing the timelines and resource availability. Based on this data, they fit the requirements into sprint buckets, just to give a rough picture of what they can achieve within the release. Then, they identify the Risks, Dependencies and Assumptions and design mitigation plan for each. This is followed by planning for the first sprint.

  This is the general procedure, but some important points need emphasis:

  - The Release Planning should ideally begin with the Product Owner briefing the team(s) on the requirements, the customer perspective and priority. This would give the team clarity on what they need to implement and why. This clarity would have a positive effect on their estimates and deliverables. A much better approach could be to have this discussion a couple of days before Release planning so that team members could raise questions and get clarification.

  - If the teams are spread across sites, it would be a good practice to have sync-up calls before and after planning sessions. This could be a good starter for the planning sessions. If not the entire team, at least the Scrum masters of the teams and the Product Owner could sync-up.

  - The team should also create/update their **Definition of Done** criteria for a story, for a sprint and also for the release. Based on these requirements, they should add stories that would help in achieving the criteria (like Performance Testing, Automation and so on)

  - The Scrum Master should follow up on the Assumptions, Risks and Dependencies and mitigate them so that the team does not face any hindrance at the time of execution.

- **Sprint Planning:** This begins with the assumption that the team is already aware of what requirements need to be worked on. The team starts with a detailed capacity planning, considering the planned time off for each member and arriving at a total capacity number. Then, they start picking up each requirement and breakdown into simpler stories if required. Acceptance Criteria is added to each Story and an estimate is made. The stories are then prioritized. Based on the estimate, priority and the team's velocity, the team analyses whether all planned stories can be achieved within the sprint. After adjusting the stories into the sprint bucket, each story is broken down into simple tasks. An effort is added to each task and owner assigned. All this is done keeping the team and individual's capacity in mind.

  Some specifics to be aware of:

  - It is always good to enter a Sprint Planning with a well-groomed set of stories. Requirements should already be broken into Stories, acceptance criteria added and estimated. This would make the Sprint planning activity simpler and more effective.

  - Any ambiguity with respect to the stories for the sprint should be clarified before the Sprint planning.

  - It would be good if the team can discuss each story in some detail, consider the Definition of Done (DoD) criteria and then come up with tasks and effort estimates. This ensures that the team has thought through all aspects around implementation of the story and as a result, the story estimation becomes more accurate. They can breakdown the story into more meaningful and granular tasks and ensure that the DoD criteria is met. If the team does not discuss in detail, they might end up under-estimating or over-estimating the story and might also miss some important tasks. This might result in incomplete implementation of stories and might also affect milestones as stories might get carried over each sprint.

  - In case of distributed teams, it would be good to have sync-up calls before and after the planning meetings. At least the Scrum Masters and Product Owners should attend.

  - Ensure the stories and tasks are granular so that they can be tracked easily.

- **Daily Standup:** The basic idea of this is to have all team members at the same level of understanding and also to accelerate identification and resolution of issues. The team meets every day at a pre-defined time, each member updates on **What was done, What would be done** and **Impediments.** They are also expected to update their status/efforts (in the tracking tool if they use one).

  Some tips :

  - Standup should be quick. People should not deviate from the 3-point agenda and should be very concise. Standup should not drag on and turn into some other discussion.

  - The basic intention of each team member should be to update the full team and not just the Scrum Master.

    - Any impediments brought up by the team members should be noted by the Scrum Master and should be worked upon.

    - Any points that might need further discussion could be taken up in a separate discussion post-standup. Only relevant people need to be included.

- **Backlog Grooming:** This is very important. During the sprint, the team needs to meet at least once a week and groom the stories planned for the future sprints. They should breakdown the story into simpler stories if required add acceptance criteria and an estimate. This would help the team get clarifications on questions and would ensure that there is no ambiguity at the time of sprint planning.

  Some key things to keep in mind:

    - Team should ensure grooming happens on a regular basis

    - Whatever question comes up during the grooming, Scrum Master should follow up on those and get clarity.

    - It would be good to also identify Risks and Dependencies during grooming so that they could be mitigated before those stories are taken up in future sprints.

- **Sprint Review:** The team presents their work to the wider audience at the end of the sprint. They seek feedback from Product Owner and other key stake holders on the completeness of the story. If the Product Owner and Stakeholders feel that all the required criteria are met, the story could be closed. If not, the story could either move into the team's backlog or a new story could be created to implement the feedback.

  Some important points:

    - Only Completed stories should be demonstrated.

    - Each story demo should begin with an overview of story followed by the acceptance criteria and then the actual demo.

    - Scrum Master should take note of the feedback and ensure it gets incorporated in future.

    - Incomplete stories should either be moved back into the backlog or should move into the next sprint.

- **Sprint Retrospective:** This is more of an introspection opportunity for the team on completion of a sprint. They usually discuss **What went well, What needs Improvement** and **What are the steps to be taken.** Each team member should list top two things which went well and top two things which need improvement. Then they vote for the top two things to be improved and identify the plan of action.
  The retrospective is a very important step towards continuous improvement. this gives each member a chance to express their concerns and also enables the team to introspect. Not only this, it is also a platform to celebrate and recognize the achievements of team members. People feel valued in the team if they get recognition amongst their peers and leadership for an extraordinary effect or some creative idea.

Some important points to note:

- A Scrum master should take care of tracking the steps identified for improvement.
- The items identified in the previous retrospective should be reviewed and the progress analyzed.
- Retrospective should not just be a complaining session, the team should also celebrate the success they saw during the sprint.
- In case of distributed teams, sharing the retrospective notes would be important so that the offshore team also gets an opportunity to see them.

The Coach plays an important role in this phase. In the initial phase, the coach would be involved in each ceremony, guiding the team through each ceremony. But as the phase progresses, the coach's involvement should decrease gradually. Towards the end of the phase, the coach's involvement should be minimal.

### 3.4.1 In our Experience

The team had very interesting experiences during this phase.

To begin with, there was a lot of confusion. They did not know what to do in each ceremony. Though they had some theoretical knowledge from the training, practical implementation was a different ball game. The coach came in very handy at this stage. He gave valuable suggestions at each stage, sometimes in the meeting itself and sometimes to the Scrum Master. The team on the other hand did not show much interest to start with. They complained that there were too many meetings. There was not much focus in the meetings too.

As sprints passed by, with constant guidance from the coach, the team started realizing the importance of the ceremonies. But now, there was another issue. The team was interested in the ceremonies but they did not know how to handle them in a smart way. This resulted in prolonged meetings and wastage of time.

Towards the end of the phase, the team learned the art of smart scrum ceremonies. The daily scrums were short and precise, grooming sessions were regular, sprint planning sessions reduced from a day to a couple of hours and the retrospectives were more open and constructive.

### 3.4.2 Comments

Some comments from McAfee team members:

Shyamsunder said, "Initially we were not sure how team would embrace and work per scrum but to our pleasant surprise everyone responded very well to work as per new methodology. Team(s) started demonstrating ownership and started delivering more stories, more productive. Our opinion changed when we saw some team(s) adhering to the process strictly. One of best things was everyone agreeing to "Sprint done criteria" and going extra length to achieve it. "

Asha said "We started good, but initially we were little lagging in planning, but slowly we picked up in that. Planning was one of the areas which we picked up slowly"

## 3.5 Own

This is the final stage of scrum adoption. By this stage, the team would have sufficient knowledge about the entire process and would more or less operate without the assistance of the coach.

With this, the team should start following the scrum processes religiously. There should be no requirement of a coach. The Scrum Master should be in a position to guide the team when they hit any obstacles. The team should also be in a position to think independently and come up with solutions. There might be occasional failures but those would be the stepping stones for their future success.

Another key aspect of this phase would be for the teams to continuously **inspect and adapt**. Each team's requirements would be different and so, they would need to tweak the processes accordingly. This knowledge would come from the learnings, mainly from retrospectives.

The team in this phase would enjoy their processes, come up with new ideas and resolve issues as they encounter. They should make their own decisions and be accountable.

Key things to remember:

- Ensure the team has a free hand in making decisions.
- The management should remove any fear of failure that the team might have. This comes with encouragement and supporting through failures.
- The team should continually inspect and adapt.

### 3.5.1 In Our Experience

The team is almost in this phase. They have started enjoying the processes. They voluntarily come up with ideas to improve processes. They think on their own and come up with process improvements when they encounter issues.

The fear of failure has reduced to a great extent, if not eliminated. They are ready to take calculated risks. They take feedback from retrospectives very seriously and act upon them in the future iterations.

There is still lot of scope to improve. The theme here is to **inspect and adapt.**

### 3.5.2 Comments

Some comments from McAfee team members:

Manish said "Initial phase was a learning exercise, but we learnt and adapted from our mistakes. We as a team and as individuals were transformed to plan and execute and strategize better in the project and scrum activities. The mantra that we still carry along - "Inspect and Adapt".

Shruthi said, "Defects were detected in early phases of development. Discussing design of various components with Dev even before the actual development started has helped the QA get in depth knowledge of the component. Discussing the test case design in Scrum meetings has helped us look into the test cases in various angles, get collective inputs from Dev as well as QA and hence resulted in fool proof testing.

Sometimes, some meetings end up being redundant; I think we have scope for improvement there."

# 4. Results

The results were not totally rosy. There were some things that went well whereas there are things that the team needs to improve upon:

- Very important, a set of unknown people came together and within a couple of quarters, they knit together as a single team.

- A group that could not take any decision in the preliminary stages became almost self-reliant in a couple of quarters.

- A team that hated processes transformed into team constantly thinking about process improvements.

- The team that had least predictability in terms of throughput, developed into a team with considerably good predictability.

- The team reached a stage where they were able to deliver around an average velocity. There were some sprints where they achieved a couple of points above the average whereas some others were a couple of points below the average. The below chart from Version One tool depicts the trend.

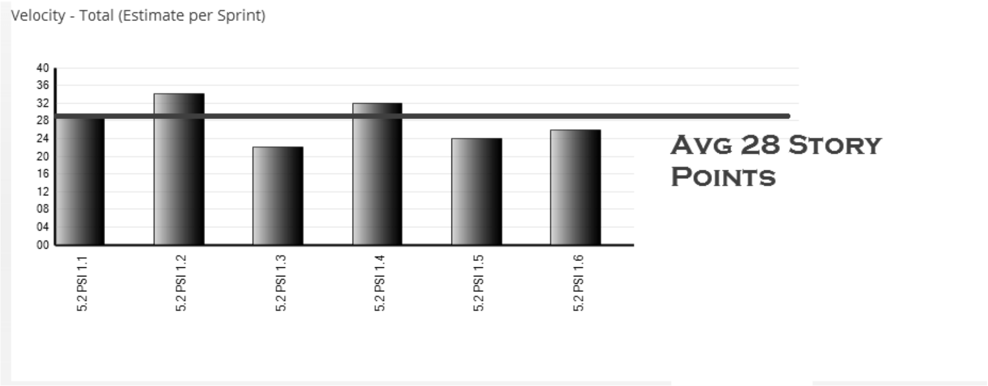


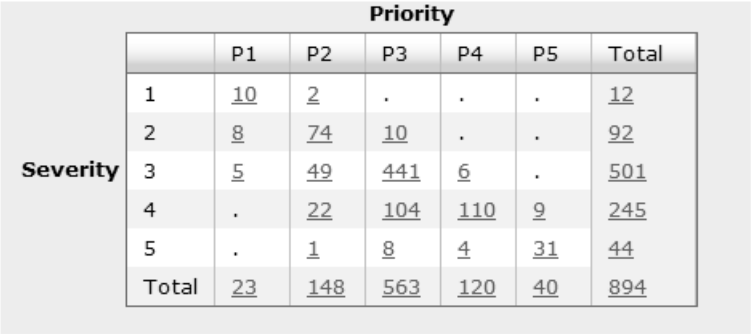
*Fig 4.1: Velocity Chart from Version One depicting Velocity across multiple sprints*

- The team was able to predict to considerable accuracy what it could achieve within a sprint.

- The defects started decreasing as the team started communicating and collaborating.

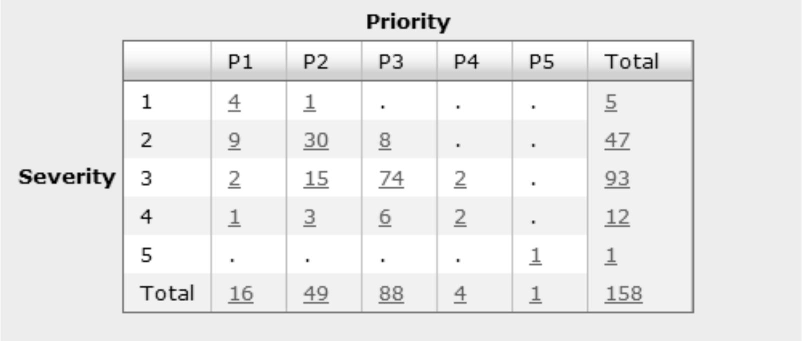
  Till the team adopted Scrum, the trend was to get lot of defects. With the adoption of Scrum, the defects reduced tremendously. The reduction could be attributed to the various quality measures like Code Review and Unit Testing, which were done as part of the Story Definition-of-Done criteria.

  Following are two defect charts showing the defect count in a project prior to implementation of Scrum and in a project with Scrum. Both projects were executed by the same team and were of similar duration and scope.



| | | Priority | | | | | |
|---|---|---|---|---|---|---|---|
| | | P1 | P2 | P3 | P4 | P5 | Total |
| Severity | 1 | 10 | 2 | . | . | . | 12 |
| | 2 | 8 | 74 | 10 | . | . | 92 |
| | 3 | 5 | 49 | 441 | 6 | . | 501 |
| | 4 | . | 22 | 104 | 110 | 9 | 245 |
| | 5 | . | 1 | 8 | 4 | 31 | 44 |
| | Total | 23 | 148 | 563 | 120 | 40 | 894 |

*Fig 4.2: Defect chart for a project prior to Scrum Implementation*



| | | Priority | | | | | |
|---|---|---|---|---|---|---|---|
| | | P1 | P2 | P3 | P4 | P5 | Total |
| Severity | 1 | 4 | 1 | . | . | . | 5 |
| | 2 | 9 | 30 | 8 | . | . | 47 |
| | 3 | 2 | 15 | 74 | 2 | . | 93 |
| | 4 | 1 | 3 | 6 | 2 | . | 12 |
| | 5 | . | . | . | . | 1 | 1 |
| | Total | 16 | 49 | 88 | 4 | 1 | 158 |

*Fig 4.3: Defect chart for a project with Scrum Implementation*

- The management gained confidence on the capability of the team as sprints passed by.

- The Dev-QA divide started diminishing. Development was ready to assist QA team when required and QA team was ready to make improvements to make life easier for the Developers.

- The ceremonies started turning into sessions of learning and enjoyment.

- The team started quite a few quality processes as part of the Definition-of-Done. Those helped to achieve better quality:

  - The team achieved more than 95% code coverage through Unit Tests. Total of more than 7000 unit tests were written.

  - The team could remove more than 60% defects through two-level code review process. More than 1000 defects were filed in Code Collaborator tool including documentation, coding standards and coding / logic issues.

There are some concerns that are yet to be resolved:

- At times, the team feels **stressed.** Immediately as a sprint comes to an end, the next sprint would be in the queue. This sometimes frustrates the team members. They get a feeling that they have no room for relaxation between sprints. They might not enter the next sprint with a fresh relaxed mindset. Eventually, they get stressed.

- There are times when there are hard timelines. The product management once sensed some opportunities in the market to sell the product. There was a Partner launch program with which they wanted to launch the pilot version of the product. Hence, the team had to put in lot of extra effort to meet the timelines. That should not be the case in scrum but there were some business compulsions. This demanded the team to work over weekends.

# 5. Conclusion

To summarize, scrum adoption is not an easy task. It is a total transformation of the team, in both process as well as mindset. So, it has to be done in a well-planned methodical manner. Here are some key points to remember:

- **Management commitment** and support is required
- **Training** is a must.
- **Scrum Master** is key person. Choose the correct one.
- **Coach** would prove to be the guiding factor. Very essential in the transformation.
- Focus on the **mindset** of people as well. **Team building** is key to the success of any Scrum adoption.
- Some hand-holding is necessary in the initial stages.
- Follow the **ceremonies**. They will start adding value gradually.
- Reduce the intervention as the team matures.
- Allow the team to take **calculated risks**.
- Remove the **fear of failure** from the minds of team members.
- **Trust** the team.

# References and Acknowledgements

## References

- The Scrum Primer, A Lightweight Guide to the Theory and Practice of Scrum (v2.0)
- Pete Deemer, Gabrielle Benefield, Craig Larman and Bas Vodde
- Various resources on the internet. Exact source mentioned throughout the paper wherever applicable.

## Acknowledgements

This paper would not be complete without thanking everyone who has been directly / indirectly involved:

- First of all, the scrum team at McAfee, which is the central point of the paper. Without them, this paper would not have been possible.

- All my colleagues for their constant support and guidance.

- Mr. Nagendra Kumar, Quality Director at McAfee whose constant encouragement and guidance helped me to complete this paper.

- Mrs. Jayashree, my ex-colleague and a Quality professional for her contribution towards reviewing and formatting the paper.

- Mrs. Jayashree Venugopala, a Lean Six Sigma Certified Marketing professional for her inputs and edits.

- My paper reviewers Leesa Hicks and Sue Bartlett for their valuable feedback.

- McAfee, my company for encouraging my effort and sponsoring my participation in the conference.

- My family for their constant support.

# How to Fail at Agile Without Really Trying

**Heather M. Wilcox**

heatherwilc@yahoo.com

## Abstract

Nobody likes to fail. We like to talk about it even less than we actually liked failing in the first place. However, if we don't talk about our mistakes, we don't learn from them. If we don't learn from them, then we are essentially guaranteeing that our failures will be repeated at some point in the future. The only thing that could be worse than failing the first time, is making the same mistake a second time for all the same reasons.

This paper examines the causes behind the failure of Agile at a Portland, Oregon software company. It presents anecdotal examples and observations of how and why problems occurred and, in doing so, provides a blueprint for a successful implementation of Agile.

## Biography

*Heather has spent the last 20 years working and learning in the software industry, choosing to focus primarily on start-up and small companies. As a result, Heather has had a broad range of job descriptions which include, but are not limited to: Tech Support Engineer, IS Manager, Technical Writer, QA Engineer, QA Manager, and Configuration Management Engineer. This has given her a wide range of experiences to draw from in her current role as a Senior Quality Assurance engineer.*

# 1 Introduction

Nobody likes to fail. We like to talk about it even less than we actually liked failing in the first place. However, if we don't talk about our mistakes, we don't learn from them. If we don't learn from them, then we are essentially guaranteeing that our failures will be repeated at some point in the future. The only thing that could be worse than failing the first time, is making the same mistakes a second time for all the same reasons.

Interestingly, most success is rooted in failure. Very few scientists, inventors, or development teams get it right on the first try. However, when the "Great New Thing" is finally revealed, nobody talks about the bugs, the failed tests, the wasted time, or all the other things that went wrong on the way to the end result that was "Spectacularly Right".

This is the story of a company that failed at Agile and some of the more obvious causes behind that failure.

To be absolutely clear, the whole time period during which we were failing at Agile, we were also successfully releasing code. Our customers were satisfied with the product that we shipped and the rate of escaped (field) defects did not increase. In fact, rate of field defects may have actually been reduced due to the increased code scrutiny that is a benefit of the Agile process.

However, we also never received the full benefits that come with a successful Agile adoption. Releases were never easy and often were uncertain until the last moment. There were missed deadlines because of a misalignment of expectations between the parts of the organization that were waterfall and the parts that were Agile. This misalignment was often caused by terminology and methodology differences between the two approaches. There was also a lot of frustration within the teams because of the challenges that came with a partially implemented Agile strategy. Essentially, things could have been much easier had we adopted Agile and utilized Scrum fully and properly.

# 2 The Road to Failure

## 2.1 Lack of Training

The road to failure is truly the path of "Not Really Trying". Our adventure started with the initial decision that only the product development team would adopt Agile and Scrum. The rest of the organization (Including Sales, most of Marketing, Support, etc.) was made aware that Development and QA would be making the transition, but only the two affected departments and the designated product owners (Marketing Business Analysts and Product Managers) received any training. The rest of the organization was essentially left to its own devices to research and learn about Agile (or not). It was decided that the POs would manage the relationship between the Scrum Teams and the rest of the organization. Essentially, they would act as translators between the Agile and Waterfall worlds.

Additionally, not all the members of the Product Development team received Agile and Scrum training. The Systems Engineering (SE)/Delivery side of the department, who are responsible for maintaining the build systems as well as the test, staging, and production environments did not receive training. So, although some of the members of the SE team knew about and understood Agile and were familiar with Continuous Integration (CI) and Continuous Development (CD), not all of them knew or understood what CI/CD was about, how to implement it, or even how to interact with Agile teams.

## 2.2 Lack of On-Going Support

Once the designated Scrum Teams and Scrum Masters were trained, they were turned loose with a set of deliverables that were to be produced: A visible burn-down chart for each team, a visible task board with stickies for stories and tasks, and a formal demo at the end of every Sprint. Additionally, all teams were expected to hold daily Scrum meetings, a retrospective at the end of the Sprint, and a planning meeting (that included planning poker) at the beginning of each Sprint.  Extra points were awarded for backlog grooming meetings.

For about six weeks, we received ongoing support:  Real time help from an onsite Agile consultant as well as additional trainings on running retrospectives and writing appropriate user stories.  We were so Agile, it almost hurt.  We had amazing momentum and it was good.  However, once the consultant left us, there wasn't really anyone with the necessary authority or desire to drive back the "rot", so the process started to degrade almost immediately.

### 2.3 Lack of Belief in Agile

When the teams were originally assembled, we ended up with a couple of groups that were improperly constructed.  Either the Scrum Master didn't believe in Agile or the majority of the team did not.  For those teams, everything was a struggle because there were deep doubts within the team about the efficacy of the process.  They "needed more training" (they could never have enough), the Sprints were too short (they were trying to do waterfall inside of two weeks Sprints – that's difficult!), and the Scrum Masters who actually believed in the process couldn't be effective because they were constantly fighting the majority of the team who didn't want to do Scrum in the first place.

### 2.4 Team Stability

Team stability (or lack thereof) was a huge contributor to our failure.  For a variety of reasons, the membership of the teams has been significantly re-adjusted about once every 6 months or so since our initial implementation of Agile.  Moving team members around in this way has had the unfortunate side effect of forcing a "reset" on team cohesiveness and output.  It's commonly accepted that it takes at least two or three Sprints for a team to gel and really become productive after formation.  It takes additional Sprints for teams to understand their velocity and begin to run smoothly.  This means that we've lost approximately two months of intense productivity out of every 6-month period.

### 2.5 Timing and Communication Problems

In the case of this company, the final stake in the heart of Agile (and Scrum) was time.  As previously explained, most of the organization did not adopt Agile.  Consequently, external commitments continued to be made to customers in "waterfall" style, meaning that clients were promised that features would be delivered on a certain date without first checking with the teams.  This is a very "top down" approach which is inconsistent with the Agile "bottom up" method of estimating and providing features.

The end result of this process was that teams were constantly behind and having to play catch up to get their work done in time to meet the commitments that were made for them.  Given that situation, Scrum rituals were the first to go to make time to simply "get stuff done."  For instance, many teams stopped tracking velocity.  Since they were given "drop dead" dates for the things they were working on, team velocity became irrelevant.  Things had to get done on time regardless of how long the work would actually take, so it made no sense to waste valuable work time on estimating velocity.  Additionally, some teams reduced their Scrum meetings to just once a week, or quit holding them altogether. Time became such a valuable resource, that it was hard to justify the expenditure of time on Scrum when much of the necessary information could be communicated via an email update.

## 3 The Signs of Failure

Some of the more obvious signs that are visible on the Road to Failure have already been described, however, there are always more than just the basic indicators.  Some of signs are a bit more complex or obscure and you need to look a little harder to see them so that you don't get lost.  In this case, these less obvious signs took the form of aberrant team behaviors.

## 3.1 WaterScrum or Scrumfall

As mentioned previously, some teams made serious attempts to run waterfall-style development inside of their Sprints. We referred to this behavior as either “WaterScrum” or “Scrumfall”. A major complaint from these teams was that a 2-week Sprint was far too short. They wanted at least 3 week Sprints with 4 weeks being ideal. Instead of breaking an epic-sized feature into smaller stories and then working on the stories in a sensible order, the hybrid teams tried to do all of the planning for the epic before starting any work. They also had a hard time trying to figure out where to store their requirements because they felt that User Stories didn’t have any place for them. This confusion around requirements arose because they didn’t understand the concept of Acceptance Criteria. Even the task stickies for these teams were waterfall – tasks were documented in painful detail, including (but not limited to) how many hours were actually required to complete a task vs. how many hours the team thought it would take.

Planning for the Scrumfall teams always took hours and sometimes took more than a day, which is a lot of time to spend planning a two week Sprint! In true Waterfall fashion, these teams would assemble every last requirement and piece of information before moving forward on any actual development work. QA for stories often happened in the next Sprint, since the entire feature had to be fully completed before it was passed to QA. Their stories also had a tendency to be too big – one story often took more than a week of development time.

## 3.2 Top-Down Estimation

Again, as previously mentioned, a large part of the organization did not convert to Agile. As a result, release schedules for projects tended to be dictated from the top-down (Waterfall) instead of bottom-up (Agile). Although most of the scheduling was not unreasonable, chosen feature release dates tended not to take Agile process time into account. It quickly became clear to many of the teams that there was no value in estimating or calculating velocity - the features needed to be done in time for the designated release date and that was that. So taking the time to play planning poker and estimate team velocity made no sense.

Additionally, in order to stay on target for the pre-determined release dates, teams threw out any process that they felt served no direct purpose or slowed them down. In-depth retrospectives were abandoned along with some Scrum meetings, backlog grooming, and other Agile rituals.

## 3.3 Feature Creep and Schedule Slippage

The top-down scheduling also resulted in a top-down approach to feature inclusion. Additional features were tacked on to releases to suit customers without first checking on how that might affect the team. If a schedule slipped for other reasons, it was taken as an opportunity to add additional features. The reasoning behind this behavior was that, “since time was added to the schedule, there must be more time available for coding, right?” However, that wasn’t really the case since the schedule was slipping in response to the need to finish work that was already committed. But again, since there wasn’t a real feedback loop to the teams, the feature would be promised to the customer and added to the release. The end result of this was often an additional (but smaller) addition to the schedule on top of the original modification.

## 3.4 Allowing the Wrong People to Drive

The longer we allowed Agile to disintegrate, the more dysfunctional things became. One of the more ineffective choices that were eventually made was to have each Development Manager act as Product Owner for multiple Scrum teams. There were several things wrong with this scenario – many of which became clear almost immediately.

The first obvious mistake we made, was allowing someone to act as Product Owner for more than one team. Being a good PO requires a big time commitment - not only to managing stories, but also researching product requirements and customer needs. Spreading someone across two or more teams meant that they were unable to do the Product Owner job well for any team. If one group was getting the time they needed, the other team(s) were getting completely neglected. So a group would have help periodically and then be on their own for a while. This created problems with keeping team backlogs in good shape. Stories tended to be inconsistent, poorly written, barely fleshed out, or not fleshed out at all. Teams also sometimes resorted to writing ALL their own stories and maintaining their own backlogs without PO input.

The second mistake was allowing the manager of development engineers act as their Product Owner. That fast became a case of the wolf guarding the hen house. As Product Owners, the Engineering Managers were being pressured to get product out. Because the POs were also the managers, the Engineers had nowhere to go when they felt like the project timelines were untenable.

The last, and possibly largest mistake, was having someone who had no marketing or customer knowledge drive the creation and prioritization of stories. Development Managers are good at development but aren't in tune to customer needs, since that's not their job. They had no clue what the product should look like or what the priority of features should be. The Development Managers also didn't understand how to turn technical requirements into stories that delivered business value. Teams floundered while the PO/Dev Managers acted as intermediaries between Product Management and the Scrum teams. Eventually, many teams ended up "firing" their POs and started getting their stories directly from Business Analysts or Product Managers.

### 3.5 Re-Inventing the Wheel

As the teams wandered further from Agile and good Scrum process, the Managers, in an attempt to "solve the problems" ended up trying to re-invent the wheel. They held meetings to discuss issues like "What the Job Description of the Product Owner should be" and "How Agile should work in the teams". The crazy part of these discussions is that these things are already very clearly defined. Unfortunately, the foundation of Agile had been lost, so the idea of going back and re-examining the actual defined roles and responsibilities to try and figure out where things went wrong didn't occur to anyone. The effort to "Fix the Process" became a twisted internal dialog that went something like: "Well, what we have isn't working right, so Agile must be broken. Maybe if we re-define the principles of Agile, it will work better." However, the crux of the issue was that the Agile implementation itself was broken, which is why the roles weren't working the way they were supposed to. If Agile was re-implemented properly, the issues surrounding the role of Product owner and Agile process itself would likely resolve themselves.

### 3.6 Fractured Scrum Teams

While the scheduling and feature creep problems continued to build on themselves, it became necessary to steal people from teams working on "future" projects to help make up the deficits in the "now" projects. Eventually, the endangered products were completed on their adjusted schedules. However the cost was pretty steep in that new development projects, which were needed sooner rather than later, were completely halted. The effects of this are still reverberating as the re-assembled "future" project teams are now trying to catch up on all the work that didn't get done while they were helping the "now" teams. Once again, schedules didn't shift to adjust to the lost work time, so the catch-up cycle has started again, with the "future" teams robbing people from the "now" teams so that they can meet the schedules that they've been committed to.

# 4 The Road to Recovery

Even with everything that's been described to this point, the one advantage of failure is always the opportunity to use it as a blueprint for success. You've already done it the Wrong Way and you intimately know that path. Simply not repeating those mistakes should get you moving in the right direction. Taking the time to really learn from and improve on the things that went wrong should dramatically increase your chances of "getting it right" the second time around.

Within our organization, we've identified several things that we could do that should fix many of the issues described and also generally improve the functionality of the Scrum teams.

## 4.1 Full Time Evangelist

It was clear that, as soon as our full time Agile consultant left, things began to disintegrate. Having an on-site Evangelist is critical. This person should exist only to regulate and reinforce the process, make sure the teams are "doing it right", ensure cross-team consistency, be vigilant, interact with management to set expectations, and teach new hires as well as those in the company who have not been trained in Agile process and Scrum.

The primary goal of this position should be to audit processes for all the Scrum teams. This would be done by periodically checking on the groups and verifying that they aren't wandering too far off the path or trying to "WaterScrum" or "Scrumfall". Although teams need to be able to adapt processes into something that works for them, everyone needs to be going about Agile the same general way. This keeps things consistent internally and makes it easier to regulate releases and Agile process. It also eases the transition for anyone moving between teams.

Additionally, support and consultation are a huge part of the role of the Evangelist. The person who fills this position needs to be available to help teams troubleshoot Agile and Scrum questions and mentor them through the transition. Not only does this person audit the teams and regulate process, but they also help guide the teams back to the right path. Any thought that this role is, in any way, punitive is entirely incorrect. The Evangelist is more of a "spirit guide" if you will, leading teams down the righteous path to Agile salvation.

If at all possible, the Evangelist should be chosen from outside the company. Ideally a new-hire or a consultant. When the position is an internal hire, they often come with a whole set of biases about projects and people that may influence how different teams are treated. If you must choose your Evangelist from within the company, said person should be as neutral as possible and also completely passionate about Agile and Scrum. You definitely want to choose someone that can fire up your teams and not only get them excited about Agile but keep them interested for the long haul.

## 4.2 "Weed the Garden"

As part of the work of reinforcing Agile, it is important to remove "non-believers" from authority positions on Scrum teams. They most certainly cannot be Scrum Masters or Product Owners. "Non-believers" should be broken up and distributed across the organization so that there isn't more than one or maybe two on a single Scrum team. If they are maintained at minority levels on teams made up of "true believers", then their potential for damaging the process can be minimized and eventually they may even convert. However, it is clear that they cannot be allowed to congregate and build a "Scrumfall" team.

## 4.3 "Seed the Garden"

Just as you don't want too many "non-believers" on a team, you can use "Agile Ninjas" or "True Believers" to bolster teams that are having trouble. Sometimes adding a person who really understands the process can help a team find their way out of the weeds. If it doesn't impact momentum, subbing a Ninja in as either a temporary or more permanent Scrum Master may really speed up the journey to team enlightenment. But, even if you can't get your True Believer into the SM role, just getting them on the team should be a big help. Our most successful teams had either an "Agile Ninja" or "True Believer" as their Scrum Master.

## 4.4 Spread the Fun Around!

Everyone on a Scrum team should have a chance to be the Scrum Master, even if it is just for one Sprint. Taking a turn as the Scrum Master gives people the opportunity to experience the job for themselves and develop some empathy for the people who do it regularly. Being a Scrum Master is often not as easy as it looks and doing the job allows the other team members to find that out first hand. Ideally, this will make team members more receptive to the Scrum Master (or the Agile Evangelist) and more likely to acquiesce to their requests.

## 4.5 Re-Commit to Agile

To really make Agile successful, the whole process needs to be restarted and must involve the entire organization at appropriate levels. At the very least, there should be company-wide training (or a series of trainings) on the Agile methodology and the Scrum process. This will assist the rest of the company in their interactions with the Scrum teams and will help them to set expectations appropriately. It will also free the teams to actually work correctly within the Agile framework instead of having to spend precious time fending off inappropriate requests from those that don't know any better. Additionally, the Scrum teams should go through the process of re-chartering, so that they can re-calibrate and restart their internal processes and get themselves back on the right path. Finally, there must be an increased focus on tracking velocity, team output, and quality of output. These things are amongst the primary "rewards" of adopting Agile and, as such, should be watched closely and nurtured.

## 4.6 Focus on Consistency

Once the Scrum process is restarted, it is imperative that every effort be made to keep things consistent and the environment "distraction-free". Management rule breaking (such as temporarily re-directing team members to other projects or interrupting Sprint work for special projects) definitely should not be tolerated. If a problem arises that has special circumstances, figure out solutions that solve the issue within the existing Scrum teams or form a new team, but don't disrupt team membership or focus to solve a one-time problem. One possibility would be to form a "SWAT" team that's dedicated to dealing with real time issues.

## 4.7 Build Teams that Make Sense

When initially assembling Scrum teams, try to choose workers with complementary skillsets and (if possible) personalities. This process is described in more depth in Bhushan Gupta's paper, Waterfall to Agile: Flipping the Switch (Gupta, 2012). In addition to having appropriate personalities and skillsets on a team, good communication is imperative. Placing a person or two in the group that has excellent communication or facilitation skills is always a great idea.

Building a good team is not an accidental process. Sticking folks together and just "hoping it all works" doesn't always turn out the way you want it to. If you start with a concerted attempt to create a strong group, it increases your chances of success. Ideally, you want to build teams that will bond fast, reinforce each other, communicate well, and work hard together.

## 4.8 Keep Your Teams Together, unless.…

Once your Scrum teams are assembled and working well together, leave them alone. Don't change membership unless it is necessary. As mentioned previously, making large changes (e.g. swapping 2 or 3 people from a team of 8) to your teams can really de-stabilize them and cause a loss of velocity and momentum. Additionally, it may take two or three Sprints before the reconstituted team is able to integrate the new membership and return to its former productivity levels. That said, people can get bored or feel trapped if all they ever do (or are expected to do) is work on the same thing with the same people. Be prepared to occasionally shift ONE worker off of a team and into a different group. In most cases, swapping out one person doesn't cause a lot of angst and a single new worker can be brought up to speed pretty quickly.

## 4.9 No Oddballs!

Don't allow "oddballs" onto your Scrum teams. It's easy to feel like everyone in the department should be assigned to a team, but that isn't necessarily the case. "Oddballs" are people with specialized skillsets or talents that aren't needed full time on a particular team (e.g. a UI design specialist.). Having these folks on a team full time can be a distraction and a frustration to the rest of the group. “Oddballs” don’t tend to be equipped to contribute appropriately to the team, so they often have “no status” to report, or they get loaned out to other groups so the status they report has nothing to do with their team’s activities.

If you have “Oddballs”, put them in a “rental pool” so that they can be “checked out” by teams when needed and then returned to the pool when their specialized work is complete. This allows all groups to have access to these unique resources without the burden of integrating them full time when they don’t need them. It also helps out the specialized workers in that they can do the tasks they’ve trained to do without having to worry about being forced to do work that they’re not qualified for or interested in doing. Finally, it avoids the worst case scenario of having highly paid, highly specialized employees sitting around doing nothing while waiting for an appropriate task to come along.

# 5 Thoughts from the Road

While doing research, it became clear that the problems we experienced were not unique. A lot of companies go through similar pains. It seems like the most successful organizations were the ones that kept trying, did a lot of thoughtful retrospection, and were finally able to internalize the tenets of Agile. There were interesting parallels between this company and other groups that are described in the online materials cited below.

Specifically, along with everything that has been discussed thus far, it is clear that we made pretty much all of the mistakes with all the resulting problems described in Martin Fowler’s article, “Flaccid Scrum” (Fowler, 2009). We didn’t pay attention to applying “strong technical processes”. We did spend time talking about technical process, but we didn’t make any real decisions and we definitely didn’t implement processes in any consistent way. The tendency was to adopt a process, which some number of teams (but not all) would take on and follow until some newer and more interesting technology came along. For instance, during the first two years of Agile implementation, we went through at least 3 different configuration management tools which were implemented, adopted, and then abandoned at random points in the teams’ projects. That’s crazy!

As described in the article, the end result of behaviors is that it became more difficult for us to add features, because our code bases were too tangled up with each other. We built up a huge Technical Debt – to the point that we ended up (for a while) with a policy that 20% of all Sprint work would be aimed specifically at tackling it. And, as predicted, our Scrum went “weak at the knees”. We officially became the poster children for “Flaccid Scrum”.

Digging deep into the source of all the issues we’ve seen, it has also become clear that per Robert Martin’s blog article “The *True* Corruption of Agile” (Martin, 2014) we failed to do a thorough job of adopting the real culture and practices of Agile. Initially at least, we took on the practices of Agile and Scrum and followed them religiously, but the entire group didn’t take on the Agile culture. Even though some of the individual teams did culturally adopt Agile, the entire group did not, which made it very difficult for the truly Agile teams to work with the fake Agile teams.

Without a real and solid Agile culture to operate within, the teams that believed in the philosophy and process were forced to make compromises to meet non-Agile expectations. There were a lot of conversations that started with “I know this isn’t the right way to do this but….” This single sentence fragment was the leading edge of what Robert Martin believes is the biggest problem within Agile – the “*elimination of practices*”. A perfect example of this is the teams that quit tracking velocity because it didn’t make sense. Top down scheduling, which is very much a non-Agile behavior, made velocity irrelevant. In a nutshell, the lack of adoption of a true Agile culture led to the rejection of the Agile processes and Scrum.

# 6 Where the Rubber Finally Meets the Road

One of the really nice things about the particular software company in question is that open-mindedness is encouraged and anyone with a good idea is welcome to put that idea out for discussion and implementation. That is how we originally came to implement Agile in the first place; Waterfall processes weren't working well for us and a couple of small teams had individually implemented it with reasonable success. It seemed like implementing Agile across the entire Engineering team might get us some pretty significant gains in productivity and quality. And, as I mentioned earlier, we have seen real benefit from making the switch to Agile, so the effort was not entirely wasted.

Happily, there has also been some recognition that things haven't worked as well as we'd hoped and we've started slowly working on fixes to some of the more obvious problems that have been described. There has been some "selective weeding" on the scrum teams to move some of the less amenable people into places where they aren't as harmful. We're also working on implementing a single CSM tool for the entire team to use. Once this step is complete, we will begin making the transition to CI/CD in earnest. As part of these efforts, QA is working closely with their development counterparts to implement smart and reasonable automation to complement the application code that is being developed.

Thus, change is definitely in the works. One of the reasons that this particular paper was internally approved was so that it could be used as an actual blueprint for improvement. As such, the content of this document is not theoretical, it is part of a greater plan to re-implement Agile process and Scrum properly and get things moving again in the right direction. With luck, there will be a presentation at the 2015 PNSQC detailing "How It All Worked Out."

# 7 Conclusions

Once you've learned all the mechanics, put good processes in place, and constructed strong, functional teams, developing within the framework of Agile is easy. However, not lapsing into old habits, especially within a resistive environment, is much more difficult. Constant vigilance is a necessity. More importantly, it's vital that your organization makes a top-down commitment to Agile and Scrum. Even if the entire company doesn't implement it, there must be a commitment to learn about the process, what to expect from it, and how to interact with it.

Without top-down Organizational buy-in, your team will constantly fight misaligned expectations, mis-understandings, and general issues that are the result of trying to align two methodologies that don't understand each other.

Within your Agile implementation, strive for simplicity and for processes that make sense. When a procedure becomes onerous, nobody wants to follow it or keep up with it. At the same time, don't try to re-invent the wheel. The processes are what they are for a reason. They work and they work well. Redefining roles and procedures only confuses things and hides problems. If things aren't working the way they are designed to work, there's a problem that needs to be addressed.

Ultimately, Agile isn't difficult, but it does require that your organization make the effort. As with all other development methodologies, if you don't try to do it right, you will fail. It's that simple.

# 8 Acknowledgements

My sincerest thanks go out to the following people, without whom, there is no way this paper would have been written: My husband Justin who, even though he doesn't understand Agile or really even technology, gracefully puts up with my babbling about it and putting gigantic sticky notes all over the house. Scott Richardson, who provided valuable feedback just when I needed it. Aaron Getz who pointed me at the articles I cited in this paper. My manager, Roger, who let me write this paper when I really should have been doing actual work. My PNSQC editors, Karyn and Bhushan, who gave me valuable guidance and who taught me that Scrum and Agile aren't the same thing (DUH!). And, finally, my Scrum teams – you know who you are. Without all of you, I wouldn't have learned what I learned and I would most definitely not have been able to write this paper.

A final and special thanks goes out to the company that I work for. They stood behind me and supported me in this endeavor, even though I cannot reveal their identity. Thanks!

## References

Fowler, Martin. 2009. "FlaccidScrum" Martin Fowler Website, article posted January 29, 2009, http://martinfowler.com/bliki/FlaccidScrum.html (accessed April 25, 2014).

Gupta, Bhushan. 2012. "Waterfall to Agile: Flipping the Switch". Proceedings from the 2012 Pacific Northwest Software Quality Conference (PNSQC) http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf (accessed August 13, 2014)

Martin, Robert. 2014. "The *True* Corruption of Agile" The 8thlight Blog, entry posted March 28, 2014, http://blog.8thlight.com/uncle-bob/2014/03/28/The-Corruption-of-Agile.html (accessed April 25, 2014).

# Success Through Failure: How We Got Agile to Work on a Distributed International Team

**Erin Chapman**

erin.chapman@gmail.com

## Abstract

Multinational teams are becoming the standard as companies compete globally. We hear stories of US-based teams who prefer Agile development practices but who encounter growing pains, including cultural and time zone differences, when they add overseas contributors.

Engineering teams want to collaborate and work well together regardless of how they may be distributed. This case study delves into the problems we experienced on one team spread across two continents (in United States and India). To bridge the differences and solve these problems, we had to take the time to really communicate with one another about how each site interpreted Agile concepts, what they saw as the main obstacles to development, and how we could best solve problems without sacrificing our priorities.

The solution was not for the team members at one site to issue a process to the entire team; instead we collaboratively reached a solution that ensured communication between sites, decreased turnaround time, improved quality, and received buy-in from the whole engineering team. Some of the practices we have put into place could assist other teams in being successful working across development sites.

In the end, we found that Agile development can work internationally if the process has enough structure, team knowledge is made more explicit, and everyone's voice is heard.

## Biography

*Erin is a software design engineer at Tektronix where she is involved in software development, scrum process management and software testing. She has an M.S. in Computer Science from Portland State University and B.S. degrees from Rensselaer Polytechnic Institute in Mathematics and Computer Science.*

# Introduction

You can find many companies switching to Agile methods these days from the traditional waterfall process. Tektronix is one of these companies, having made the switch several years ago to Scrum. While we've generally had success with our collocated teams, regardless of their location, it's been a mixed bag when it comes to distributed teams, even within the same time zone. This mirrors the experiences that we hear from other companies and developers.

## 1.1 Distributed Scrum Teams

When Scrum was first proposed, it was recommended against using it for distributed teams. One can find many industry experts, such as John Puopolo arguing that Scrum should only be used with collocated teams [4]. Over the years, it has become more of the norm for a company to have multiple teams across multiple sites working on the same project. With this has come a variety of models for managing a distributed team. Teams can choose whether they have a separate scrum team at each site or not, how many Scrum Masters and at what points the teams coordinate. You will most likely see one of three versions of scrum: One Scrum run across all sites, one Scrum at each site that are coordinated via a **Scrum of Scrums**, or one Scrum at each site that are not coordinated and run independently of each other. Each method has its strengths and weaknesses, and which method you choose depends on the specific circumstances of the project [1].

## 1.2 International Teams

Adding the international element additionally complicates the distributed Scrum team arrangement [5]. Each site may have a different management structure, as is the case at Tektronix. Along with this, the sites may have different work flows that do not mesh with each other or complicate the coordination between teams. At Tektronix, it is not uncommon for management at one site to have a different definition of what progress looks like and push their team to meet their definition, while another site has a completely different view. When working in a single code base, this can cause the team to disagree with each other on all parts of the development process.

Additionally, **communication** can more easily fall by the wayside with international teams. When working across time zones where teams do not have a common day, the **increased turnaround time** and **mental distance** will decrease the amount of time teams spend talking to each other. Tools such as instant messaging, e-mail, and teleconferences can help overcome these issues, but only if the team is cohesive at the start. [3].

## 1.3 Historical Trends at Tektronix

Tektronix has been slowly making the switch from a traditional waterfall style process to Scrum over the past several years. The transition has been somewhat piecemeal, and how Scrum is run depends heavily on the department. Within the department that the XYZ team worked, all of the scrums were run independently and with varying levels of coordination. The Scrum teams at various sites often did not communicate until close to a customer release date, when major issues would often appear. Scrum teams at the same site sometimes coordinate their work, but often only because the Scrum Master was the same person for both teams.

During a sprint, code could be delivered to the development branch in the source control repository at any time, without any verification testing. All testing was done in a specific hardening sprint as the product release occurred. Frequently, incomplete features would have to be removed near release time due to too many last minute issues discovered in the code.

# Our Story: The XYZ Team

When the XYZ team was formed to work on a new product, they were given the opportunity to change not only the tools that they used to do their job, but also develop a new work flow for development. The XYZ team consisted of two groups at two different sites: the Beaverton team who would be working on the core infrastructure for the new system and the Bangalore team who would be writing algorithms to do data analysis. The original plan, as laid out by management, would be that the Beaverton team would get a head start on creating the infrastructure and once a sufficient base was in place, the Bangalore team would be spun off fairly independently on writing the algorithms.

As is often the case, the best of intentions often go awry. The Beaverton team imposed strict code quality standards on not only themselves, but the Bangalore team. The Bangalore team was brought into the project before the base infrastructure was stable. Both teams had very different development processes in mind and very different ideas of what is “agile”. The result was a **jarring discord** between the two teams that threatened to derail the project. This is a look back at how we got to that point and how we moved forward to become one of the best distributed, multinational teams at Tektronix.

## 1.4 Founding the Beaverton Team

The Beaverton team was started with four members to work on the core infrastructure of the new project, O, R, V and N. They were given the opportunity to choose the toolset everyone on this project would be working with. Given that freedom, they decided to aim for their dream build environment that would help the team reach high quality goals.

Historically, the development environment had been somewhat of a **free for all**. Each developer had a local copy of the code that they could work in, but everyone committed their code to the same branch on the server, without any gatekeepers as to what would and would not go in. One build would run at 6 PM PST with whatever was checked in and the results would be e-mailed to the team (except only one team member reliably read it).

Frequently, one person would check in their code and move on to the next thing they needed to do. The next morning the build e-mail would be deleted by everyone and no one would notice the broken code until the complaints built up. And that was if the build actually failed. Sometimes errors would only be discovered during final testing before a release and an entire feature would need to be pulled at the last minute.

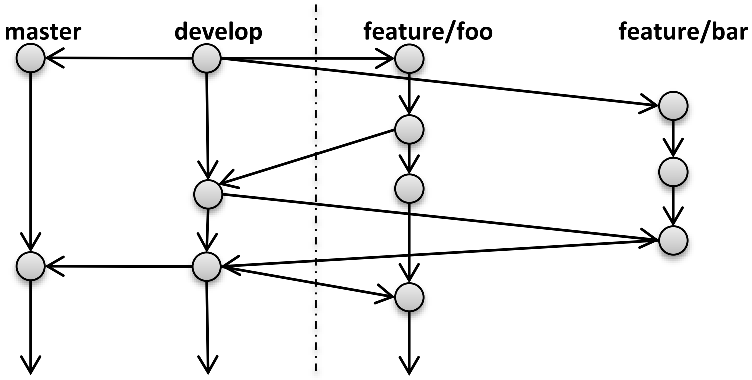
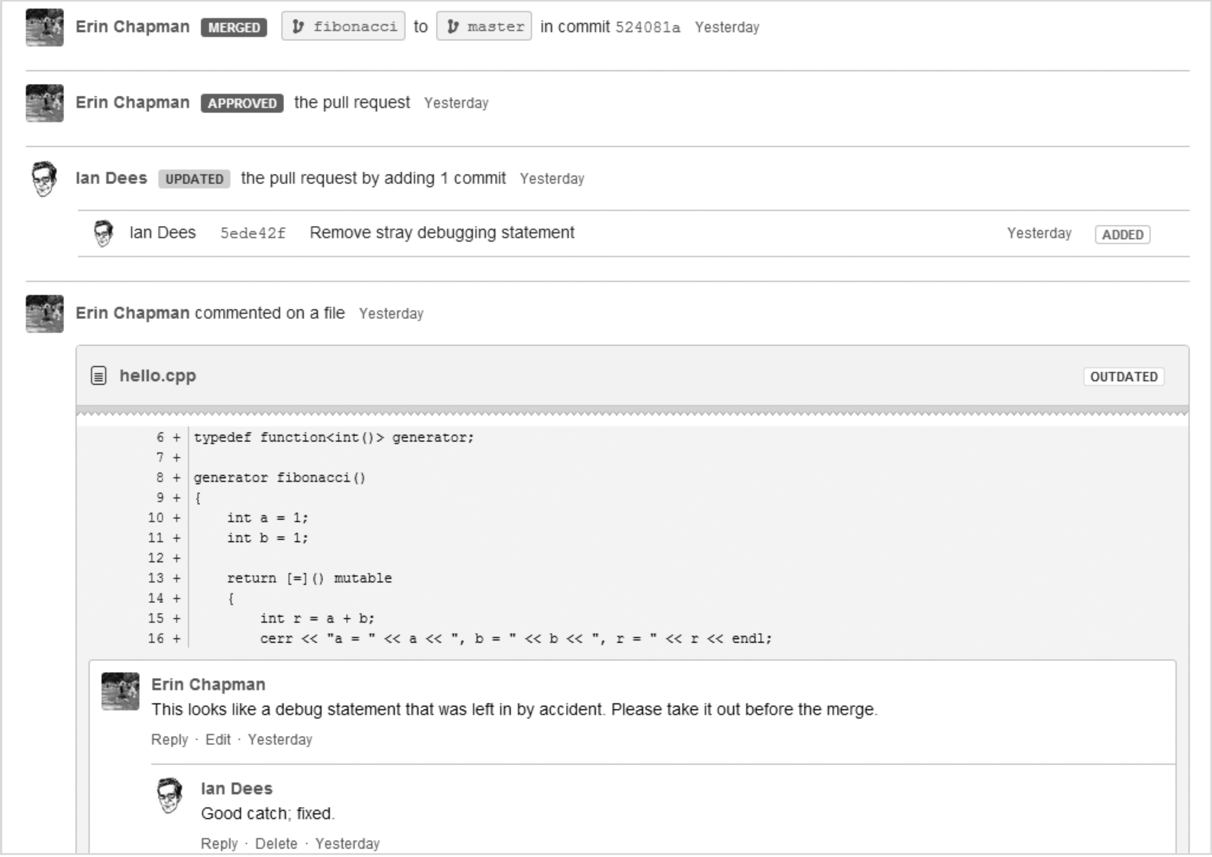


Figure 1: The branching model used to keep work in progress (feature branches right of the dashed line) separate from completed work (**develop**) and released work (**master**)

The Beaverton team wanted to get away from this mode and move to a model with continuous integration, testing and gatekeepers so they could produce a high quality product. They started by choosing Git for source control, with Stash from Atlassian to manage the repository. To preserve code stability, they used the **Git flow** [2] branching model to keep all work in progress in isolation. All work is done in a **feature branch** by the developers. Each individual feature has its own branch so it can remain in isolation until complete. When the feature is done, fully tested and meets all requirements, it is reviewed and merged into the develop branch. The develop branch contains all completed features that have not yet been released to a customer. Finally, develop is periodically merged into a master branch which is then released to customers. All code in develop and master was expected to be stable at all times, even during a sprint.

To prevent code from being released before it was complete, the Beaverton team adapted a **strict rule set** enforced by their tools. First, they restricted who had permission to push code to develop and master to a small set of people. This requires all merges to develop to go through a **pull request** in Stash (essentially a peer review). In Stash, no one is allowed to merge their own code. This allowed the team to insist that all code go through peer reviews, one of the best ways to catch defects early [6]. In addition to requiring a peer review, they also configured **Stash** to tie into the **Jenkins** automated build system. Jenkins built their code across three platforms (Windows, Linux and Mac OSX) and ran unit and functional tests across all platforms as well. If the build or any of the tests failed on any platform, Jenkins would report an error to Stash and block the merge until it had a successful build. The team also collectively decided that all comments in a **pull request** had to be addressed before the code was merged.



Erin Chapman MERGED fibonacci to master in commit 524081a Yesterday

Erin Chapman APPROVED the pull request Yesterday

Ian Dees UPDATED the pull request by adding 1 commit Yesterday

Ian Dees 5ede42f Remove stray debugging statement Yesterday ADDED

Erin Chapman commented on a file Yesterday

hello.cpp OUTDATED

```
+ typedef function<int()> generator;
+
+ generator fibonacci()
+ {
+     int a = 1;
+     int b = 1;
+
+     return [=]() mutable
+     {
+         int r = a + b;
+         cerr << "a = " << a << ", b = " << b << ", r = " << r << endl;
```

Erin Chapman
This looks like a debug statement that was left in by accident. Please take it out before the merge.
Reply · Edit · Yesterday

Ian Dees
Good catch; fixed.
Reply · Delete · Yesterday

Figure 2: A **Pull Request** in Stash showing a typical interaction

During this initial project startup phase, the Beaverton team experienced some growing pains. Initially the team was not collocated together in the building. O and R, having worked together on a previous project, were in cubicles somewhat close to each other, while V and N were on another floor. While the team was seated apart from each other, they found it difficult to frequently work together as a team or in pairs. This set up some initial habits of working as individuals on tasks that created barriers to communication later. After several months on the project, the team was able to **collocate** in a section of the building together. Once they  started sitting together, there continued to be communication issues for a short period. Not all of the team members would attend the **Daily Stand Up** and there was a lack of awareness as to what each person was working on.

After a month of rough communication, the team reaffirmed the importance of the Daily Stand Up, which was attended by all team members going forward, and also started a group work time in the afternoon where the team would leave their cubicles to work around a table together.

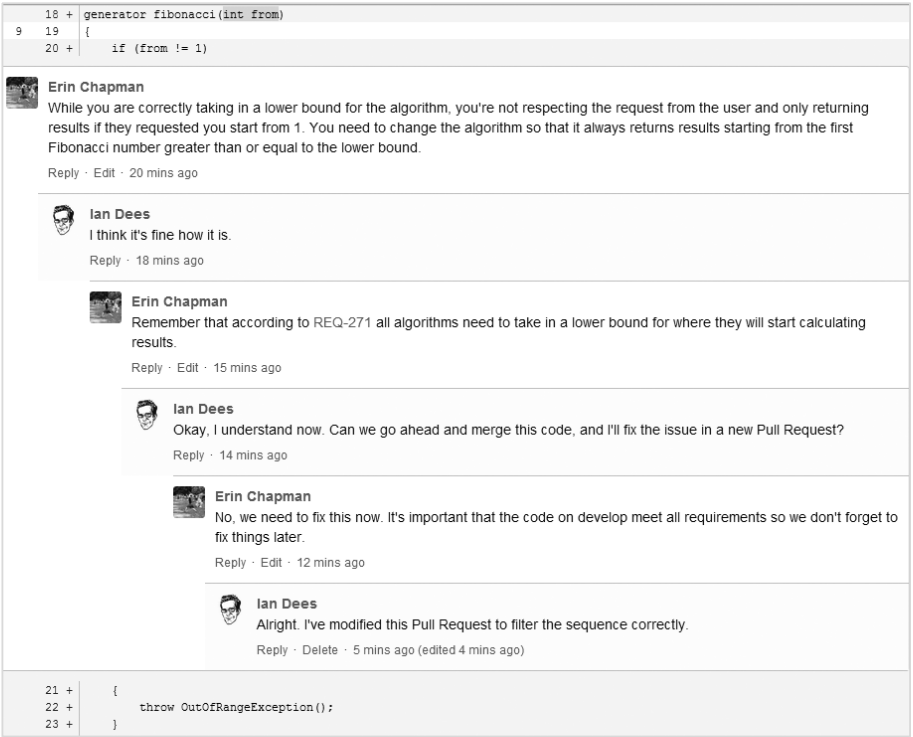
## 1.5 Bringing in the Bangalore Team

After the Beaverton team had been working on the infrastructure for roughly eight months, there was **pressure from management** to bring the Bangalore team (five members: Z1, F, C, E and Z2) on line to start work on the algorithms. Because the core of the project was still relatively unstable, the Bangalore team started working in their Git repository with the base system pulled in as a component. To get them started with this work, the Beaverton team provided a demonstration of the APIs and wrote several how-to documents on the architecture, requirements and data types.

The Bangalore team began their sprints in sync with the Beaverton team to work independently on the algorithms. During this time there was minimal communication between the two teams. N would attend a regular phone call with the Bangalore team but little of that information made it to the rest of the Beaverton team. The remainder of the Beaverton team had little or no contact with the Bangalore team.

## 1.6 Disagreement and Discord

After several months, the Bangalore team sent a copy of the algorithms in progress to the Beaverton team for a quick review. Upon receiving the code, the Beaverton team had **concerns about specific requirements** being forgotten or misunderstood, as well as **conformance to the team coding standard**. The Beaverton team e-mailed their concerns to the Bangalore team for revision.



```
generator fibonacci(int from)
{
    if (from != 1)
```

**Erin Chapman**
While you are correctly taking in a lower bound for the algorithm, you're not respecting the request from the user and only returning results if they requested you start from 1. You need to change the algorithm so that it always returns results starting from the first Fibonacci number greater than or equal to the lower bound.
Reply · Edit · 20 mins ago

**Ian Dees**
I think it's fine how it is.
Reply · 18 mins ago

**Erin Chapman**
Remember that according to REQ-271 all algorithms need to take in a lower bound for where they will start calculating results.
Reply · Edit · 15 mins ago

**Ian Dees**
Okay, I understand now. Can we go ahead and merge this code, and I'll fix the issue in a new Pull Request?
Reply · 14 mins ago

**Erin Chapman**
No, we need to fix this now. It's important that the code on develop meet all requirements so we don't forget to fix things later.
Reply · Edit · 12 mins ago

**Ian Dees**
Alright. I've modified this Pull Request to filter the sequence correctly.
Reply · Delete · 5 mins ago (edited 4 mins ago)

```
    {
        throw OutOfRangeException();
    }
```

Figure 3: An example of early interactions between the Beaverton and Bangalore teams

Around this same time, it was decided to pull all of the algorithms into the same Git repository as the core infrastructure. At this time, the Beaverton team started reviewing all of the pull requests created by the

Bangalore team. Because of their various quality concerns, the Beaverton team insisted on **reviewing all code** and did not give anyone on the Bangalore team merge permissions. When they felt the issues were not being addressed, the Beaverton team required any concerns to be addressed in the current pull request, rather than merging the code as-is and having the author make the necessary updates in a future pull request. The Bangalore team, meanwhile, wanted to push on bringing in an initial version of all of the algorithms and address all of the requirements and review comments after the first pass was complete. The Beaverton team, in part due to historical issues with incomplete features and having to remove work in progress from the code base immediately before a release plus a tight project schedule with little time to rework issues later, had very strong feelings against this particular work flow and stopped merging code. Because no one was willing to budge on their process, all work on the teams stopped and angry e-mails started flying back and forth between the two groups and their managers.

## 1.7 Time Out to Talk

As the e-mails escalated and no code was being merged, the team hit a hard stop for roughly a month where little work was completed. It was soon evident that this was not going to be solved via e-mail, so the Beaverton team started scheduling conference calls with the Bangalore team to discuss what was going on.

The first meeting between the teams was a review of the requirements, which it seemed like everyone understood but there was **disagreement on how the requirements were being implemented** by the Beaverton team. The Beaverton team, on the other hand, felt like those decisions were behind them and understood how that design had been reached, so they were eager to end discussion of the requirements and move on to the next agenda item. This particular issue would crop up several times, until the Beaverton team explained the various designs they had considered and why the particular design they had chosen was the only one that met all requirements. Once that understanding was reached, this particular issue disappeared.

The rest of the meetings all related to the **development process**. What emerged was that each Scrum team had a very different idea of what the development process was. The Beaverton team was looking for each piece of functionality (infrastructure or algorithm) to be **complete** before being merged into **develop**. To them, complete meant that it implemented all of the requirements, had unit and functional tests, used current C++11 best practices and passed a peer review with no one else on the team having concerns (admittedly, the Beaverton team set a high bar for both the Bangalore team and themselves). The short summation of the Beaverton team's idea of agile: only customer releasable code is in **develop**.

On the other hand, the Bangalore team wanted to create a **basic version** of each algorithm and then go back and **add additional features** one at a time to meet all of the requirements. Testing the algorithms would be done after all of the requirements were implemented. The Bangalore team also preferred automated testing to peer review. The team leads there felt that automated code review was better at catching issues than a review by other developers. There were also feelings that it was unfair that no one in Bangalore had permission to merge the code and only Beaverton team members did. The Bangalore team felt that only **master**, not **develop**, needed to contain releasable code and that being agile meant frequent, small pushes to **develop**, regardless of completeness of code. When release time came around, the completed code could be separated from the incomplete code and be merged into **master**.

The Beaverton team had strong feelings against this type of development process as we had historically run into problems come release time at Tektronix. After further discussion, we determined that the teams had different definitions of when an algorithm was done. This realization gave them a place to start in an attempt to find middle ground.

## 1.8 Process Changes

The Beaverton team started with two major changes. First, R took responsibility for the relationship with the Bangalore team instead of trying to pass information via N. Since R was the one reviewing all of the code, it made sense for her to be the one interacting the most with the Bangalore team, especially when it came to planning and communicating requirements. Because of the streamlined communication flow, the correct information made it back and forth between folks and R could confirm with the Bangalore team whether they understood what was being asked for.

As part of this, R made herself available to the Bangalore team for support. It was made clear that anyone could e-mail R with questions or concerns, but if it was felt that the issue could not be resolved via e-mail or code review, an on line meeting could be scheduled. At first, no one on the Bangalore team took R up on the offer, but after R started scheduling one on one meetings with folks on the Bangalore team when issues came up during code reviews, the Bangalore team got to the point where they feel comfortable putting meetings on R's calendar to discuss requirements, work in progress or issues that came up during a code review.

R and O also started phoning into the Bangalore team's **daily stand up**. After a while, it became clear that R and O were doing a scrum of scrum style meeting with Z1, so the team switched to doing a scrum of scrum meeting twice a week for just the three of them attending.

To help clear up the confusion about what was being looked for during code reviews, the Beaverton team created two **checklists**. The first checklist was a list of items they consider when reviewing the actual code, made as concrete as possible. This included items like using C++11 types, such as smart pointers, nullptr keyword and enum classes; using class APIs instead of writing your own function, correct usage of data types and writing Doxygen headers for classes. The second checklist included a list of unit and functional tests that would be used to verify the code meets requirements. After releasing these checklists, the Bangalore team reviewed all existing code, making huge changes and removing many of the concerns the Beaverton team had about code quality.

During this process, the Beaverton team also realized that they were not **recognizing the good work** done by the Bangalore team. While the Beaverton team is used to being critical of each other's work, they realized that just being critical of the code written by the Bangalore team may not go over the same as the relationship is not the same.

The relationship between Beaverton teams and Bangalore teams has always been a bit strained. The Beaverton engineers are typically recognized as being senior to the Bangalore engineers. Additionally, the Beaverton engineers tend to make the decisions and inform the Bangalore engineers what they will be doing. This leads to the Bangalore engineers feeling like they have little say in what happens. Coming in to a situation where you were working independently before, into one where not only are other people making the decisions, but they’re pointing out you are doing wrong, can be hard to take, regardless of where you are.

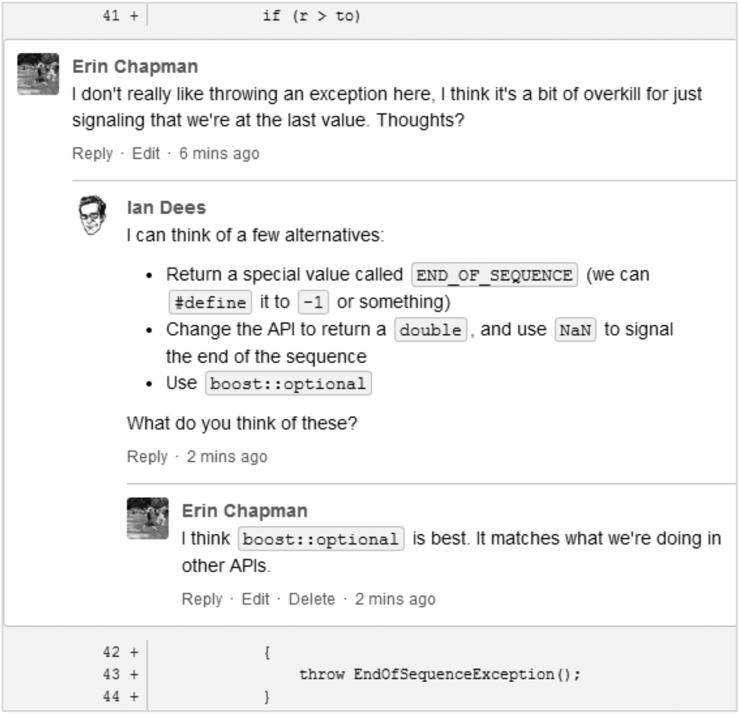
After recognizing how difficult this could be on the receiving end, the Beaverton team decided to make a concentrated effort to not just point out issues in **pull requests**, but also to point out things that were done well and thank developers for making changes or responding to comments.

## 1.9 Current Process

At this point, the Beaverton and Bangalore teams have a relatively smooth process in place that has been working well for over half a year. About a week before a new sprint, R, O, Z1 and A (the product manager in Bangalore) get together and discuss the upcoming goals for the sprint. O and R check if the Bangalore team is going to require anything in the next two to three months from the Beaverton team and let Z1 and A know if they're going to need anything from them. The two teams then do their respective sprint planning and send each other the lists of tasks pulled into the sprint afterward to confirm that both teams are still aligned. During the sprint, these four meet on the phone twice a week to discuss what each team is doing and make sure there are no issues or blocking items that need to be addressed.

Once the sprint starts, **early communication** is heavily emphasized. If anyone has questions about an algorithm or an architecture piece, they know the expectation is to ask questions before starting work. At this point, the Beaverton team gets questions from developers on the Bangalore team on a weekly basis (most importantly, before code is written or as soon as the question comes up, instead of waiting).

Both teams have discovered that **code reviews** in Stash are a great communication tool as part of their early communication efforts. If a Bangalore team member has a question on work in progress, or a Beaverton team member is filling out some architecture needed by the other team, developers have started opening **pull requests** before code is ready to be merged so that the teams can have a discussion in the actual code about what is being done. These have been some of the best conversations between the two teams, sometimes going back and forth for a week or more on design.



```
+            if (r > to)
```

**Erin Chapman**
I don't really like throwing an exception here, I think it's a bit of overkill for just signaling that we're at the last value. Thoughts?

Reply · Edit · 6 mins ago

**Ian Dees**
I can think of a few alternatives:

- Return a special value called `END_OF_SEQUENCE` (we can `#define` it to `-1` or something)
- Change the API to return a `double`, and use `NaN` to signal the end of the sequence
- Use `boost::optional`

What do you think of these?

Reply · 2 mins ago

**Erin Chapman**
I think `boost::optional` is best. It matches what we're doing in other APIs.

Reply · Edit · Delete · 2 mins ago

```
+            {
+                throw EndOfSequenceException();
+            }
```

Figure 4: **Pull requests** for design discussion between Beaverton and Bangalore team members are now common

Along these same lines, when a **pull request** is opened for a final review and merge, comments are no longer viewed as a negative thing. The team has had some **pull requests** remain open for days or weeks while debating some point or discussing requirements. If a complicated merge needs to happen or an API has changed, everyone trusts that others will help out. The attitude is no longer merge now and figure it out later, but one of wanting to reach a conclusion about what's best here. Questions or concerns about code are not viewed as a personal attack on the developer, but wanting the team as a whole to produce high quality work.

The biggest gain seen between the two sites is probably with how well the teams talk with each other. Every developer at each site is willing to talk with anyone else. E-mails with questions about requirements and code go back and forth on a daily basis. If a solution can't be reached via e-mail quickly or someone suspects that there is miscommunication, a conference call is rapidly scheduled. Surprisingly, it's not just the Beaverton team scheduling meetings with the Bangalore team anymore. The developers on the Bangalore team have reached a point where they feel comfortable setting up meetings of their own to discuss code.

Since the communication has become smoother between the two teams and the perceived code quality has improved, the Beaverton team has even reached the comfort level where they feel confident in assigning work to the Bangalore team. This is a widely divergent place from where the team was a year ago, and was mostly resolved by fixing various communication problems between the two teams.

# Lessons Learned

The failure to do use Scrum successfully and produce high quality code really came down to a number of **communication** problems, both between Beaverton and Bangalore and within the Beaverton team.

- **There Can Never Be Too Much Communication:** Scrum emphasizes communication as a major part of enabling the development of quality software. With the Beaverton and Bangalore teams, we learned that erring on the side of over communication, especially early in a project, can reduce the speed bumps hit later on. The Beaverton team adapted a policy of e-mailing the Bangalore team before making any changes that may affect them (since changes like that were typically one way on this project) and again when the changes were merged. Any information that the other team might want to know was e-mailed out immediately. Sometimes this produces a high rate of traffic in between the two teams, but there have been few surprises since then.

- **Learn How the Other Team Wants to Communicate:** Part of communicating with the other team, involves communicating with the other team effectively. This is a learning curve and involves adapting what one typically does. With this project, the Beaverton team learned that phone conversations get complex information over to the Bangalore team better than e-mails. They also learned that some of the more junior team members in Bangalore aren't comfortable asking questions during group conference calls, so they stay on line for a while after each call in case someone wants to instant message them with questions privately.

- **Get Everyone Comfortable With Talking:** The Beaverton team noticed that while they would say that they could be contacted directly by anyone with questions it rarely happened. Instead, they had to make it clear that no one would get in trouble for contacting them and they'd prefer to set up phone calls to talk with the Bangalore team in the mornings or evenings. A concentrated effort had to be made initially to communicate one on one with the developers on the Bangalore team, even if it was just a quick check in e-mail, until everyone was comfortable initiating a conversation

  Along these same lines, noticing who is comfortable talking to who plays a large roll. Two of the developers are more comfortable talking with R than with O, so O keeps one on one contact short and positive, while R handles more complex issues or things that may be perceived as being critical..

- **Personal Relationships Matter:** Initially there were no relationships between the Beaverton and Bangalore teams; no one had any idea of who was on the other end of the phone or keyboard. This made some of the conversations between the two teams very confrontational. Once the two teams made an effort to get to know each other, the conversations softened. On conference calls now the teams ask about families, talk about vacations or other activities in their colleague's lives. Knowing who is on the receiving end directly effects how you interact with them.

- **Defined Structure for the Process:** While Scrum typically prefers a loose, less defined process, when working across sites, a more defined process may help. Explicitly listing out the scrum of scrums process, plus the checklists for code reviews, and putting subtasks on tickets for “talk to developer X on the Beaverton team before starting work” has helped to set the expectation for what is being looked for by the Beaverton team.

# References

[1] Anderson, Rick D., 2013. “Scrum in a Land Far, Far Away… (or Using Scrum Across Dispersed Sites).” *31$^{st}$ Annual Pacific Northwest Software Quality Conference*, pages 79-91. http://www.uploads.pnsqc.org/proceedings/PNSQC_2013_Proceedings.pdf

[2] Driessen, Vincent, 2010. “A Successful Git Branching Model.” http://nvie.com/posts/a-successful-git-branching-model/ (accessed August 3, 2013).

[3] Kiesler, Sarah and Jonathon N. Cummings, 2002. “What Do We Know about Proximity and Distance in Work Groups? A Legacy of Research.” http://www.ece.ubc.ca/~leei/519/2002-ProxmityDistanceWorkGroup.pdf (accessed May 5, 2014).

[4] Puopolo, John P., 2007. “Be There or Be Square.” Scrum Alliance. http://www.scrumalliance.org/community/articles/2007/august/be-there-or-be-square (accessed May 16, 2014).

[5] Simons, Matt, 2002. “Internationally Agile.” InformIT. http://www.informit.com/articles/article.aspx?p=25929 (accessed June 2, 2014).

[6] Sommerville, Ian, 2011. *Software Engineering*. Boston: Addison-Wesley.

# The Agile Advantage: How Agile Integration Improves Outcomes For Software as a Service (SaaS) In The Corporate Environment

**Greg Spehar MBA, PMP, PMI-ACP**

spehargreg@yahoo.com

## Abstract

More and more enterprises are adopting software-as-a-service (SaaS) “Cloud” applications. These applications are developed and maintained with an Agile Methodology, increasing their quality and performance. How can an Enterprise integrate these SaaS systems in an effective manner without significant risk to the organization and a decrease in application/process quality? An agile data migration and process integration methodology can be defined to ensure a high probability of success with the highest quality delivery when integrating a new system into a corporate environment.

Many of the new applications being developed today are cloud-based SaaS[1] applications developed using agile methodologies that are used by multiple customers or clients. These SaaS applications need to be integrated into the corporate environment in a logical and methodical manner utilizing agile methodologies to ensure that the client organization can meet their performance needs. The problem that needs to be solved is the solution a SaaS application has defined will only include 50% to 80% of the processes implemented by the client organization. For the client organization to fully utilize the SaaS software all impacted processes must be migrated or changed. With the use of change management techniques as well as a thorough understanding of the SaaS agile process, an agile data migration and process integration methodology can be defined to ensure a high probability of success while minimizing the disruption.

This approach was utilized on a Portland based organization, which allowed them to establish their go live date and meet that date with minimal process impact. This agile integration approach can be standardized and utilized with any organization that is struggling to tame their data migration and process integration efforts into a vendor’s agile SaaS implementation.

## Biography

*Greg Spehar is a Senior Project Manager at CorSource Technology Group Inc. working on agile projects that bring customer value developing and integrating applications. Over the past 15+ years he has been a leader in consulting projects that range from software development to integration of Software as a Service (SaaS) solutions. He has worked with non-profits, healthcare, state/federal organizations, apparel manufacturing, energy, financial and other organizations.*

*Greg has worked extensively developing and honing his skills in Project Management, Project Leadership and Change Management. He has a BS in Aerospace Engineering from Purdue University, his MBA from University of Texas at Austin and Erasmus University in the Netherlands, and he has earned his PMP and PMI-ACP from the Project Management Institute.*

# 1 Introduction

When working on a migration project there are many challenges and issues that need to be addressed that need to be organized in a fashion to ensure that the project has the highest probability of success. In this paper we will look at using the agile philosophy and approach as a defining framework since in most cases the relationship between the vendor and the client cannot be held within a fixed contract when the vendor is using an agile development approach for their product.

The first step out of the gate that needs to be established is the agile notion of Customer Collaboration over Contract Negotiation[2]. If this approach is not taken from the start the entire project will be in jeopardy. The primary reason is inherent in an agile project, as the train of software updates will not be integrated in a linear process. We shall see in this paper, the integration effort must mirror the development effort to have a significant chance of success. To best illustrate this approach we will first decide on the type of SaaS vendor configuration we will explore in this paper. There are several kinds of SaaS vendors and they can fall into the following three categories as shown in Figure #1.

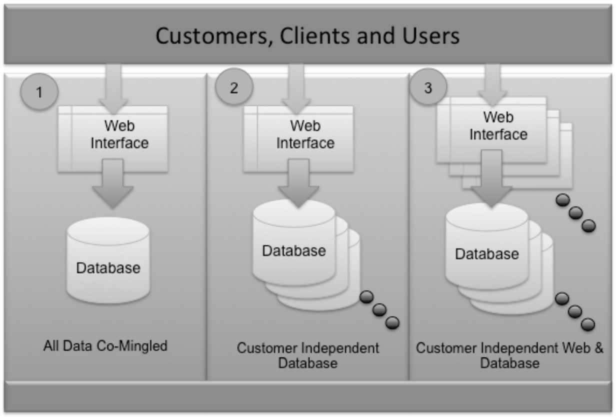

Customers, Clients and Users
1
Web Interface
Database
All Data Co-Mingled
2
Web Interface
Database
Customer Independent Database
3
Web Interface
Database
Customer Independent Web & Database

Figure #1: Different Integrating Approaches for SaaS Projects

For this paper we will focus the second type of SaaS vendor, The Customer Independent Database Vendor, this vendor has separate databases but uses a common web client code. Within this construct we then have several constraints and levels of freedom at the same time. Those constraints are as follows:

1) Independent Databases allow daily data dumps into a “data warehouse”
2) Not all client change requests will be implemented on face value
3) System Configuration will predominate

As stated before, we have a SaaS system where system changes are managed through the use of an agile approach. Because of the approach dictated by the agile process we have a level of freedom as follows:

1) Contract Negotiations are focused on larger grained features amounting to a dozen or so capabilities (No matrix details are defined).
   - Giving the vendor and client leeway in the kind of features and changes that will occur to meet the needs of the client.

2) Contract Payments are broken in terms of milestones over a period of time with the go-live date plus some post go-live stabilization period amounting to the total system implementation cost.
   - Giving the client the leverage necessary to ensure features are in place before go-live and post go-live.

3) Process mapping meeting change management is driven by backlog priority instead of details in the contract.
    - Giving the client extensive opportunity to change and shift priorities as they discover what is important and what is not important as they learn more about their business as time passes.

Within this context we can define project goals that include the following:

1) Data Migration and Configuration
2) Process Evaluation and Change Management
3) Reporting and Data Integrity
4) Portal Integration into Existing Website/s

We can then define some ongoing project maintenance processes that ensure the project's success:

1) Stakeholder Management and Communications
2) Risk Management and Action
3) Action Management and Follow-up

This paper will not cover in detail the known processes around Project Management body of knowledge[3], but articulate the most critical factors of the PMI process or changes to the PMI process that are relevant to an agile SaaS project's success and quality. Let's dive into the project communications processes first.

# 2 Communications

## 2.1 Stakeholder Management

As we all know, when managing a critical project, the onboarding and management of the stakeholders is the #1 job of who ever is responsible for making this kind of change happen[4]. The key items to ensure your success (See Figure #2) with your Stakeholders are the following:

1) Set up routine meetings and status reporting
    - Consistent communications over time ensures comfort for leadership
2) Never delay or do anything that would harm the sense of trust
    - Do not delay or sugar coat bad news if it happens!
3) Establish the most critical aspect of the project
    - They are after new revenue, operational savings, etc.
4) Focus on what can be done and how resources are managed
    - Don't bring problems to solve, bring solutions!

Meetings & Status
Ensure Trust
Define the Critical Aspect
Bring Solutions

Figure #2: Stakeholder Management Responsibility

The communications on this project will be also critical in the area of the Stakeholders and the immediate team that is working on the migration effort. These groups are the most important to maintain a good solid face time and building an honest open relationship. This kind of environment promotes the immediate identification of risks and then their possible resolution. This component of trust is key in an Agile SaaS Integration effort. Finally, the rest of the organization that will be impacted from this change will be explored within the Change Management section.

## 2.2 Risk Management and Action

Risk management is another area that is not taken as seriously as some people might think it is needed[5]. The challenge which can appear in an integration project that is run in an agile like method is the necessity to maintain and mange the risks that will inevitably appear on the project. The need for the project team to identify them early and as quickly as possible allows the team to make important decisions that can have many months impact or ultimately kill an integration effort. The steering committee working with the team must trust that the team will provide the necessary decisions to ensure the technology can perform with their organization in the best way possible; even if that includes the difficult decision of cancelling the project entirely.

Typical risk management tools can be made and defined, but for this project the risks that should be closely identified are as follows:

1) Client Project Team Personnel Risks
2) Change Management Risks for Client
3) Staffing Issues with Vendor
4) Delivery of Feature Sets in a Timely Fashion (Burn rate)
5) System Performance once the system is in place
6) All normal technical challenges such as bandwidth, desktops, etc.

## 2.3 Action Management and Follow-up

The most interesting aspect of any integration effort with systems that are pre-defined and configured on site is the lion share of the activities required at the client site will be focused on the simple question:

Can the software do what I need it to do to meet my process objectives?

If not, what changes (Requirements) can we define to help the Vendor conceive of the most effective approach within their software?

NOTE: In this example of a Customer Independent Database Vendor, the vendor is considering many other vendor’s needs and the impact to them when using the software.

The majority of the actions being defined for the client team will be around how to ensure that the new system can meet their demands at the process level. If not, what steps are needed to augment the system and will those changes meet the needs of the client processes. In some cases the changes will NOT be adequate and the VENDOR may not be willing to make the changes. In these situations the client must be prepared to change business processes to meet the methods that are outlined by the vendor or make a decision to abandon the effort.

Experience has shown that the latter option never happens and a compromise can be developed to the benefit of both parties. The usual approach we have been witness is the willingness of the client to change their processes if the Vendor can state a good case as to why the software cannot be changed. In most cases the client’s proposed changes fundamentally change the assumptions of the SaaS system and thus will have a wide impact to other clients the SaaS vendor is attempting to satisfy. Usually in these kinds of situations the client discovers that their process approach will be more effective if they adopt the Vendor’s technique.

All these items need to be in place to communicate and track the details to lead and ensure the project success. The following sections are the constructs of the overall effort.

# 3 Migration

## 3.1 Data Migration and Configuration

As with any integration of a system into an environment, the data becomes one of the key factors of success (See Figure #3). The movement of the old data into the new system includes several areas of concern:

1) Data that does not translate cleanly into the new system
2) Configuration data
3) Leaf Data - Dropdowns
4) Data that is new or is in places hard to get

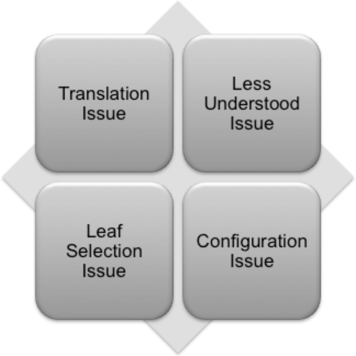


Figure #3: Data Issues to Resolve

Data that translates cleanly will not be data that you will normally have challenges with and in most cases your technical team will have an easy time getting to that data and translating it successfully into the new system. The challenges include data which one might think needs to be translated in the same way turns out to not be the same in the new system. Most of the vendors will assist in this work of ensuring the data is matched perfectly, but the general approach to management of data transfer is the agile approach to putting things up in the system as soon as possible for the client and their business people to review.

The following is a general approach to pushing data up to a test system over time:

1) Test System Release 0 – Generally has a few hundred records or less for as much of the system data that is easily identified.
2) Test System Release 1 – This release will consist of over 70% of the necessary data if possible. This will be the first time the users of the system will see how things work and begin to really evaluate their processes.
3) Test System Release 2/3/4 – These releases are all being performed to hit the goal of near 100% of the data being scripted.
4) Go-live Migration – Usually occurs over a weekend or an entire week depending on the vendor at hand.

The most critical notion around this approach is that the client should be purchasing 50% to 80% of the features[6,7] already in place, so moving 50% to 80% of the data should be something that is very doable in a short period of time. (Within a 6 month period of time realistically.) Once this Release 1 happens, the most severe risk has been managed and the team can dig in to fully understand the gap that exists in data and system functionality.

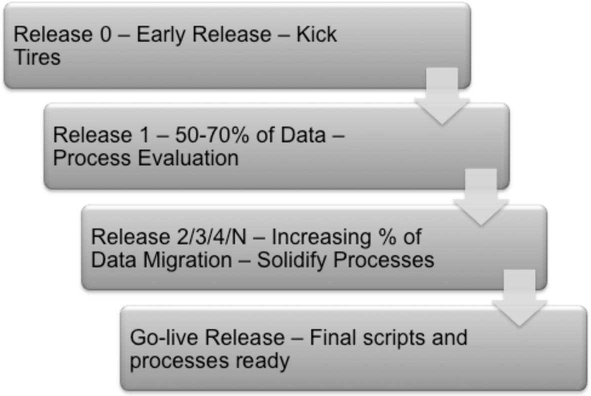

Release 0 – Early Release – Kick Tires
Release 1 – 50-70% of Data – Process Evaluation
Release 2/3/4/N – Increasing % of Data Migration – Solidify Processes
Go-live Release – Final scripts and processes ready

Figure #4: Agile Based Migration of Data Approach

One of the most notorious and misunderstood component of SaaS integrations in this realm is the necessity of configuration management. Since the system is a shared system, there may be hundreds or thousands of configuration settings that need to be turned off or on depending on your specific set of use cases. The vendor will do their best up front to manage this transition but will not fully understand the ramifications of the setup till the software is put under stress.

When evaluating process steps and process approaches, the project team should always ask the vendor integration team if the software does something one way or can it be done another way. That maybe the case as the configuration setting needs to be updated properly. In many cases there will be a feature request that comes back to the team as a configuration setting. As systems can become complex, the SaaS vendor may not remember all of the configuration settings; only when the problem is encountered can the developer discover that a configuration setting exists for the solution. NOTE: As many of the SaaS systems grow organically over time, the usage of the systems becomes more important than the documentation of the system, hence the sheer lack of documentation for larger SaaS systems will occur since the system changes much too rapidly for any documentation to be maintained cost effectively.

The management of Leaf Data is also an activity that is not entirely dealt with in the context of its importance. The need for the integration team to properly populate the data is clear. Mischaracterized leaf data results in improper loading of data that requires that the process be repeated. Proper review and care should be taken in analyzing all relevant dropdowns and the resulting updates in the configuration files should be effective. A couple of important notes on this effort:

1) Try to do it right the first time
    - Improperly loaded leaf data can have an impact to migration if discovered late in the process, more due to the business agreements required to finalize the data to be used.
2) Make feature changes right away if there are system limitations
    - These structural changes can take some time to make and these risks need to be mitigated early in the process or you will find you will need to delay the project since groups of people are not able to agree on a specific way of representing the leaf data.

Data that is new or in places hard to get will be the most challenging of the data to migrate. The project should expect this kind of data to be discovered and migrated in some cases near the start of the process. The lion share of this challenge will appear at the tail end of the project when the “Necessary” data is discovered in new processes and new excel spreadsheets that “Have to be integrated” into the system at the last minute. A gentle question should help establish perspective, “If it was so important, why are we only discovering it now? And if so, can it not wait till after the integration is complete?”

More importantly this discovery tends to fall into the general overall challenges of unknown processes, new processes or little understood processes that will require the creation of the transaction data as well as the creation of new configuration changes and drop down leaf data. These items should be properly evaluated and determined to be included or not in the overall integration process. We will explain more in the process evaluation and change management section on how that can be done within this agile approach without derailing the project.

## 3.2 Process Evaluation and Change Management

The most telling issue of any integration project is the ability for the team to properly evaluate their processes to ensure that the team has the ability to implement these processes into the new SaaS software. There are several schools of thought around process re-engineering efforts but they fall into the following two groups[8,9]:

1) Perform process re-engineering after the migration has happened
2) Perform process re-engineering before the migration happens

Most people move their decision making to #2 since they think that there is a better chance of making true change occur since a new system is being implemented. This makes sense at the surface but we should also consider the other perspective that the additional activity of re-engineering while switching software systems is one fraught with risk.

There are three re-engineering approaches that can be utilized:

1) Make re-engineering efforts prior to migration and implement processes as best one can with the old system
2) Make re-engineering efforts during the migration efforts
    - Only effective with organizations that are able to absorb the risks of broken processes and significant change management endurance. Organizations that are very stable and people have embedded themselves into their processes will find that the other two options may result in better performance.
3) Make re-engineering efforts after the migration is over and endure the rework that might be necessary to change the software

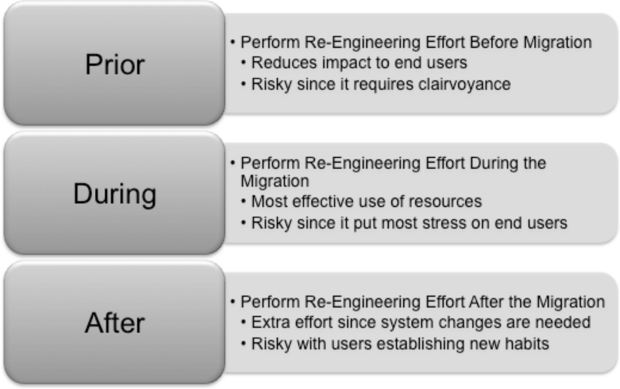

Prior
• Perform Re-Engineering Effort Before Migration
• Reduces impact to end users
• Risky since it requires clairvoyance
During
• Perform Re-Engineering Effort During the Migration
• Most effective use of resources
• Risky since it put most stress on end users
After
• Perform Re-Engineering Effort After the Migration
• Extra effort since system changes are needed
• Risky with users establishing new habits

Figure #5: Re-Engineering of Processes Options

Either path results in necessary changes to the vendor SaaS system and the need for a way to evaluate the importance of those changes. The first step in defining the methods to building a good product backlog for the vendor to manage is to not think of the Agile SaaS Integration effort as a waterfall effort. The reason being is that as time passes the client and vendor will have more opportunity to understand what is working in the system and what is not working in the system. This agile flexibility that can be experienced should become a pattern that will be adopted throughout the project. An approach that can be adopted (Figure #6) is were there are several levels in which the system can be evaluated and they are as follows:

1) First pass evaluation – Does the system meet the most basic features?
    a. Generally occurring during “Test System Release 0”
    b. Focused on finding show stopper features
    c. Opens door to contract negotiations if major issue found
2) Multi-Pass Evaluation of Feature Changes
    a. Generally occurring with “Test System Release 1 through go-live”
    b. As-Is process evaluation
    c. To-Be process evaluation
    d. Training Documentation (Also workbook)
3) Fully Acceptance Testing Walkthrough of all To-Be documents
    a. Generally occurring with “Test System Release Pre-go-live”
    b. Last determination for Go/No-Go Decision

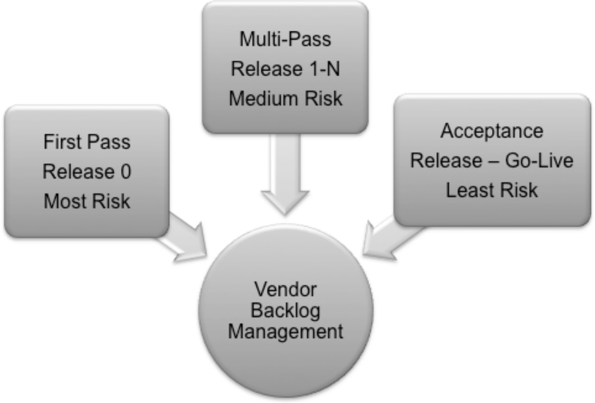


Figure #6: Process for Eliciting Backlog Changes to Vendor

In this approach there is the notion that large high-risk issues be discovered up front and properly scheduled for system updates or the determination that the risks are so great that the project be terminated. The later passes discover more and more detail upon each pass giving users a more structured process for evaluation and reporting of issues.

Once these issues are identified they are logged and given a priority that should be at three levels of importance:

1) Level 1 – Must have for go-live, NO WORKAROUND available
2) Level 2 – Must happen soon after go-live, temporary workaround available
3) Level 3 – Can happen anytime after the go-live event, workaround in place

Of all of the lesson’s learned for this approach, this factor was the most invaluable since it allowed the technical staff to push back on the business staff to challenge the need to call something a Level #1 when it is clear that it is a level #2 or #3 (Figure #7).

Given this model, it became clear that not all of the processes would be managed to the level that the business would have liked or wanted. It also allowed the business to understand that some things had a higher need than their own specific change and they should be prepared to live without the change for some time if possible. This approach then applies nicely to those last minute process discoveries that always appear at the last minute as projects are about to go-live. The Integration team will then have the chance to challenge these processes so that the business can make a good decision to attempt to integrate these new processes or not prior to the go-live event.

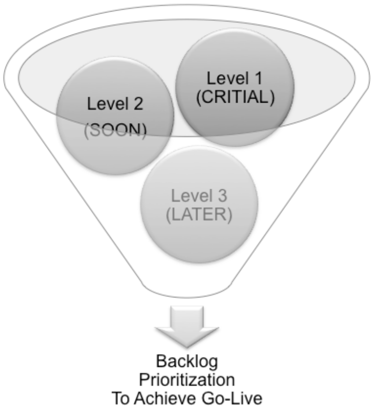


Figure #7: Backlog Prioritization to Chase Go-Live Within an Agile Framework

This brings us to the final challenge in this area that was faced. That is the need to document what the new process is and then to train the staff on those new processes and software usage. The change management technique that is mostly followed by most organizations is called the ADKAR method (Awareness, Desire, Knowledge, Action and Reinforcement). In the project that utilized this approach we found the staff to be well prepared to engage the development of the training materials and we found the staff continued mentoring the use of the new system well after the go-live event occurred.

For the project that was part of this effort, the team selected the training and documentation approach of writing just one set of documentation that would be used for training and as a desk reference. These word documents had all of the processes for each of the areas with the steps to perform the work. With each of the project releases revealing more and more of the application that could be used to define client processes (Figure #8). Where there were gaps in the software, since it was still being developed, work-arounds had been put in place to ensure the go-live could still happen. Those workarounds were documented right into the training materials, with the future process also documented, but caveated with the message to not use until the change has taken effect. Once the change went into play and was verified, the documentation could be quickly updated to show the new process.

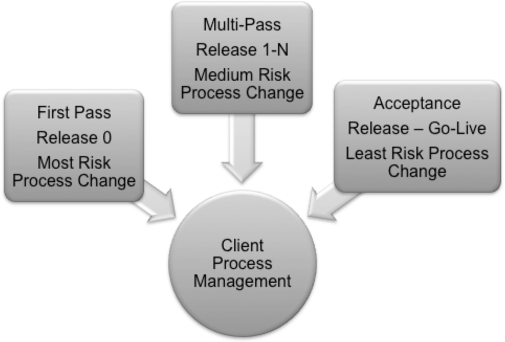


Figure #8: Process Risks Discovered and Challenged Early and Often

Since there is so much moving at the same time, every sprint cycle the client had the chance of re-organizing the product backlog and making changes to what they wanted the software to do and how they were going to manage their existing processes. In some cases deciding not to implement changes that they thought they needed since they decided to take a different approach with their processes. ***And, most importantly, in all of these cases we did not have to negotiate a change to the contract.***

## 3.3 Reporting and Data Integrity

Another important aspect of a system is the reporting that is generated since the reports give the management team the level of detail they need to make the right choices and ensure that the organization functions properly.

Many of the system changes that occurred at this level came from the re-organization of the required reports and in some cases the need to capture or look at the data in a different way. These changes, pushed through the product backlog allowed the team to make changes based on need.

As such the team was able to quickly focus on delivering the reports that were required for go-live. These changes reduced the total number of required reports to a more manageable level and the final reports required for go-live became a more meaningful number and in the end more useful.

In the implementation that utilized this approach in this paper, the Vendor had implemented a “day-later” data dump process. So every night, the database was delivered to the client’s ftp folder and the client then pulled their database into their own environment. This allowed the client to customize specific reports that did not need real-time data. Additionally, many of the Vendor Reports could be generated into excel spreadsheets that allowed additional reports be generated.

Within the context of flexibility, the integration team was able to utilize an agile approach to prioritizing and building the necessary reports to ensure that the team could meet their go-live commitments. This approach additionally, gave the team options when the vendor would decline to update a report in a given manner due to other customer conflicts.

Finally, as part of the post-go-live effort, the team performed several assessments to understand what Data Integrity tools should be created to ensure that the data going into the system is not a problem as the system expands and people use it in ways that was intended and not intended. Due to the fact the system was a multi-user system the team found several cases that if the process was not followed and some of the screens where used improperly, some data integrity issues would appear. Instead of trying to get the vendor to change their interface (Which they most likely would not do) they could generate scripts that would run nightly or weekly to identify situations where these issues would appear and the staff then could correct.

## 3.4 Portal Integration into Existing Website

Most systems now days have Portal integrations and existing websites that represent what they do as a company. These external systems will now have access to screens that can update the content that might otherwise require a phone call to someone to update the information directly into the system. During most projects you will find once the Agile SaaS Integration approach proves to be effective, the Web Integration team will seek out the same effective process and procedures to ensure their sites go-live in the same successful manner. Additionally, if there is a tie to the SaaS system to the customer’s web site, there may still be changes that are needed in the SaaS system to meet their web site design needs. These web-based changes can also be defined and prioritized and feed to the SaaS vendor within this agile framework.

# 4 The Real World Example

## 4.1 Non-Profit Integration of Granting and Donor Management System

The project, where we used this framework for the first time, was a large non-profit which had selected a SaaS vendor to host and provide all of their non-profit system needs. We had to move all of their funds management and donor relations systems to the new product. The challenge with this migration was that they had about 50-60+ people that had to be on the new system all at one time. These people had been doing the same job in the same manner for years, so change management was going to be key the project's success. The framework that we have defined in this paper was utilized and was effective in not only identifying the necessary changes in the system, but also proved to be an effective manner in which to build project champions within the steering committee.

The use of the release cycles for migration allowed the core team to see the changes happening to the system and allowed them to peal the onion back on their processes as they explored and defined their "As-Is" and "To-Be" Processes. This effectively allowed the team to develop a level of comfort and confidence that was translated to the executive team and the end users that eventually had the responsibility of implementing the new system.

Overall the project was extremely successful in managing the organization to their new system due to the competence of the entire team. But the most effective skills that helped to perform the necessary risk management to ensure the project's success was as follows:

1) Project Managers – Seasoned and pro-active project managers for the Client and Vendor
2) Technical Resource – Effective senior technical resource is a must to package and deliver data
3) Business Analyst – Process Based – To help define the documentation & drive process change
4) Business Analyst – Report Based – To help manage the technology and report delivery

Finally, the full support of the Executive Team and their engagement once issues are raised for their attention cannot be missed. Any organization in the midst of change cannot succeed if their leadership is not an interested, motivated and engaged party in the effort. This Agile SaaS Integration Methodology allowed that to happen on this project with the ever-growing system and confidence of the people involved in the overall effort.

# 5 Conclusion

This resulting Agile SaaS Integration framework provides the basis to utilize one of the most revolutionary approaches to software development, the agile methodology, within an integration environment to almost guarantee success, or at the very least push the most risky components into the spotlight to either address them early or challenge the project and organization to consider delaying or cancelling the proposed project. The failures of projects are mostly rooted in the project's inability to reveal the risks that challenge the project and the methods in which those risks are evaluated. This framework provides some of the first documented steps in defining an Agile SaaS Integration process that can best guarantee an effort rooted in establishing and maintaining quality throughout the risk management effort where the resulting effort can put the least amount of stress on the organization while delivering the value that the project set out to achieve.

# References

1. Williams, Alex. 2013. “Forrester: SaaS And Data-Driven “Smart” Apps Fueling Worldwide Software Growth.” techcrunch.com, January 03. http://techcrunch.com/2013/01/03/forrester-saas-and-data-driven-smart-apps-fueling-worldwide-software-growth/ (Accessed August 22, 2014).
2. Cunningham, Ward. 2001. “Manifesto for Agile Software Development,” http://agilemanifesto.org (Accessed August 24, 2014).
3. Project Management Institute. 2014. “Project Management Book of Knowledge Guide and Standards,” http://www.pmi.org/PMBOK-Guide-and-Standards.aspx (Accessed August 24, 2014).
4. PMI’s Pulse of the Profession In Depth Report. 2013. “The high cost of low performance: The essential role of communications,” http://www.pmi.org/~/media/PDF/Business-Solutions/The-High-Cost-Low-Performance-The-Essential-Role-of-Communications.ashx (Accessed August 24, 2014).
5. Hamilton, Byatt and Hodgkinson. 2011.“Risk Management and Project Management go hand in hand.” cio.com.au, May 03. http://www.cio.com.au/article/385084/risk_management_project_management_go_hand_hand/ (Accessed August 22, 2014).
6. ZDNet/Topics. 2013. “Cloud: How to do SaaS Right.” Zdnet.com, March. http://www.zdnet.com/topic-cloud-how-to-do-saas-right/ (Accessed May 03, 2014).
7. Gilbert, Jody. 2013. “Executive Guide to Best Practices in SaaS and the Cloud.” Zdnet.com, March 01. http://www.zdnet.com/executive-guide-to-best-practices-in-saas-and-the-cloud-free-ebook-7000012032/ (Accessed May 03, 2014).
8. King, Julia. 2012. “The rebirth of re-engineering.” computerworld.com, September 10. http://www.computerworld.com/s/article/9231002/The_rebirth_of_re_engineering (Accessed May 03, 2014).
9. Gartner/ IT Glossary. 2014. “Business Process Re-Engineering (BPR),” http://www.gartner.com/it-glossary/bpr-business-process-re-engineering/ (Accessed May 03, 2014).

# Improving R&D Productivity by Triangulating Failures

**Richard Léveillé, Synopsys Inc.**

Richard.Leveille@Synopsys.com

## Abstract

Software developer productivity is vital to an organization's ability to meet its customers' needs. This presentation focuses on an innovative solution to optimize the regression testing cycle. Using the principles of triangulation to the software domain, we are able to identify the cause of the regression failures, shorten the test and debug loop, and enable quicker failure resolution. This paper presents background on the code integration method, describes our use of the Triangulation system of failure analysis, and provides insights of the impact of the solution in developing predictable and high quality software releases.

## Biography

*Richard Léveillé is a professional with over 20 years of experience in commercial software development, Software Quality Management Systems, and Software Release Management. Over the years, he has held positions as developer, Project and Program Manager in various organizations, as well as Quality Assurance Manager with experience managing ISO Certification programs. Currently, he is Senior Manager Corporate Quality and Program Management at Synopsys. In this role, he has had direct involvement in process design, Customer Satisfaction Programs, and promoting software development best practices in the organization.*

*Richard graduated as an Electrical Engineer from Sherbrooke University, Canada. He holds certification from Stanford University as a Stanford Certified Project Manager (SCPM).*

*He is a member of the American Society for Quality (ASQ) organization.*

# 1 Introduction

With constant pressure for faster development and additional functionality, a software developer's productivity is paramount to an organization's ability to meet the needs of its customers. A product must be kept current with customer needs and adapted to emerging technologies.

In the context of EDA (Electronic Design Automation) applications, there are numerous challenges in delivering on the promises of predictable and high quality releases:

- Applications range in size from ½ MLOC (millions Lines of code) to more than 20 MLOCs
- Size of development teams range from 20 to more than 250 individuals
- Worldwide teams perform around-the-clock development.

To address these challenges, an organization has a responsibility to constantly evaluate opportunities to best meet expectations from the customers while keeping a highly productive development environment. A structured and methodical approach is necessary to create conditions for effective and efficient development leading to a successful release.

As we consider the entire development life cycle, we realize that testing still represents a major portion of the development schedule. Actually it is estimated at ~30% for multiple stage testing [1]. Despite best efforts to design-in quality, software development still relies on testing of all sorts to validate the implementation. Opportunities for productivity improvements are numerous in that space and many avenues of investigation can be pursued.

By examining the various tasks involved in testing, we have found that the Test and Debug activity a developer performs is repetitive, time consuming, and most conducive to automation. Reproducing an error is always the first step in debugging. The task requires recreating proper conditions, extracting results, and comparing with a baseline. There is one set of tests in particular, regression tests, that always run on a known configuration and are totally under the control of the software development team.

This is where the concept of Triangulation comes into play. "Triangulation" [2] is a method used to locate an object by correlating information from different angles. In the context of software development, and specifically when we consider failure analysis, triangulation can help locate the cause of the failure to speed up defect remediation.

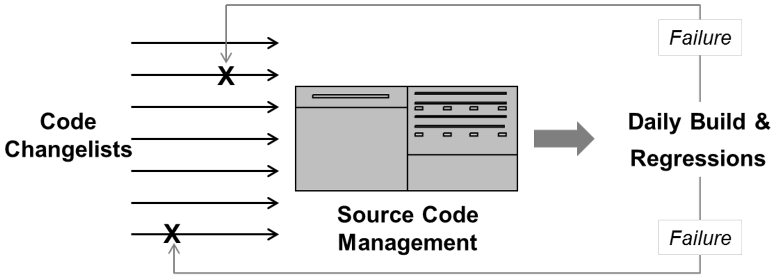


**Figure 1 - Build and Regression Failure Tracking**

In order to accelerate defect location and resolution, a triangulation system, consisting of software process analysis tools and methods, is used to identify the "bad code changelist" that may cause build failures or regression failures, as illustrated in Figure 1. A "changelist" [3] is a set of code changes made in a single checkin the Source Code Management System (SCM). An internally developed tool, called Tracer, was developed to perform failure triangulation and thus improve the efficiency of the daily Build & Regression cycle.

This paper outlines how the development environment ties to productivity, describes the triangulation system, its flow and implementation, and ultimately, how it tangibly contributes to higher productivity for the developers.

# 2 Conditions for High Developer Productivity

As we indicated above, the focus is to improve developer productivity. To do so, the organization must provide the right environment and tools to enable it. This is accomplished by providing favorable conditions and by removing obstacles for efficiency. We are focusing here on two conditions supporting high productivity: **Stable code** and a **quick feedback loop**.

## 2.1 Continuous Code Integration Model for Stable Code

Business needs demand faster releases and the capability to be more responsive to customer requests and issues. This requires high predictability for delivery schedules and, therefore, constant monitoring of the state of the code.

With many developers aggregating code changes in the same development branch, it is important to ensure stability of the code at all times. The changes from one developer might affect the whole team and slow down progress. For that reason, and to minimize the impact of large code merges, the Continuous Code Integration Model [4] is adopted.

Every developer performs development in their own "sandbox." As soon as the development is complete, they check the code into the main code branch. Before a developer checks in code, a set of strict checkin criteria must be met. The earlier the changes are integrated with the rest of the code, the sooner its effect is factored in and any issues addressed. It is the goal to minimize large code changes being checked in at once because it usually imposes a clean-up phase and affects the productivity of the entire team.

At the product level, daily build and regression runs are performed. Results are posted and the status of the branch is constantly assessed. The goal for the product team is to keep the branch very stable as measured by a regression pass rate between 98% to 100%. In some cases, the team will allow passing rates to slip down a bit lower prior to the alpha release date milestone, but the criteria go up quickly for the Beta milestone.

Those are challenging goals but it has enabled product teams to release stable software quickly if required. The release criterion is 100% regression passing rate. If there is a need to release, a focused effort to close on fewer regression tests makes the task manageable and achievable within a reasonable and predictable time. This is a great contributor to improved responsiveness.

The success of this approach depends on a developer's discipline to fix regression failures as soon as they are discovered. As part of the development discipline, the expectation is that a developer's first task of the day is to review assigned regression failures and take action. Test results are made available to the developers via a web application that consolidates all assigned issues for each developer.

## 2.2 Short Feedback Loop for Developers

One condition essential for high productivity is to provide a short feedback loop on any task so that remediation, if necessary, can take place before any ripple effect can be triggered. In a development environment with a quick feedback mechanism, developers can immediately see the effect of the changes they make. With feedback in minutes instead of days, developers can confidently address any remaining issues and proceed to the next assignment without having to constantly revisit their previous work.

A multipronged approach is taken to shorten the feedback loop and support the developers:

- Early validation of the checkin on the code build (part of triangulation mechanism)
- Validation of the effect of each checkin on the integrated code base (part of triangulation mechanism)
- Predictable and fast turnaround time for the daily Build & Regression cycle

The daily Build & Regression is closely managed to ensure all developers have complete results before the start of their day. A central code build takes place on a snapshot of the code and the complete set of regressions, consisting of thousands of test cases, are run against the various development platforms. This operation establishes a new and fully qualified baseline for the next day of work for the entire development team.

The Tracer tool described herein, which performs Build triangulation and Regression triangulation, is designed to further shorten the feedback loop for developers.

# 3 Effective and Efficient Triangulation

As mentioned, the software development progress relies heavily on the stability of the code and the reliable outcome of the Build & Regression process. Any failures must be quickly dispatched and resolved in order to keep all members of the development team productive. Therefore, the goal of the automated tool is to locate the point of failure to notify the developer who can best address the issue.

To achieve this objective, the Tracer tool interacts with the Source Code Management system (SCM) and the Build & Regression system. As illustrated in Figure 2, Tracer essentially monitors the code changes submitted throughout the day to improve success of the official daily build. Tracer also performs analysis of the regression test results to identify the source of failures.

Tracer includes two components: Build Tracer and Regression Tracer.

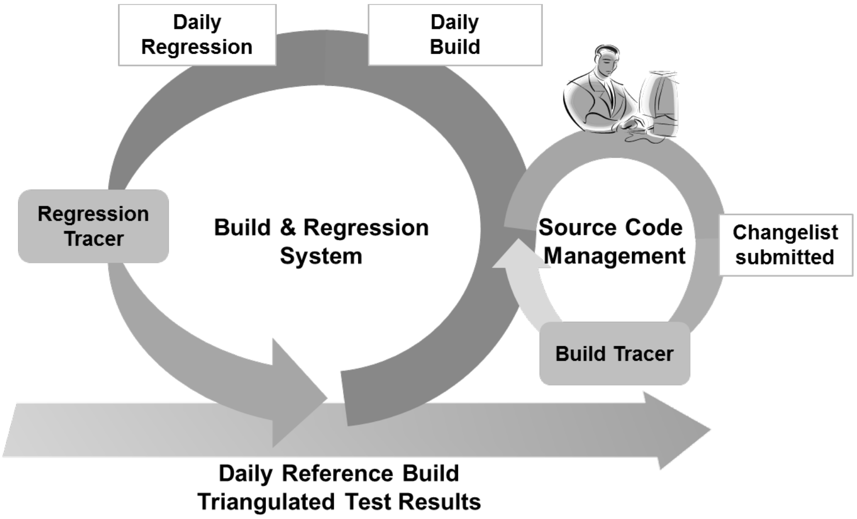


**Figure 2 – Tracer Solution as part of Continuous Integration Model**

**Build Tracer** builds every changelist as they come in to make sure it won't break the daily build on the Integration branch. If a failure is detected, Build Tracer sends email to the person who checked in the code.

**Regression Tracer** uses two sources of input:

- Daily regression test results from regression tool
- Executables for every changelist saved by Build Tracer

Regression Tracer compares today's daily regression test result with yesterday's result. For every new failure detected, Regression Tracer invokes the test cases that failed and runs the tests against all executables built from each changelist of the day. Regression Tracer can then point out suspicious changelists and notify owners.

Let's examine in more detail the logic of the Build Tracer and Regression Tracer as key components of the Continuous Integration process, and the respective results.

# 4 Build Triangulation

Build Triangulation supports two main objectives:

- Achieve 100% daily build success
- Detect build errors early on and notify the developer who checked in the code

The Build Triangulation workflow, illustrated in Figure 3, is essentially a four-step process. Build Tracer performs the following functions:

- Starts an incremental build at every changelist as soon as the code is checked in

- Detects any build errors
- Notifies success or failure to the developer who checked in and other stakeholders (e.g. integrator, branch manager)
- Maintains record posted on web pages

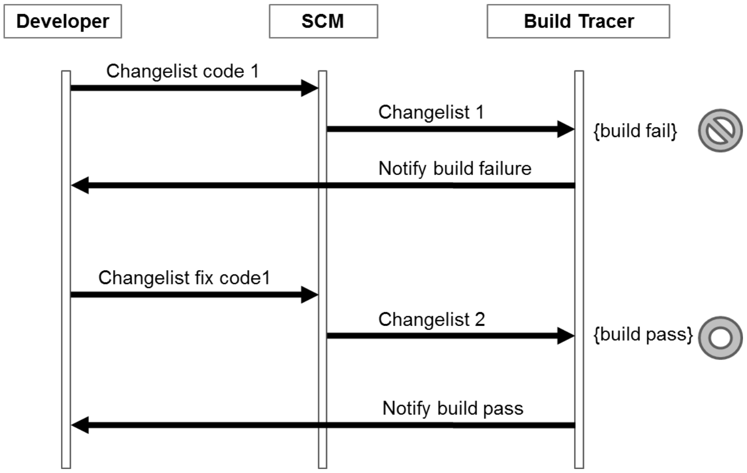


**Figure 3 - Build Triangulation Workflow**

The code branch is constantly monitored. An email notification is sent as soon as the incremental build fails. The developer can then take immediate action to correct the failure before the daily build cycle takes place. Build Tracer contributes to higher daily build success rate by ensuring all changelists have been validated prior to the daily build.

Each successful incremental build is kept for later use by the Regression Triangulation system.

## 4.1 Results from Build Triangulation

The effectiveness of Build Tracer can be measured and its impact observed. Table 1 below profiles a sample of products with key indicators to assess effectiveness:

- Volume of changelists in the code branch
- Number of changelists identified as faulty by Build Tracer
- Percentage of daily builds prevented from failure by use of Build Tracer

**Table 1 - Build Triangulation Effectiveness**

| | 3-Month Period | | |
|---|---|---|---|
| | **# Changelists** | **% Faulty Changelists** | **Prevented Daily Build from Failure** |
| Product 1 | 592 | 4% | 26% |
| Product 2 | 1675 | 1% | 21% |
| Product 3 | 5771 | 2% | 70% |
| Product 4 | 1826 | 2% | 33% |
| Product 5 | 1398 | 1% | 19% |
| Product 6 | 1121 | 4% | 35% |
| Product 7 | 1121 | 2% | 28% |
| **Average** | | **2%** | **33%** |

Looking closely at the results, it is reassuring, first, to observe we only have a small percentage of bad changelists. We note that the number of bad changelists does not necessarily match the volume of changelists. Some of the large teams with hundreds of developers actually have a smaller percentage of faulty changelists than smaller teams with ~30 developers. The rigor of the pre-checkin practice is a better indicator of the frequency of bad changelists.

The other observation is the fact that, even with a small percentage of bad changelists, the potential for disruption and destabilization of the code branch is significant. It only takes one bad changelist, without the Build Tracer in place, to make the official daily build fail. On average, Build Tracer actually prevents 33% of daily builds from failing. In other words, if Build Tracer was not in use, 1 out of 3 daily build would have failed. This is a significant gain in productivity for all developers.

# 5 Regression Triangulation

The objective of the Regression Triangulation is to ensure maintaining a high Regression Pass rate on the code branch assuring continuous readiness for release. In order to do so, the Regression Triangulation must cope with complex test environment and the large number of regressions running on various platforms.

To put things into perspective, the typical product runs over 5k -10k test cases/platform. Some of the products exceed 35,000 test cases. With a goal of keeping the pass rate above 98%, it means that 200 test cases might be failing at any given time for a product with 10,000 test cases.

The Regression Triangulation workflow, illustrated in Figure 4, is initiated as soon as the Regression test results against the daily build are available. Regression Tracer performs the following functions:

- Compares today's daily regression test result with yesterday's result
- Selects test cases to triangulate
- Invokes these test cases upon all executables built from each of the day's changelists
- Notifies developer(s) who made the suspicious checkins

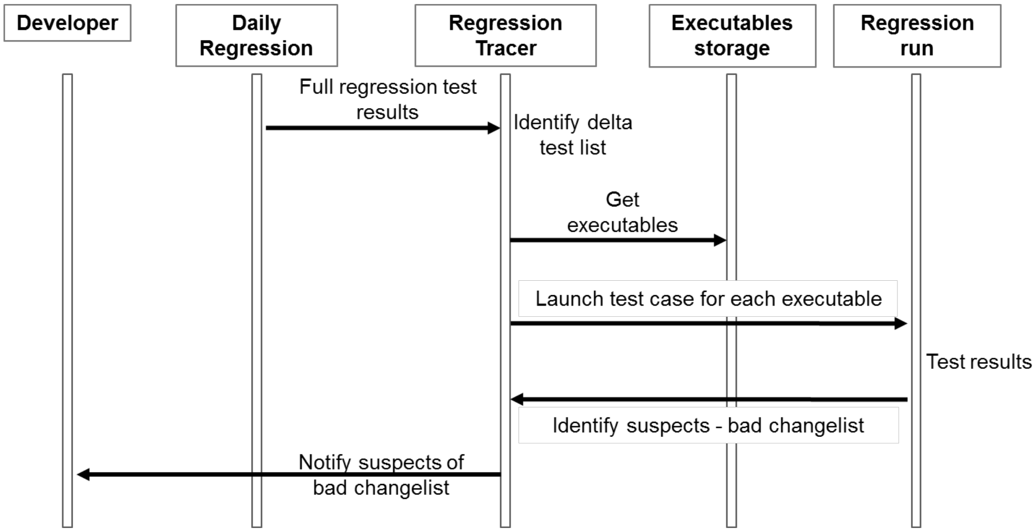

Developer
Daily Regression
Regression Tracer
Executables storage
Regression run
Full regression test results
Identify delta test list
Get executables
Launch test case for each executable
Test results
Identify suspects - bad changelist
Notify suspects of bad changelist

**Figure 4 - Regression Triangulation Workflow**

The first step is to determine the list of tests to triangulate. Only the new test failures for the day and test cases whose result worsened are analyzed; previously failed tests are already assigned and their resolution is tracked as part of the daily monitoring the product team performs. Table 2 below illustrates the logic to determine the list of tests, or deltas from the previous day's results, requiring triangulation. Applying this logic reduces the number of cases to triangulate for any given day.

**Table 2 - Logic for Tests to Triangulate**

| Daily build / Test Case Results | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 |
|---|---|---|---|---|---|---|---|---|---|
| **Yesterday's results** | **Pass** | **Pass** | **Pass** | **Fail** | **Fail** | **Fail** | **Fatal** | **Fatal** | **Fatal** |
| **Today's results** | **Pass** | **Fail** | **Fatal** | **Pass** | **Fail** | **Fatal** | **Pass** | **Fail** | **Fatal** |
| **Cases to Triangulate** | | √ | √ | | | √ | | | |

Once the list is available, Regression Tracer runs each test against each of the builds of the day. The Regression Tracer utilizes the incremental builds from the Build Tracer. By analyzing the Pass/Fail/Fatal patterns, Regression Tracer can locate the suspect changelist and notify the appropriate developer. The following Table 3 illustrates three (3) typical cases.

**Table 3 - Pass/Fail Triangulation Patterns**

<table>
<tr><th colspan="7"><b>Regression Tracer launches test case for each executable</b><br>(One Executable per Changelist)</th></tr>
<tr><td></td><td>Exec2</td><td>Exec3</td><td>Exec4</td><td>Exec5</td><td>Exec6</td><td>Exec7</td></tr>
<tr><td>Test2</td><td>Pass</td><td><b>Pass</b></td><td><b>Fail</b></td><td>Fail</td><td>Fail</td><td>Fail</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Test3</td><td>Pass</td><td>Pass</td><td><b>Pass</b></td><td><b>Fatal</b></td><td>Fatal</td><td>Fatal</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Test6</td><td>Fail</td><td>Fail</td><td><b>Fail</b></td><td><b>Fatal</b></td><td>Fatal</td><td>Fatal</td></tr>
<tr><td colspan="7"><b>Inference</b>: (One Executable per Changelist)<br>• Test2's failure is found within Executable4 corresponding to Changelist4<br>• Test3's failure is found within Executable5 corresponding to Changelist5<br>• Test6's failure is found within Executable5 corresponding to Changelist5</td></tr>
</table>

In the case of Test2, we observe that the failure appears when running against Exec4. This indicates Changelist #4 as the suspect. Test3 goes from Pass to Fatal with Exec5 (Changelist #5), identified as suspect. "Fatal" is a third state of the regression test outcome as it terminates the application as opposed to "Fail" which might, for example, just return the wrong results for that operation. For Test6, the result worsened from Failed to Fatal with Changelist #5 identified as the suspect.

From an operational standpoint, instrumenting and running these combinations can be onerous. Continuous refinement of the Tracer algorithm helps to manage the operation within acceptable resources. It is key to realize though, that if the Regression Tracer is not in use, the same tasks would need to be performed by each developer and using a less optimal compute environment, therefore introducing more variability and delays in the debug cycle.

## 5.1 Results from Regression Triangulation

In order to assess the effectiveness of the Regression Triangulation, we consider four (4) indicators:

- Volume of regression tests
- Volume of regression tests to be triangulated
- Proportion of suspect Changelist found – Found Rate
- Accuracy in identifying the correct Changelist suspect
- The results for the sampled products are summarized in Table 4.

**Table 4 - Regression Triangulation Effectiveness**

| | 12-month | | |
|---|---|---|---|
| | **Volume of regression tests** | **Volume of regression tests to triangulate** | **Found Rate** |
| | Daily Average | Daily Average | 12-month Average |
| **Product 1** | 8200 | 22 | 42% |
| **Product 2** | 7900 | 26 | 51% |
| **Product 3** | 7000 | 3 | 54% |
| **Product 4** | 4200 | 5 | 67% |
| **Product 5** | 22000 | 35 | 58% |
| **Product 6** | 4200 | 8 | 63% |
| **Product 7** | 5100 | 18 | 70% |
| **Average** | | | 58% |

The first observation is the volume of transactions to be triangulated is manageable, with an average daily load of new test failures at relatively low levels over 12 months of results. There are days where the volume of regression to triangulate is high. If the regression-pass rate drops drastically from one day to the next, for example, the Regression Tracer will analyze only a portion of the results. Our experience shows that generally, this is caused by a single changelist, or by a code merge. At that point, there is no basis to triangulate every single issue individually. Limiting the triangulation to a maximum daily number allows better management of the compute farm resources.

One of the indicators, the accuracy of the findings, is not provided in the table. This is because we found that very rarely does Regression Tracer identify the wrong changelist. There are some anecdotal cases, but typically very few are reported. This is due, in part, to the logic applied to determine the suspect which is “conservative”, meaning that if there is a doubt in identifying a single suspect or because the test environment appears unstable, the case is classified as “Suspect Not Found.” The accuracy of the findings was favored in the implementation to establish high confidence in the results.

That last point, also explains, in part, the lesser than 100% Suspect Found Rate, but only in part. The Found Rate averages 58%. In general, reproducing failures involves many moving parts. To get a more comprehensive understanding as to why some failures cannot be triangulated, we have to consider some of the variables.

- **Nature of the application**: Within the domain of the EDA (Electronic Design Automation) software application, not all tests are deterministic [5]. For example, an EDA application compiles instructions from the programming language for two equivalent but slightly different electronic circuits.
- **Brittleness of the tests**: In many cases, the results must be interpreted or are acceptable within a range of values. For some of these tests, the range is too strict or requires constant revision. The design of the tests must then be revised.
- **Modifications in the test environment**: Changes in the test environment may also prevent Regression Tracer from singling out the suspect. For example, the test passes for all changelists during the Tracer runs. This might be because the test environment during the daily regression run was unstable.

Root cause analysis of cases where the checkin was not identified by our regression triangulation system leaves a path for further improvement initiatives to refine how the tests are designed and how to improve control of the test environment.

# 6 Impact on Productivity

The argument for a developer’s high productivity is based upon two aspects: a short feedback loop and a stable code base. The results of the Build Tracer and Regression Tracer demonstrate significant gain in the goal to shorten the feedback loop.

To assess the gain on stabilizing the code, we ultimately need to look at the Build Success Rate and Regression Passing Rate. Table 5 provides the figures for a 12-month average period.

**Table 5 - Impact on Productivity**

| | **Daily Build Success Rate** | **Daily Regression Passing Rate** |
|---|---|---|
| | (12-month Average) | |
| **Product 1** | 98% | 95.6% |
| **Product 2** | 97% | 98.3% |
| **Product 3** | 95% | 99.6% |
| **Product 4** | 100% | 99.2% |
| **Product 5** | 99% | 98.0% |
| **Product 6** | 96% | 99.4% |
| **Product 7** | 96% | 93.7% |
| **Average** | 97% | 98% |

The results definitely show, for the majority of the products, high predictability of the Daily Build, as well as a code branch stable enough to allow daily code integration from many developers. The Build Success Rate does not yet achieve the goal of 100%. The root cause analysis has identified few improvement opportunities in the Build process that will close the gap to 100%.

The Tracer tool has been in use for more than two years by many teams. Its use has had many positive outcomes:

- Reduced effort required to dispatch failures to all stakeholders
- Reduced effort by each developer in parsing reports and running multiple tests to locate the point of failure
- Shortened the defect resolution cycle
- Improved stability of the code development branch

Many of these points are difficult to quantify but product teams within our company, using Tracer, agree that it is now essential to their operation. For one particular team, Tracer is credited in saving considerable effort by the QA team dedicated to reproducing failures.

One side effect in deploying Regression Tracer is on the assigned responsibility for taking action for a failed test. The accountability now lies with the person who makes the regression test fail as opposed to, in the past, the person who owns the regression test. This is gradually forming a more disciplined culture within the organization.

# 7 Future Development

As we have presented in this paper, the solution is not optimal in all aspects. There are further avenues of development to pursue to increase the value proposition of the Tracer tool. Among those identified are methods and processes needed to do the following:

- Characterize cases where regression triangulation cannot determine a unique changelist for the failure
- Shorten turnaround time to obtain regression triangulation results
- Triangulate other test suites beyond the ones for regression tests

# 8 Conclusion

R&D productivity has been at the forefront of Synopsys development strategy. Automation through tools like Tracer is one successful approach towards optimizing the development resources. The point of improving productivity for developers is to alleviate tasks that are not contributing to the value of our products for our customers.  All these gains result in more time to develop new features and solutions that solves our customer's challenges.

The Triangulation System has proven effective at providing predictable and high quality software releases. The benefits of the system are tangible to the development team on a daily basis and it is now part of the standard tool set in the Synopsys development environment.

# References

## Acknowledgments

*I would like to sincerely thank Wei Gui and Sangeeta Aggrwal for their patient support in providing the data necessary for this paper.  It is much appreciated.*

## Reference

[1] Jones, Capers and Bonsignour Olivier, "The Economics of Software Quality", Addison-Wesley 2012

[2] Wikipedia, "Triangulation definition", http://en.wikipedia.org/wiki/Triangulation

[3] Wikipedia, "Changelist", http://en.wikipedia.org/wiki/Revision_control

[4] Fowler, Martin (1 May 2006). "Continuous Integration". *martinfowler.com*. (Accessed 9 January 2014)

[5] Wikipedia, "Nondeterministic algorithm", http://en.wikipedia.org/wiki/Nondeterministic_algorithm

# Using AOP Techniques as an Alternative Test Strategy

**Lian Yang**

ly8838@gmail.com

## Abstract

Ever since its inception [**1**], researchers and software professionals saw that Aspect-oriented Programming (**AOP**) has huge potential to become a powerful new approach for software development and software testing. There have been many papers in recent years on the subject of Aspect-oriented Test, exploring various interesting ideas and techniques in software test automation [**2**], [**3**]. This paper presents a strategic new approach, using AOP, for software test automation and attempts to incorporate test automation filters (or **interceptors** in **AOP** jargon) in real applications through-out an application's life cycle.

## Biography

*Mr. Lian Yang is an independent software solution and quality consultant based at Redmond Washington. He is the principal solution architect for Yachi Technical Consultant, Inc. His passion lies in cloud computing, SOA, next generation programming languages, software performance issues, application security, and software quality controls.*

*Since 1991, Lian has been working in software industry. As soon as he graduated from Portland State University with an M.S. degree, he became a Software Engineer at ImageBuilder Software in Portland, Oregon, working for award-winning PC games for kids.*

*He later joined Microsoft Corporation in Redmond Washington as a Software Developer in 1995 and worked at several software testing and performance analysis tools. He then became a Lead Software Developer in Test and led QA teams for MSN and Windows Storage Server.*

*After 2008, he worked as an independent consultant and served as lead developer for various consulting companies, building and testing commercial web site for high profile clients such as Microsoft and Avanade. He also worked as solution architect for start-up companies in cloud computing areas.*

*Lian has an M.S. in Computer Engineering from the Portland State University.*

# 1. INTRODUCTION

The software testing landscape has changed dramatically in the past decades amid the development paradigm shift from traditional waterfall to an agile-style test-driven development. The wide acceptance of developer-driven unit test practices has made certain test automation practices obsolete.  On the other hand, the emerging cloud-based and mobile-based applications have changed the core of the traditional software development life cycle, causing software test practice changes.  Emphasis on rapid test turn around, on-line test, and non-interference testing are more and more prevalent. As a result of this development, the traditional software test methodology simply cannot meet the software application's development demand and the dynamic life cycle of average cloud and mobile application.

This article proposes an AOP-based white-box test strategy, which treats software quality assurance as an aspect of software functionality and thus employs certain aspect oriented programming (AOP) principals to software testing.

The goal is to make the always sought after white-box testing less obstructive and more dynamic. Upon achieving this goal, the following effect will be shown:

- Testing and maintainability concerns would be treated as an essential feature, improving software quality in today's agile and fast paced software development arena.
- Making test automation non-obstructive by addressing testing concern at the architectural level and defining these concerns at various *point-cuts*.
- Running and improving test cases and runtime diagnostics can be conducted at runtime in parallel, without the need for re-compiling and taking your application offline, thus making test more relevant to the entire life cycle of application.

## 1.1. ISSURES FACING SOFTWARE TESTING TODAY

With today's trend for test-driven development, some testers feel a little lost as to what their roles have become. Traditional test automation uses a black box test strategy and does a good job covering key functional areas. However, it overlaps, to certain degree, with developer-driven unit test framework. This could result in resource waste and cause under-testing in integration, performance, and more complex test scenarios.

Your test team should adapt to the new challenge and focus more on integration, performance, and live incidence diagnostics, which require more white-box testing methods over black-box testing.

Traditionally, we can conduct white-box tests by adding test hooks and logs. Although they are still widely used by developers as diagnostic and maintenance tools, they are rarely systematic and not widely accepted as best practices.  The side effects they cause to application are:

1) Loss of code brevity
   Mixing test code and application logic produces ugly code and will likely cause long-term code maintenance difficulties.
2) Performance
   Even the test hooks and logs are usually turned off by conditional statements, they nevertheless count on overall code size and their tiny runtime effects could accumulate and result in noticeable performance loss.
3) Security
   Mixing test code and development code together is always a security concern.

It would be nice to separate development concerns and test concerns entirely and apply test concern across functions, classes, and modules, with minimal impact on development processes and code. Aspect-oriented programming (AOP) provides a good alternative.

## 1.2. AOP CAN PROVIDE A NEW TEST STRATEGY

According to Wikipedia, **aspect-oriented programming** (**AOP**) is a programming paradigm that aims to increase modularity by allowing ***the separation of cross-cutting concerns***. AOP forms a basis for aspect-oriented software development.

What are cross-cutting concerns? They are the concerns shared across different types, functions, and modules, thus are not very natural and efficient to be expressed by traditional OOP concepts, such as class inheritance and polymorphism. The latter is best for expressing families of entities, while the former is better expressed by cross-cutting points, joint points, and point-cuts.
The following diagram illustrates the difference:

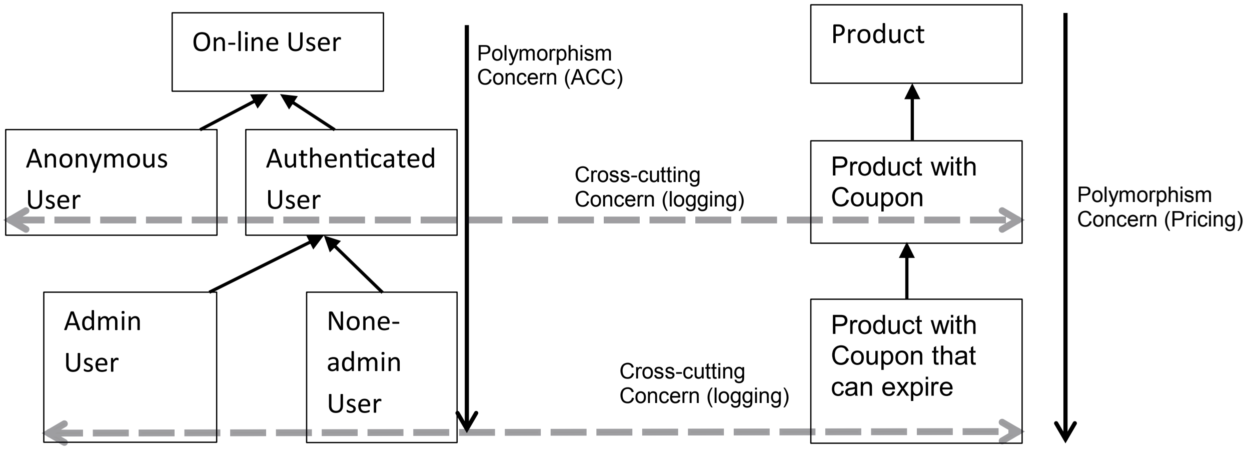

On-line User
Anonymous User
Authenticated User
Admin User
None-admin User
Polymorphism Concern (ACC)
Cross-cutting Concern (logging)
Cross-cutting Concern (logging)
Product
Product with Coupon
Product with Coupon that can expire
Polymorphism Concern (Pricing)

**Fig**. 1: Polymorphic concerns ( ↓ ) vs. Cross cutting concerns ( ⟵ ⟶ )

In the above on-line shopping example, we have polymorphic concerns for both user and product family (solid vertical line). For the user family, the concern is Access Control while for the product family the concern is pricing. The family-specific concerns are better expressed by OOP polymorphic functions defined in each class family roots.

We also have common (cross-cutting) concerns for both class families for logging. For these concerns, OOP polymorphic functions would work, although requiring lots of ad-hoc coding: we could add another layer as the root class for both user and product families and use another polymorphic function to support the cross-cutting concern for logging. However, that would be inefficient and not extensible because we cannot expect that the newly added root would support all the future unrelated cross-cutting concerns. More than anything else, the solution would possibly violate OOP's single responsibility principal (SRP) and open-close principal (OCP).

In essence, the OOP design philosophy dictates that the design changes should only extend vertically within the class hierarchy, not horizontally across different class families.

# 2. AOP FOR SOFTWARE TESTING

Aspect-oriented programming is exactly proposed for solving the above cross-cutting concern design issues. At the first thought, we could use “global functions” for cross-cutting concerns and apply these functions across the types and modules. Conceptually, it is not wrong technically to realize AOP principals. But that would hardly be considered as a new programming “methodology”. For AOP to become useful beyond a talking point, we need some technology and tools to help us expressing cross-cutting concerns without resorting to tedious ad-hoc coding. AOP does have a novel way and it is called an “***interceptor***”.

An AOP interceptor is a design pattern which uses ways to inject code between two components and alter the execution flow. Places where the interceptor can inject itself in are called ***joint points*** and the filters with which the joint points are identified are called ***point-cuts***, figuratively describing cutting a line in between two components where code can be injected.

For the example, if we use AOP interceptor to implement logging for “all” the required class methods at the *before or after function call joint points*, we can save lots of coding efforts and conceptual complexity. The following diagram shows the logging example using interceptors to decorate function foo with a pre-call logging and a post-call logging:

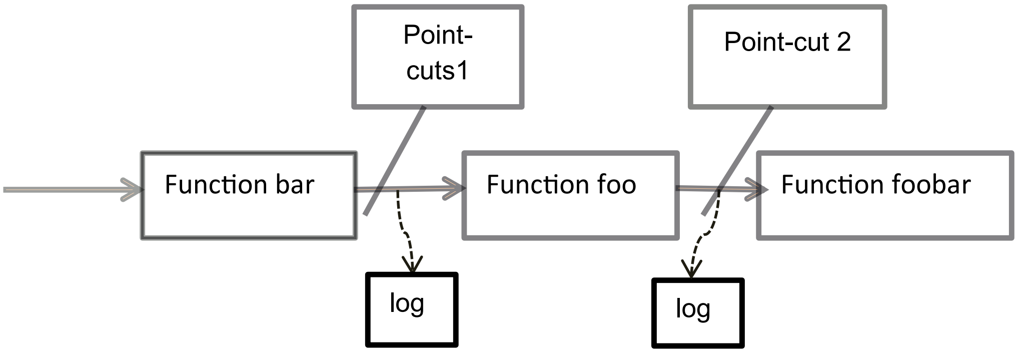


**Fig**. 2: an illustration of an AOP interceptor.

## 2.1. Design Goals

Using AOP in test automation, we are aiming to accomplish the following design goals:

- Rapid development evolution cycle
- More white-box testing for maintainability and runtime diagnostics
- Minimum performance impact
- No service interruption
- No security impact
- No interference with app development
- Not introducing a huge infrastructural change

## 2.2. AOP based White-box testing

To accomplish the above goals, we first take a look at what AOP can help in developing white-box test automation:

Traditional black box testing does not need to know the internal states of the target application. But today, lots of complex bugs can only be understood and fixed by examining an app's internal states, such as resource usage, logged on users, connection status, and other domain-specific states. Those bugs are "hot" bugs in that they can best be diagnosed during live sessions. It would help diagnostics of live bugs by peeking into the internal state.

AOP can help us obtain above states by identifying point-cuts and intercepting function calls at various joint points and gathering the information.

## 2.3. Identifying Cross-cutting Concerns

There is no way to foresee all the joint points that an interceptor is needed for, nor the kind of interceptor is needed. Therefore the first design and architectural decision is to identify these elements:

1) Address Cross-cutting test concerns
- Verification of an internal state at a joint point (for both pre and post release test)
- Logging of the internal state for later analysis
- Injecting error condition at an internal joint point (more applicable for pre-release test)
2) Connect test automation with app logic by interceptors
3) Develop the test automation against interceptors separately along with app development

## 2.4. Code-Interception Techniques

There are built-in supports for AOP interceptors from various programming languages. For languages that do not provide interception support, there are design patterns such as delegates or decorators as "poor-men's AOP" as well.

- Attributes
  Attributes are the most popular ways to "decorate" a function or class. Together with reflection, they are widely used ways to intercept a function call or type construction.
  **Pros**: easy to use and proven to be an effective way to intercept any functions calls or class constructions.
  **Cons**: the attributes have to be fixed at design time and there is no way to add/remove them at runtime.

- Filter infrastructure
  Filter infrastructure provides a systematical way to "filter" a function call before or after a function is called. A good example of the filter infrastructure is the ASP.NET MVC filters, which provides an infrastructure to intercept any "controller actions" at different processing stages. This is a very powerful technique for modifying a page output or monitoring certain server status before the page is rendered. It can be combined with attribute to turn on/off the interception for individual controller actions.

- Source code re-write or injecting new code
  This is the ultimate way to intercept a call. This was impossible to do for traditional statically typed languages such as C/C++. With the popularity if dynamical typed programming languages such JavaScript (NODE JS as its server side version) and C# (still statically typed with support for dynamics and runtime code ejection).
  **Pros**: this technology goes beyond AOP and brings programming to an unseen new level.
  **Cons**: lacks support from statically typed programming languages

- Decorator design pattern
  Statically typed programming languages use this design pattern to achieve AOP interception. Obviously, this technique requires lots of coding and cannot support runtime code injection and requires recompilation for adding/removing decorators.

- Runtime configuration
  Using runtime configuration, we can dynamically turn on/off a piece of code inside the target application, thus achieves the enabling of test behavior without affecting app running. The drawback of this approach is obvious in that we have to add code at the point-cuts.

## 2.5. More on Dynamic code generation

Dynamical programming gives us greater support for injecting dynamical code at runtime. For example, in JavaScript it is pretty easy to “re-write” an existing function and create an entirely new function based on the function:

```
function insertCode(func, replacer, pattern) {
        var newFunc = func.toString().replace(pattern, replacer);
        eval(newFunc);
}
```

JIT compiled code allows more dynamic code generation as well.

We could potentially generate an entirely new test automation application by altering the target app code and inserting test code at different joint points. AOP.JS is such a tool.

This kind of tool serves as a base idea for future system which can dynamically “re-write” the target system into a new system with full-fledged white-box testing automation capability.

Although this approached sounds like a far-fetched idea and not exactly aspect-oriented, it is inspired by AOP and not too hard to achieve under dynamical programming systems such as JavaScript or .NET platform. The potential of this technique sees many cutting edge usages and one of them is a completely innovative test automation framework.

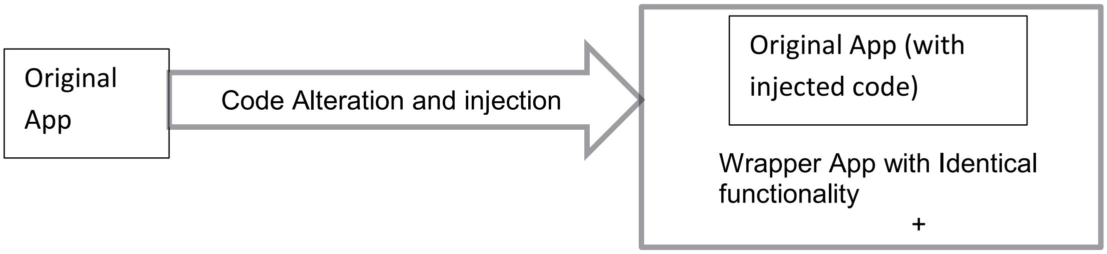

Original App
Code Alteration and injection
Original App (with injected code)
Wrapper App with Identical functionality
+

**Fig**. 3: illustration for a code generation system, which is not only an AOP interceptor but a turn-key solution for dynamically generates a new application based on an existing application – a test automation framework for any target application in a runtime environment that supports dynamic code injection and source code generation.

# 3. CASE STUDY

## 3.1. An ASP.NET MVC 4 Web Application

Microsoft ASP.NET MVC provides great AOP supports via attributes and application filters. This was one of the most important reasons for us to choose it as our web development platform.

Our web site is a political polling application, supporting localization, role-based security, and location-based advertising. We started using an attribute-based filter to test certain aspect of test that would be hard to automate. For example, we need to test location based resources such as style and images, as well as localized content based on user roles and user locations. We found that it is extremely useful to use a filter inside the web server so we can execute a test function when a user with "test role" submitted a request. It has since been developed into a full-fledged test automation infrastructure, which can be turned on/off in live site and conduct essential test in areas of security, localization, and content.

The following diagram shows the idea and infrastructure:

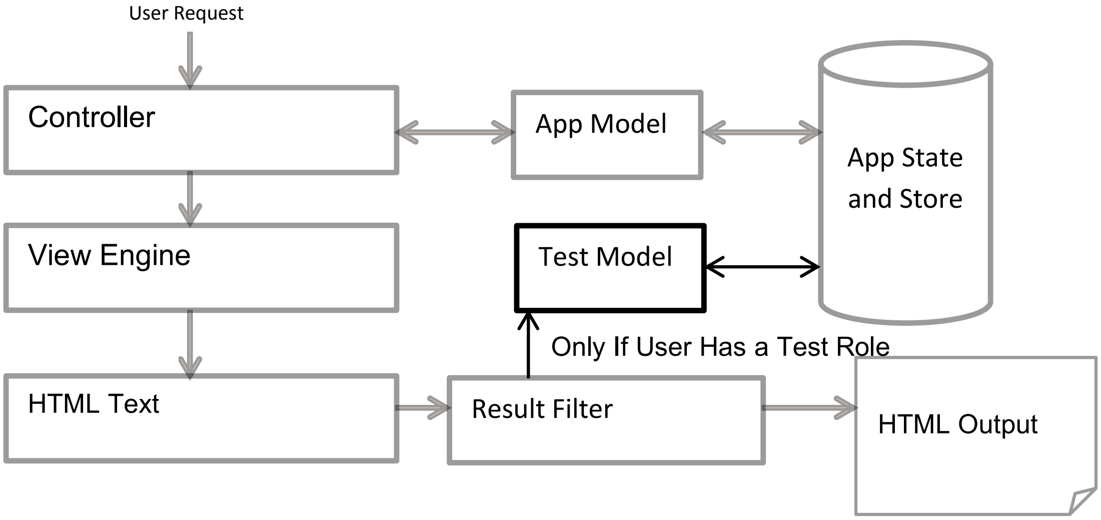


**Fig**. 4: an actual web application using AOP as test infrastructure. Test model can be turned on/off at live site and invoked via a global request filter.

## 3.2. Advantage

1. Test automation does not interfere with app development as long as the "result filter" infrastructure is considered and decided at the design time.
2. Test automation works with live system and support application operation / maintenance via test role (using ASP.NET role based security). This is also secure since the "test role" is approved and provisioned only by admin through managed deployment process and will not be exposed to end users ever.
3. Can do white box test through-out the application life cycle.

## 3.3. Issues

1. Impact of performance
   This should be minimal since the test model will not be called unless the request coming from a test role.
2. Security
   It should be well documented and the deployment is well managed so that no end user will ever be granted test role.
3. Microsoft ASP.Net MVC platform centric
   We are biased since we use ASP.Net MVC, which has first class AOP support. However, its idea and philosophy shall apply to other platform that supports some sort of role based security and server side filters.

## 3.4. Code Snippet

**Create a global filter for test-only**

```
 [AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
 public sealed class TestAuotmationAttribute : ActionFilterAttribute
{
        /* Some state info is maintained here for HTML output capture
           (omitted)
       */

        // called when a request is processed by controller
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
       // Do nothing unless used as test automation by testers
        if (filterContext.HttpContext.User.IsInRole("testAutomation") == false)
        {
           base.OnActionExecuting(filterContext);
           return;
        }
        /*
        Details for capturing output omitted…
        */
    }

    // called prior to HTML rendering. It is an ideal place for
    // capturing HTML output and context for verification and analysis
    public override void OnResultExecuted(ResultExecutedContext filterContext)
    {
        if (filterContext.HttpContext.User.IsInRole("admin") == false)
        {
           base.OnResultExecuted(filterContext);
           return;
        }
        /*
        Details for capturing output omitted…
        */
            // Test Automation method, passing cached app states, HTTP context, and
            // generated HTML output
            DoTest(CachedState, filterContext.RequestContext.HttpContext, htmlOutput);
    }
  }
```

**Register a global result filter at App Start-up**

```
public class FilterConfig
{
  public static void RegisterGlobalFilters(GlobalFilterCollection filters)
 {
    filters.Add(new HandleErrorAttribute());
    filters.Add(new TestAuotmationAttribute ());
 }
}
```

## 3.5. Conclusion

As you can see from the above example, we have a simple and clean solution for white-box test automation infrastructure with the AOP supports from ASP.NET MVC web platform.

We have created a test automation engine that can be used throughout product development cycle. In particular, it can be used after the product has been published with ***negligible*** impact on the live site.

However, I want to remind you that the support for AOP is not without issues.  In the above example, there is still significant expertise needed to design and maintain the solution. It is not entirely dynamic either.

We expect that with more advanced AOP support on the way, we will see more advanced test automation solutions taking advantage of AOP methodologies.

# REFERENCES

[1] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C., Loingtier J.M. and Irwin J., Aspect-oriented Programming. http://cseweb.ucsd.edu/~wgg/CSE218/aop-ecoop97.pdf

[2] Rajan H. AND Sullivan K., Generalizing AOP for Aspect-Oriented Testing

[3] Misra A., Mehra R.: Novel Approach to Automated Test Data Generation for AOP. International Journal of Information and Education Technology, Vol. 1, No. 2, June 2011

# Automated Synthetic Exploratory Monitoring of Dynamic Web Sites Using Selenium

**Marcelo M. De Barros**

marcelod@microsoft.com

## Abstract

Web search engines are very dynamic in nature; not only are the backend and data powering the site evolving, but the frontend is always adapting to different browsers, devices and form-factors, and experiments are often running in production. In fact, when it comes to User Experience (UX), it is likely that users are always falling into some live experiment in production: variation of colors, fonts, typography, different Java Scripts and so on. Issues can occur on the live site for very particular contexts, where a context is defined as a particular configuration of browser, market and experiment. As an example, a JavaScript error can occur on a certain page, for certain types of queries, against a certain market on a particular browser. Creating a priori monitoring for all these different contexts is not feasible.

We at the Microsoft Bing Experiences Team developed a concept of synthetic exploratory monitoring that can focus on the important features on the sites and pages, and use invariants (conditions that should always hold true, or always hold false, for specific contexts) to detect potential anomalies in the current context. We make use of stochastic models to ensure maximum relevant coverage of contexts and devices. We use the power of the Selenium testing framework to drive end-to-end automation on browsers and devices, the notion of exploratory tests based on simple finite-state machines, and a set of heuristics and invariants (text-based and image-based) that can auto-detect problems on the live site in very particular contexts.

We implemented the idea explained in this paper to monitor large-scale web sites such as Bing Search Engine where alerts are generated automatically whenever the anomaly conditions are detected. The solution is easily expandable to other sites. We envision, as future work, moving this technology to the cloud that would allow easy customization of all parameters (browsers used, definition of the finite-state machine, heuristics and invariants).

This paper explains the fundamental principles to create a stochastic monitoring model and demonstrates how to apply the principles to large-scale web sites and services. We will utilize Bing Search Engine to illustrate the techniques explained here.

## Biography

*Marcelo De Barros is a Principal Software Test Manager at Bing, Microsoft. A 15-year veteran at Microsoft, Marcelo has worked on a variety of different services, from Xbox to Search Engines. He has given many talks across the industry, most recently speaking at the Swiss Testing Day 2014 about the evolution of tests in the context of agile systems and organizations [1]. He holds 6 U.S. Patents and 9 U.S. Patent Applications, a M.Sc. degree in Computer Science by the University of Washington and a B.S. degree in Computer Science by UFPE (Federal University of Pernambuco, Brazil). In his spare time Marcelo is an avid programmer, being ranked in the top 1% of all Project Euler (projecteuler.net) members. He also actively writes an algorithms blog (http://anothercasualcoder.blogspot.com/).*

# 1 Scaling Systems to Devices, Browsers and Markets

In today's world whenever a new online system is launched it is usually available across several devices (devices that display web contents), browsers and markets instantaneously and simultaneously. This poses a significant development challenge since:

a) Different browsers, devices and markets have specific requirements and resources that may differ from each other, and there is no enforced global standard across them.
b) Support for CSS (Cascading Style Sheet) [2] and HTML5 compatibility and support [3] vary significantly from browser to browser.
c) The form-factor for the different devices varies significantly. Because of smaller screens, code might need to be optimized to show the user different information or presentation of the information.
d) Markets are also another important dimension given the differences in language grammars as well as geo-cultural differences. Large-scale systems such as Google, Bing and Facebook are always dealing with such challenges.

**Figure 1: different devices on which Bing is enabled**

Many large-scale web sites are now making use of "flights" or "experiments" [13]. An experiment is a way to expose a percentage of the site's users to a different treatment of the site (which can be differences in the UI, middle-tier, backend or even data differences) in order to collect early feedback and then make an informed decision about the upcoming features for the system. For example, a search engine might want to expose 2% of its users to a search results page (or SERP) that shows only eight "blue links" by default instead of ten blue links. The telemetry for that experiment is then collected and analyzed against the "control" (the ten blue links) and data analysts work on distilling the positive and negative aspects of the experiment. Experiments can overlap with each other. At any point in time there might be tens or even hundreds of experiments running in production environments.

# 2 Monitoring Complexities

Since the code is somewhat customized to different user experiences (experiments, browsers, devices and markets), there is a possibility of encountering specific issues on any of these and worst, on combination of these dimensions: a specific problem may only happen on an experiment, on a given browser, on a given device and for a particular market.

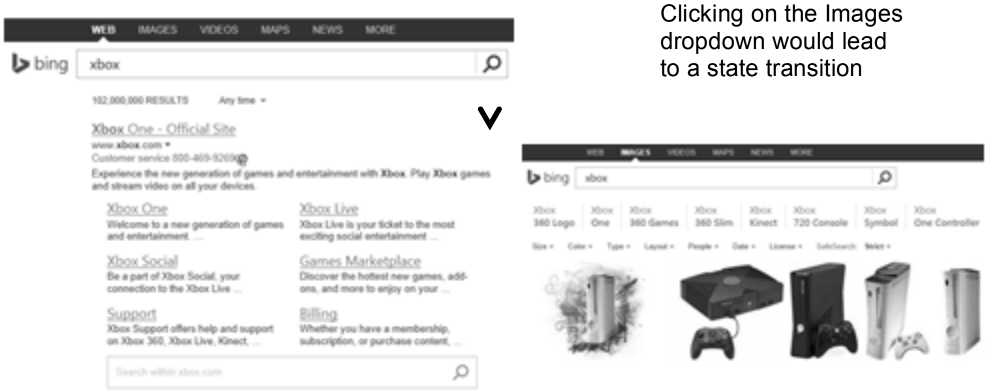
Some simple lower-bound calculations show the complexity and the scale of this problem. If we have around 30 experiments, 30 browsers, 30 devices and 200 markets, the number of possible combinations (assuming no overlaps on the experiments) becomes 30*30*30*200 = 5,400,000 different permutations. Even using well known monitoring techniques such as Gomez [4] or Keynote [5], it becomes impossible to monitor all these variations. In reality, though, most of these contexts are either not significantly crucial to the business or are not valid at all (for example, most of the time experiments are limited to either a group of markets or a group of browsers), hence understanding the valid and important permutations can prune the combinatorial space considerably.

# 3 Using Markov Chains

We use Markov Chains [6] to model the behavior of the system, limiting the monitoring space to the most probable paths. A Markov Chain is a type of stochastic model based on the concept of Finite-State Machines [14] that undergoes transitions from one state to another on a state space. It is a random process usually characterized as memory-less: the next state depends only on the current state and not on the sequence of events that preceded it.

For search engines, the states in a Markov Chain are web site landing pages, such as: the search engine Home Page, the search engine Web Results Page, Videos Results Page, Images Results Page, Settings Page, and any the other page type included in the search engine substrate.

The actions that lead to a state transition are the different actions that can be performed by the end user, mainly Searches, Clicks, Tabs, Hovers, and so on. With enough anonymous log-based information about the different states and actions, one can build a comprehensive Markov Chain diagram modelling the proper behavior of the average user of the web system in questions. The assumption is that most web sites nowadays log information about their users' iterations with the page (in an anonymized manner). The picture below (Figure 2) gives an example of a state transition, and the table below (Table 1) gives an example of a simple Markov Chain. Notice that the key aspect here is that each action is associated with a certain probability (the "Probability Weightings" column in table 1), calculated based on the number (percentage) of users who triggered that respective action based on captured data.



Xbox One - Official Site
www.xbox.com ▾
Customer service 800-469-9269
Experience the new generation of games and entertainment with Xbox. Play Xbox games and stream video on all your devices.

Xbox One
Welcome to a new generation of games and entertainment. ...

Xbox Live
Xbox Live is your ticket to the most exciting social entertainment ...

Xbox Social
Be a part of Xbox Social, your connection to the Xbox Live ...

Games Marketplace
Discover the hottest new games, add-ons, and more to enjoy on your ...

Support
Xbox Support offers help and support on Xbox 360, Xbox Live, Kinect, ...

Billing
Whether you have a membership, subscription, or purchase content, ...

Clicking on the Images dropdown would lead to a state transition

**Figure 2: Example of state transitions**

| Initial State | Final State | Action | Probability Weightings |
|---|---|---|---|
| Home Page | Search Results Page | Typed Query | 20% |
| Home Page | Search Results Page | Clicked on “Top News” | 15% |
| Search Results Page | Search Results Page | Typed Query | 60% |
| Search Results Page | Non-Search Page | Clicked on Ads | 15% |
| Search Results Page | Non-Search Page | Clicked on Algo Result | 33% |

**Table 1: example of Markov Chains states and weighted transitions**

The granularity of the states and the actions is something that varies depending on the applications. In the example above, the Typed Query action could certainly be further refined by specifying the category/class of query being typed, such as “Local Query”, “Adult Query” or “Electronics”. Likewise the “clicked on” event can be grouped into categories (such as “clicked on Algo Results”) or further refined down to the domain of the link being clicked (such as “clicked on an amazon.com link”). The important aspect is to create the chain in such a way that it truly encompasses the users’ behavior but keeping it concise enough to prune the overall search space. For our project, we also added some random aspects to our testing in order to provide extra coverage. For example, when the action is “Send a new query” we take a random query from a pool of pre-defined queries, usually a combination of head and tail queries (See “Web Search Queries” [9]).

Some states are outside the scope of the pages being tested. For example, if the scope being tested are all the pages under the bing.com domain, any site outside that domain would be considered an out-of-scope state. It is important to model the chain in such a way that once out of the scope, actions will lead to in-scope states (such as clicking the back button, or navigating back to the initial state).

With the Markov Chain created, the monitoring approach can be tweaked to randomly follow the paths and probabilities specified by the chain. Notice that the approach will necessarily focus on the most probable paths (assuming a random distribution), which is the desired approach.

In addition to using a Markov Chain for transitions, another important aspect that needs to be taken into consideration is the overall distribution of browsers, devices, markets and flights (experiments).

There are two different approaches to integrating Markov Chains for these additional dimensions into the monitoring system:

1) Create the Markov Chain to take into account browsers, devices, markets and flights (experiments). In such cases there can be multiple Markov Chains for each dimension, or combination of dimensions, or one Chain where states and transitions take into account these dimensions; or, alternatively
2) Create the Markov Chain without the particular data about browsers, devices, markets and flights, and use an orthogonal table with the distribution of the population across these dimensions, and randomly switch to a certain dimension as you navigate the chain.

The approach we have taken is the second one. The Markov Chain is created with the overall usage pattern across all the users in the system. At the same time we get the distribution of users across all browsers, devices, markets and experiments. In the following hypothetical example (Table 2), we see several user context distributions across browsers, devices, markets and experiments. We then combine these two sources of data (the Markov Chain and the User Context Distributions) in order to come up with the proper stochastic model for the exploratory tests. Section 5 explains the details of how these two data sources come together.

*Browsers Distribution*

| **Browser** | **Percentage of users** |
|---|---|
| IE7 | 6% |
| IE8 | 8% |
| IE11 | 15% |
| Firefox | 9% |
| *Others* | *62%* |

*Devices Distribution*

| **Device** | **Percentage of users** |
|---|---|
| Windows Phone | 34% |
| iPhone | 17% |
| Kindle Fire | 17% |
| Android | 9% |
| *Others* | *23%* |

*Markets Distribution*

| **Market** | **Percentage of users** |
|---|---|
| United States | 52% |
| China | 17% |
| Brazil | 4.5% |
| Canada | 7% |
| *Others* | *19.5%* |

*Experiments Distribution*

| **Experiment** | **Percentage of users** |
|---|---|
| Experiment #1: light-blue background color | 2% |
| Experiment #2: larger font size for titles | 3% |
| Experiment #3: larger images | 20% |
| Experiment #4: new relevance ranker | 1% |

**Table 2: example of user context distributions**

# 4 Selenium

Selenium [15] is a portable software testing framework for web applications that provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including Java, C#, Groovy, Perl, PHP, Python and Ruby. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. The way we use Selenium for exploratory tests and monitoring is thru Selenium WebDrivers. Selenium WebDriver accepts commands and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser, and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox or Internet Explorer). Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. There is an ongoing effort by the inventors of Selenium to make it an internet standard [7].

Selenium provides an easy interface to interact with the browser, and the same test scripts can be used against many supported browsers. The ability to perform clicks, hovers, navigation manipulation, simulate different keyboard commands to the browser, scroll, change the browser settings and even detect and manipulate pop-up windows make it ideal for web automation.

In order to provide extra reliability, one can make use of a Selenium Grid. Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provides access to browser instances (WebDriver nodes), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines, and to manage different browser versions and browser configurations centrally (instead of in each individual test).

# 5 Exploratory Runs

The term Exploratory Runs here is loosely used to define the process of semi-randomly exploring different parts of a system while performing different verifications and validations that are pertinent to the current part of the system in question. The semi-random nature is accomplished via two methods: walking the generated Markov Chain, and modifying the context based on the users' distribution of markets, browsers, devices and experiments. The process usually starts at the initial page of the system, such as the user's home page. At that point a frequency-weighted random set of actions gets triggered based on the weight (probability) of the actions in the Markov Chain. It continues from that point on following the same approach indefinitely or until a certain time amount elapses. The transition of the states is imple-mented via commands in Selenium. Figure 3 below illustrates a simple Markov Chain being walked probabilistically:

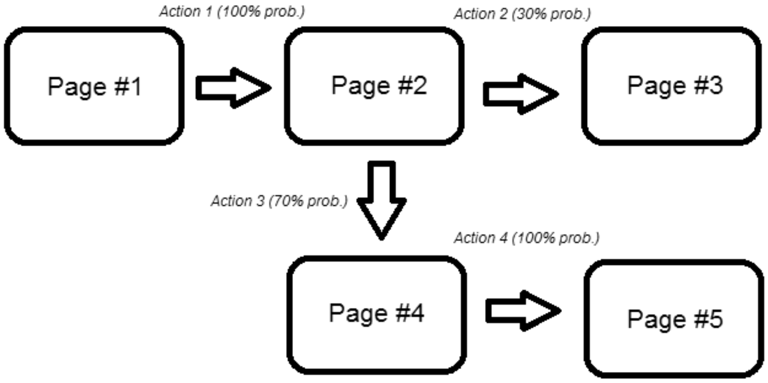


**Figure 3: schema depicting a simple Markov Chain**

Orthogonally to the walk of the Markov Chain, we make use of the contexts distribution in the following manner:

a) Markov Chain traversal keeps happening randomly for a period of time (say N minutes)
b) After that period of time elapses, a change of context happens based on the distribution table

We use N = 30 minutes, which is based on our observations with real Bing user data, 30 minutes is the average time for a user web session. After 30 minutes, the contexts in which the tool is running may change: browser, device, market or experiment. The change is random but weighted based on the distribution tables. We utilize a number of Selenium Grids, one for each type of IE browser (from version 7 to the latest version), and all the grids also contain other browsers, such as Chrome and Firefox. Markets also change based on a set of pre-defined markets (around 200 in our case). The device is simulated on the desktop browsers by manipulating the user-agent [8]. This simulation isn't ideal as some issues only appear or repro on the actual devices, but it is a good stopgap solution to catch some types of issues (like features being under/over triggered). We also force the exploratory run to fall into one (or combination of) experiments by using test hooks (in our case query-string parameters that are only enabled/visible inside the Microsoft corporate network). The automation keeps running indefinitely as a monitoring mechanism against production.

# 6 Subscription-Based Validation Modules

It is common to see the schema of a validation module (or test case) as a self-contained unit that performs all the steps necessary to set up the proper pre-validation before the validation takes place, followed by the validation itself, culminating with the post-validation (or teardown).

Schematically we have:

```
SampleTestCase()
{
  Pre-ValidationSetup();
  Validation();
  Post-ValidationSetup(); //Teardown
}
```

There are many advantages of such scheme: simplicity, standard pattern, readability, reproducibility, determinism, to name a few. However, such a model does not fit well into the exploratory runs mentioned previously. Instead, what we want is a subscription-based model where the test case subscribes to the current state (or action) if the current state (or action) meets certain criteria pertinent to that test.

Schematically, subscription-based test cases have the following format:

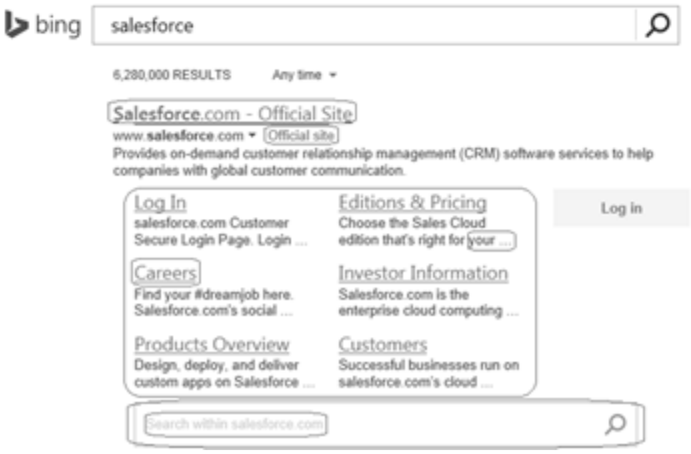
```
SubscriptionBasedSampleTestCase()
{
  If(IsRelevantState(this.CurrentState))
    Validation();
}
```

In essence we're proposing a separation of the validation method from the configuration. The test becomes opportunistic rather than deterministic: if we reach a situation during the traversal of the Markov Chain where the test is applicable, then it runs; otherwise it ignores the current state.

An example of a subscription-based test case would be the following: suppose that we want to write a test case to validate behaviors for a certain segment of queries called navigational queries, which are queries that seek a single website or web page of a single entity (See "Web Search Queries" [9]). A query such as "sales force" is a navigational query. There are several types of validation that can be performed for navigational queries. As per the example in Figure 4, when searching on "sales force", we can base validation on:

a) Correctness of the algorithmic first result returned
b) Proper attribution for "Official Site"
c) Proper number, format, truncation for deep-links [10]
d) Proper placement and usage of inner-search boxes

The picture below (Figure 4) depicts the items that can be subjected to validation:



Salesforce.com - Official Site
www.salesforce.com ▾ Official site
Provides on-demand customer relationship management (CRM) software services to help companies with global customer communication.

Log In
salesforce.com Customer Secure Login Page. Login ...

Editions & Pricing
Choose the Sales Cloud edition that's right for your ...

Careers
Find your #dreamjob here. Salesforce.com's social ...

Investor Information
Salesforce.com is the enterprise cloud computing ...

Products Overview
Design, deploy, and deliver custom apps on Salesforce ...

Customers
Successful businesses run on salesforce.com's cloud ...

**Figure 4: Validation aspects for web-search deep-links**

There are two types of tests that can be used in this model:

1) Custom Tests are specific for only certain states (or actions). For instance, the deep-links validation shown above is an example of custom test since it only applies to pages originated from navigational queries
2) Invariant Tests verify general invariants that should always be true (or always be false) no matter what state we are

Invariant Tests are very powerful since they apply to all states (or actions). It is important and recommended that the product being tested be properly instrumented with test hooks in order to enable invariant conditions that can then be tested thru invariant tests. An example of an invariant test would be a test that looks for java script errors. No state (or action) should lead to a java script error on the page. We instrumented some of the Bing pages so that whenever inside the Microsoft corporate network and when a certain query string parameter is passed in the URL, any java script error is caught via a global try/catch and written into a hidden HTML div tag [11]. With such instrumentation implemented, the invariant test for java script errors becomes trivial – basically checking for the presence of the java script error div tag. Other types of invariant tests are:

a) Links: no links should lead to 404 pages
b) Server Error: no state/action should lead to server errors
c) Security: no state/action should expose any security flaw
d) Overlapping: no state/action should contain overlapped elements

Selenium also provides a capability of taking the screenshot of the current page. This allows the engineers to implement image-based test methods, some of which can be custom methods (such as the rendering and placement of some objects on the page specific to certain contexts) or invariants (such as the space between blocks on the page). Also, it is important to notice that some of the methods only apply to certain contexts (browsers, devices, markets or experiments). In such cases the test needs to verify that the current context is relevant for the test in question to be executed.

# Methodology

Combining all the approaches described in this paper, we come up with the following methodology for synthetic exploratory testing or monitoring of large-scale web systems:

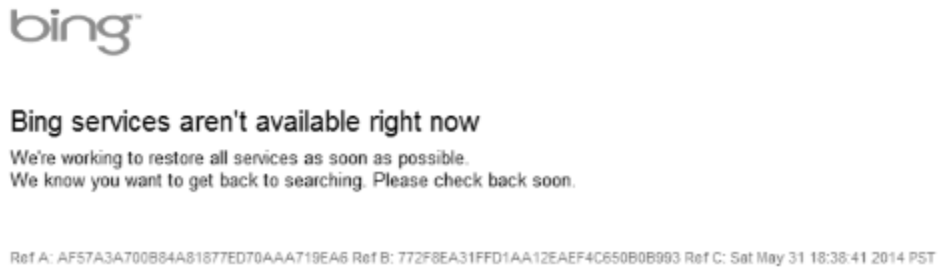
1) Mine the logs to create the user's profile Markov Chain
2) Retrieve the percentage distribution of different contexts (browsers, devices, markets and experiments)
3) Create custom and invariant tests that adhere to the subscription-based model

4) Stochastically run through the Markov Chain using Selenium or Selenium Grid. When testing search engines a key aspect is the generation of relevant queries to be used. It can be a combination of top queries based on frequency as well as segment-specific queries (such as queries that trigger local results or movie results)
5) Sporadically (time-based) switch contexts based on the distribution from #2
6) At each state (and action), apply the subscription-based tests from the library (#3). Alert in case of failures.

We differentiate monitoring from testing in terms of running the tests post-production and pre-production, respectively. The approach can be used for either one. However, we prefer to have deterministic tests as a pre-production mechanism, leaving the non-deterministic ones (such as the stochastic ones based on Markov Chains) as a monitoring mechanism (post-production). Also, the different tests have different priorities, so not all the tests will lead to an escalation (usually the invariant ones are deemed higher priority than the custom ones).

As the approach above executes, over time the critical monitoring paths will certainly be covered. Given that the approach follows a weighted-probability model, the critical paths will be covered more often than the non-critical ones. That is desirable since in today’s fast-pace development environment of large-scale web systems, only the critical problems (the ones affecting the vast majority of users) get real attention, others are treated as low priority. The stochastic model is an elegant way to ensure highly-probable coverage of critical scenarios, and yet also cover some low-key scenarios.

Below are two examples of invariant failures when the model was applied to Bing.com. We used a set of 5 high-end servers executing around 1,000,000 state transitions per day, and running over 100 validation methods (of which 15% were invariant ones). The first example (figure 5) is an invariant that looks for HTTP 500 server errors, in this case generated by a combination of experiment and different interactions with the site:



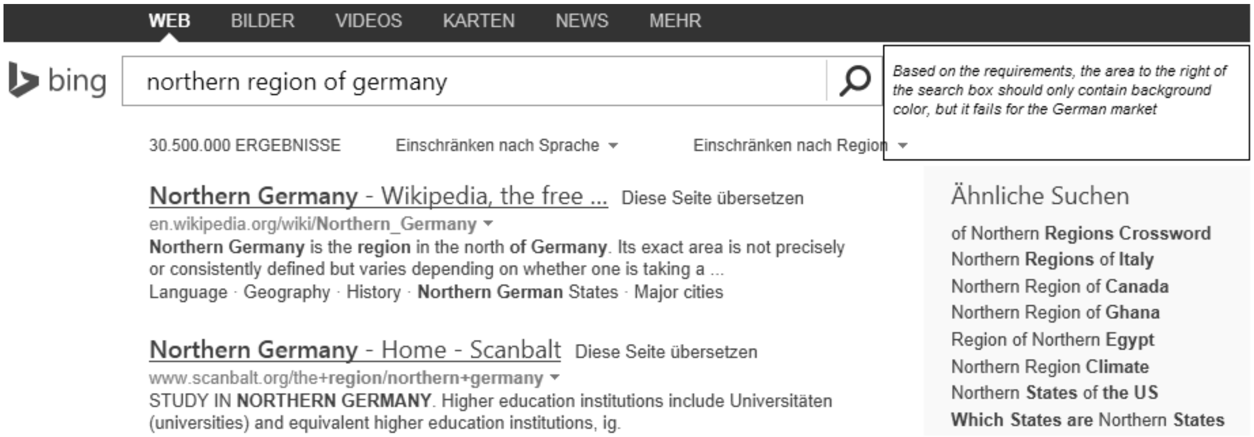
Bing services aren't available right now

We're working to restore all services as soon as possible.
We know you want to get back to searching. Please check back soon.

Ref A: AF57A3A700B84A81877ED70AAA719EA6 Ref B: 772F8EA31FFD1AA12EAEF4C650B0B993 Ref C: Sat May 31 18:38:41 2014 PST

**Figure 5: Issue discovered thru an invariant test method**

The second one is a low-priority invariant test based on image processing. In this case the area to the right of the end of the search box should always contain background color only. But in the case of the German market, whenever search filters are present due to the long words in German, the placement of the filters are going beyond the limits of the search box, breaking the pre-specified requirement. Figure 6 shows an example of such an issue:



northern region of germany

Based on the requirements, the area to the right of the search box should only contain background color, but it fails for the German market

30.500.000 ERGEBNISSE Einschränken nach Sprache ▾ Einschränken nach Region ▾

**Northern Germany** - Wikipedia, the free ... Diese Seite übersetzen
en.wikipedia.org/wiki/**Northern**_**Germany** ▾
**Northern Germany** is the **region** in the north **of Germany**. Its exact area is not precisely or consistently defined but varies depending on whether one is taking a ...
Language · Geography · History · **Northern German** States · Major cities

**Northern Germany** - Home - Scanbalt Diese Seite übersetzen
www.scanbalt.org/the+**region**/**northern**+**germany** ▾
STUDY IN **NORTHERN GERMANY**. Higher education institutions include Universitäten (universities) and equivalent higher education institutions, ig.

Ähnliche Suchen

of Northern **Regions Crossword**
Northern **Regions** of **Italy**
Northern Region of **Canada**
Northern Region of **Ghana**
Region of Northern **Egypt**
Northern Region **Climate**
Northern **States of the US**
**Which States are** Northern **States**

**Figure 6: Example of an image-based error related to markets**

Notice that the use of Markov Chains and context distributions allows the monitoring system to be highly adaptive: as the user patterns and context distributions change over time, the system will adapt itself based on the new data. The other important aspect is that the validation and monitoring mechanisms can certainly be extended to more than functional use, such as covering security concerns. At each step during the traversal of the Chain, we can also plug-in penetration tests [12] which would be characterized as invariant methods.

## Summary

Monitoring large-scale dynamic web sites across multiple browsers, devices, markets and experiments is a very complex task. In this paper we have proposed a way to model the users' behavior via stochastic methods such as Markov Chains and use Selenium to recreate the same conditions experienced by real users in production. In addition, the validation approach is also changed from self-contained validation methods to a subscription-based model where the validation method subscribes to only the applicable states. Finally, validations can be invariant ones (applicable to all states) or custom ones (applicable to specific states).

## References

[1] M. De Barros, Chap Alex, 2014. *Agile quality-centric development process of large-scale web systems*, Swiss Testing Day

[2] Comparison of layout engines - CSS, http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(CSS) (accessed July 16, 2014)

[3] Comparison of layout engines – HTML5, http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(HTML5) (accessed July 16, 2014)

[4] Gomez Network, https://www.gomeznetworks.com/?g=1 (accessed July 16, 2014)

[5] Keynote, http://www.keynote.com/ (accessed July 16, 2014)

[6] M. De Barros et al, 2007. *Web Services Wind Tunnel: On Performance Testing Large-Scale Stateful Web Services*, 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)

[7] Selenium Software, http://en.wikipedia.org/wiki/Selenium_(software) (accessed July 16, 2014)

[8] Browser User Agents, http://en.wikipedia.org/wiki/User_agent (accessed July 16, 2014)

[9] Web Search Queries, http://en.wikipedia.org/wiki/Web_search_query (accessed July 16, 2014)

[10] Deep linking, http://en.wikipedia.org/wiki/Deep_linking (accessed July 16, 2014)

[11] The HTML <div> tag, http://www.w3schools.com/tags/tag_div.asp (accessed July 16, 2014)

[12] Penetration Tests, http://en.wikipedia.org/wiki/Penetration_test (accessed July 16, 2014)

[13] A/B Testing, http://en.wikipedia.org/wiki/A/B_testing (accessed July 16, 2014)

[14] Finite-State Machine, http://en.wikipedia.org/wiki/Finite-state_machine (accessed July 16, 2014)

[15] Selenium HQ Browser Automation, http://docs.seleniumhq.org/ (accessed July 16, 2014)

# Transition from a Rapid Prototyping to Programmatic Test Framework

**Robert A Zakes and**
**Brenden Beaman**
robzak@state.or.us and Brenden.beaman@state.or.us

## Abstract

The office of the Oregon Secretary of State was getting an increasing number of help desk calls related to running our applications in different browsers and mobile platforms. Automated testing had been implemented in 2007 using Selenium IDE (a rapid-prototyping test framework) with great success as reported in a 2009 PNSQC paper.

Unfortunately, the IDE version of Selenium is a Firefox plug in and consequently only tested our Web applications using Firefox.

We needed a way to run our automated test scripts in multiple browsers and on the most popular mobile platforms. Selenium WebDriver, the follow on product to Selenium RC has this capability, but requires more technical programming skills to build and maintain scripts.

This paper describes our transition from a rapid-prototyping test framework, Selenium IDE, to a programmatic test framework, Selenium WebDriver. Specifically, it will compare the strengths and weaknesses of both tools in terms of:

- Capabilities and adaptability
- Time investment, metrics for script development
- Script maintenance
- Technical abilities required to become proficient
- Scalability
- Load testing

The paper will discuss our evolved strategy for conversion of a large inventory of IDE regression test scripts and when and where we have decided not to convert.

## Biography

*Bob Zakes is a Requirements and Testing Manager for the Oregon Secretary of State.*
*Bob has over forty years of experience in project management and software requirements and testing. Bob has worked for IBM as an instructor and marketing and systems engineering manager, National Retail Systems West as a product manager, Hanna International as Engineering and Quality Assurance Manager and the State of Oregon as a project manager and his current position. He has been responsible for several successful system implementations at the State. His current primary project is Oregon's Central Business Registry.*

*Brenden Beaman is a Testing Analyst for the Oregon Secretary of State.*
*Brenden has ten years of experience in software development and testing. Brenden has worked at Hewlett Packard as a quality assurance lead, and the State of Oregon as a testing analyst as his current position. His current primary project is Oregon's Central Business Registry.*

# 1 Introduction

## 1.1 Our Agency Mission

The Office of the Oregon Secretary of State runs several of public-facing software applications. Most of these are complex, high exposure web apps, with front-end interfaces as well as searchable databases. To put this in context, the applications are integral to the integrity of Oregon's politics, business, financial condition and history. The applications include Oregon's Central Voter registry and election system, ORESTAR for filing all state candidates for elections, recording all political action committees, campaign contributions and expenditures and the Voter Pamphlet. The Central Business Registry serves as the one stop site for registering businesses and getting registered with state regulatory and licensing agencies. Other applications record notices of security interests in moveable and personal property, statutory liens and warrants. Also included are the registration of the States Notaries Public and indexes to the Oregon's Archive. The Office is also the auditor of all Public accounts. More information about the Office, and the current Secretary of State, Kate Brown, can be found at sos.oregon.gov.

## 1.2 Our Testing Automation Progress

To meet the challenge of testing these applications we started automated testing seven years ago. This was primarily driven by the need to maintain quality testing in an environment of accelerating application growth and limited testing resources. Selenium IDE was selected as the test automation tool due to its ease of use, wide acceptance, and open source being all we could afford. The initial success was reported in a 2009 PNSQC paper, Some Observations on the Transition to Automated Testing. We started automated testing concurrent with our shift to Java web development. A very agile methodology evolved where scripts have closely followed system development. We have found this to be a significant benefit of automated testing, and we believe agile is dependent on automated testing. Since we started, we have built about one hundred and fifty test scripts for most agency applications.

We build three types of scripts, quick (aka smoke, happy path or steel thread), full regression and special scripts for load testing (light), interface testing, performance tuning using timed loop scripts, timeout testing, etc.

| Web Application | Quick Scripts | Regression Scripts | Special Scripts | Total | Lines of Code |
|---|---|---|---|---|---|
| **Central Business Registry** | 7 | 5 | 32 | 44 | 31,526 |
| **Uniform Commercial Code Lien Filing** | 1 | 7 | | 8 | 7392 |
| **Notary Registration** | 1 | 1 | 2 | 4 | 2860 |
| **Elections Reporting and Voter Registration** | 15 | 17 | 30 | 62 | 20867 |
| **Municipal Audit Report Billing and Filing** | 1 | 1 | 3 | 5 | 3850 |
| **Identity Management (user authentication)** | 7 | | 12 | 19 | 3212 |
| **Historic Records** | 1 | 1 | 3 | 5 | 1408 |
| **Cashiering and Revenue Accounting** | 1 | 4 | | 5 | 4,246 |
| **Totals** | 34 | 36 | 82 | 152 | 76,021 |

As soon as a build is deployed, we run our scripts, log and report results. The scripts are run on a bank of five PC's each with Selenium IDE. When an error occurs, the script stops and the failing instruction is highlighted in red. It can be rerun and analyzed at that point.

## 1.3 Anatomy of our Quick and Regression Scripts

The quick scripts are run first after each build and frequently find issues prior to running our full regression such as a missing library or an interface that was not updated. Our model for these scripts is

- Create a script for each major transaction so all major navigation paths are covered,
- Each script is cradle to grave, and will test all interfaces if possible. Here's a sample scenario.
  - Start with customer sign in
  - Customer completes application, makes payment and submits
  - Internal reviewer rejects

  - Customer reopens, fixes and resubmits
  - Internal reviewer approves
  - Customer searches and verifies completed status
- Provide switches to change test environments and stop at key points in the script for debugging,
- Complete all fields with realistic data, e.g., first, middle, last name and suffix,
- Use a date and time stamp or token to append to the primary name to make completed transaction unique
- Choose the predominant path through dynamic pages
- Provide logging for run date and time, duration, environment, reference to entities saved,
- Typically run for one to two minutes.

The regression scripts are spawned from the quick scripts and are also run after each build. Our model for these is

- Include all of the features of the quick scripts
- Validate content on all pages
- Test all default values
- Test all paths for the function or transaction
- Test all paths for dynamic pages
- Test all validations, mandatory fields, options, min/max sizes, ranges, allowable characters and character sets, valid dates, emails, phone numbers, addresses, etc
- Test interface exception handling, e.g., payment failure types
- Typically run for twenty minutes to an hour.

## 1.4 A Change in Culture

As automated testing became part of our development process we noticed a culture change. Our user department testers depend on these scripts and they can now focus only on new functionality and ad hoc testing. Becoming a tester in our user departments is recognized as privilege and an activity leading to advancement. It used to be drudgery. Our developers depend on our script building as part of our agile development to catch inadvertent errors early and to reduce unit testing. Management depends on quality being developed throughout the development and maintenance cycle and the low number of production issues reported.

## 1.5 A Perfect Storm

We started to recognize deficiencies in our test automation methodology. It started at our help desks manned for each of our applications. We were getting an increasing number of calls regarding browser/platform specific issues. A page would not open in a current version of IE. A button was missing in Safari. Table headings were wrapping on an iPad. Selenium IDE is a Firefox add-on, and only works in that browser. While our functional test coverage was excellent, we did not have a good way to look for bugs and compare differences between different browsers and platforms. We had depended on our users testers to test with IE to address the Firefox limitation. When we started we only had to worry about IE and Firefox, but this was no longer the case. Google Analytics showed our users were accessing our systems with a wide distribution of browsers and platforms. The Selenium IDE limitation to Firefox became a major issue.

We also had a few situations where our applications went down due to high traffic. Selenium IDE is not a load let alone a stress test tool. At best, we could get ten PC's running a loop script but this would only generate a fraction of the load required. We tried JMeter load testing tool and found it was useful for testing our application server, but did not stress the database or external interface connections. We could not correlate the test results to our application behavior at heavy loads. We needed a load/stress tool that would replicate our transaction loads and give us the ability to stress to determine our peak stress level.

About a year ago a decision was made to upgrade the Central Business Registry system to improve usability, streamline design, and modularize the sections designed for specific agencies. That decision would obsolete the CBR test scripts, the largest and most complex in our library. That forced the question, should we use Selenium IDE for the new application?

### 1.6 What tool to use?

There were a lot of alternative test frameworks we could consider, and we did look at several, but the choice of Selenium WebDriver was fairly simple. Selenium WebDriver is the companion product designed to provide what is missing in IDE. There are tools to partially convert IDE scripts to WebDriver. WebDriver is open source and no procurement was necessary (a big factor in the public sector). It is widely accepted and has a very active user group. We have been pleased with the stability and reliability of Selenium IDE and it appears the Selenium Project support will continue.

# 2 Selenium IDE versus Selenium WebDriver

Once we had focused down on the two products, we started drilling down into the comparative details of them. We wanted to assess which existing IDE scripts would benefit from being re-developed into WebDriver scripts. In addition, we wanted to determine which product would be used for new script development going forward, and when. To help make these determinations, we started to compare and contrast the various advantages of the two products, as detailed in this paper.

## 2.1 Browser support

Selenium IDE, a Firefox plug-in, is obviously limited to testing with Firefox. Selenium WebDriver offers support for Firefox, Internet Explorer, Chrome, and Safari with very little added effort. With a moderate amount of additional setup, it can also execute your test scripts against an iPad or Android tablet. Being able to easily validate web applications play nicely with the myriad of browser idiosyncrasies was a huge selling point in this regard. One of the first scripts developed was a simple page viewer that we used to test our agency's new website. The script opens pages from a list of URL's for a variable length of time in any browser selected. Content managers were able to test the site on the browsers and tablets mentioned above. Since the script opens every page without any navigation, the tester can focus on the layout. The script was a big help in rolling out a near perfect new website.

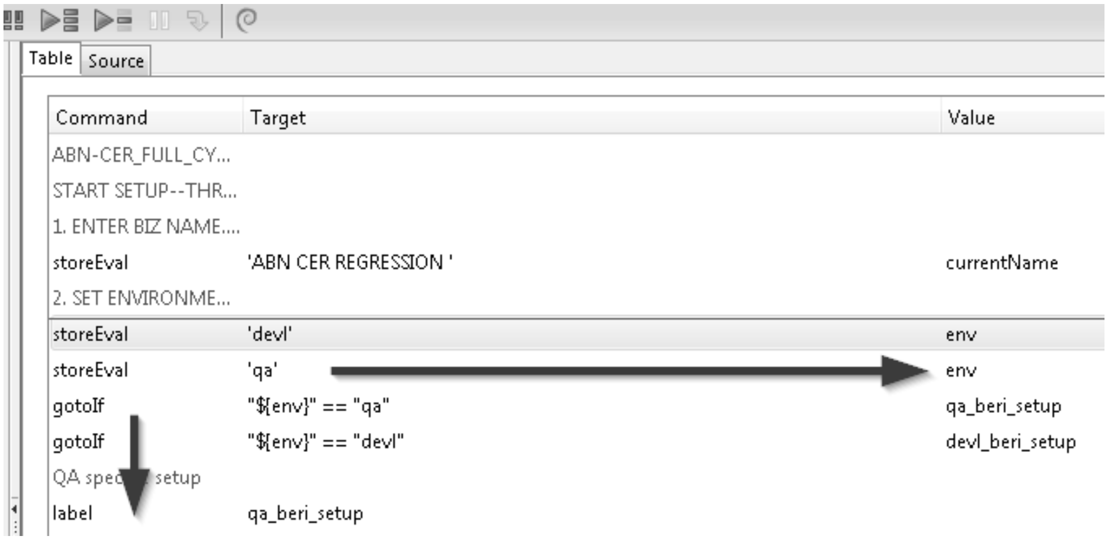
## 2.2 Script portability

The WebDriver API built into a java codebase, can be compiled into an executable, standalone JAR file. These can also package up any required resources such as images, into one file that can be shared or linked to. They're platform independent, so a Mac or Linux user can run them. No install files are required; you can even run them over a network without copying the file locally. This process makes the scripts highly portable. The page viewer mentioned above was packaged in this way.

One of our agency goals has been to encourage content experts to be able to utilize the automated test scripts themselves, in addition to the testing team. These staff members are highly familiar with the business rules and details of their respective departments, but not necessarily technically proficient with testing software. They may or may not have the Firefox browser and/or plugin installed, but if given access to a standalone executable script, they are more prone to use it. The high portability and ease of use of these jar files was a large factor in helping us determine which IDE scripts needed to be converted into WebDriver scripts.

## 2.3 Switch setting versus an options dialogue

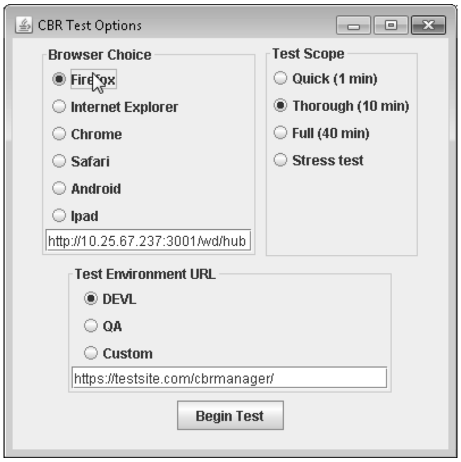
Setting up variables to manage test behavior in the IDE was somewhat primitive. We initialized global variables that held values based on how we wanted the test script to run. For example if we wanted the script to test the DEV environment rather than QA, we would save a string value to a variable which is logically checked throughout the script. Before running the script, you would move the command that sets the wanted environment last. In the example below the value is set to "qa". A gotoIf command executes other code based on the value of that string, as shown below.



| Command | Target | Value |
|---|---|---|
| ABN-CER_FULL_CY... | | |
| START SETUP--THR... | | |
| 1. ENTER BIZ NAME.... | | |
| storeEval | 'ABN CER REGRESSION ' | currentName |
| 2. SET ENVIRONME... | | |
| storeEval | 'devl' | env |
| storeEval | 'qa' | env |
| gotoIf | "${env}" == "qa" | qa_beri_setup |
| gotoIf | "${env}" == "devl" | devl_beri_setup |
| QA spec[illegible] setup | | |
| label | qa_beri_setup | |

Figure 1: *Selenium IDE script showing 'switch setting'*

The Selenium WebDriver API is built on top of an existing program language, as additional API. We used Java in our implementation. Because the existing Java API's are also included, we were able to develop user interfaces using the Java Swing libraries.



CBR Test Options

Browser Choice
- ● Firefox
- ○ Internet Explorer
- ○ Chrome
- ○ Safari
- ○ Android
- ○ Ipad

http://10.25.67.237:3001/wd/hub

Test Scope
- ○ Quick (1 min)
- ● Thorough (10 min)
- ○ Full (40 min)
- ○ Stress test

Test Environment URL
- ● DEVL
- ○ QA
- ○ Custom

https://testsite.com/cbrmanager/

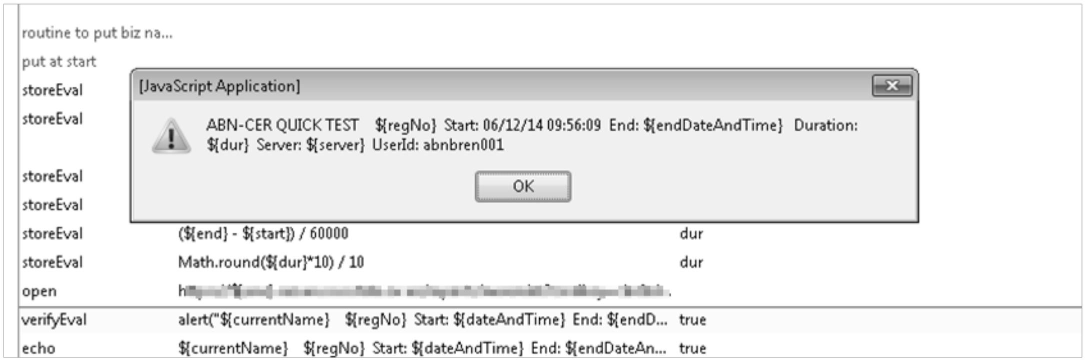
Figure 2: *Options dialog used to set testing parameters of WebDriver script*

This allowed us to easily change the testing parameters before executing the test. Not only did this make the test scripts more powerful, but also very user friendly. Scripting is also considerably easier since you are not limited to gotoIf. Again this capability allows us to build test scripts that can be easily run by non-technical staff, even project managers.

## 2.4 Data analysis functionality

Using an automated script to test an application is one thing, but what do you do with the results you gather? After your script has executed, how do you know if it was successful or not, and if there were any bugs detected? Gathering and displaying test results is a critical aspect of automated testing, and testing in general.

Data gathering and results tracking is possible in the IDE. Throughout the script, you can store variables with various pieces of data, and then display them at the end with an alert window showing various metrics and statistics



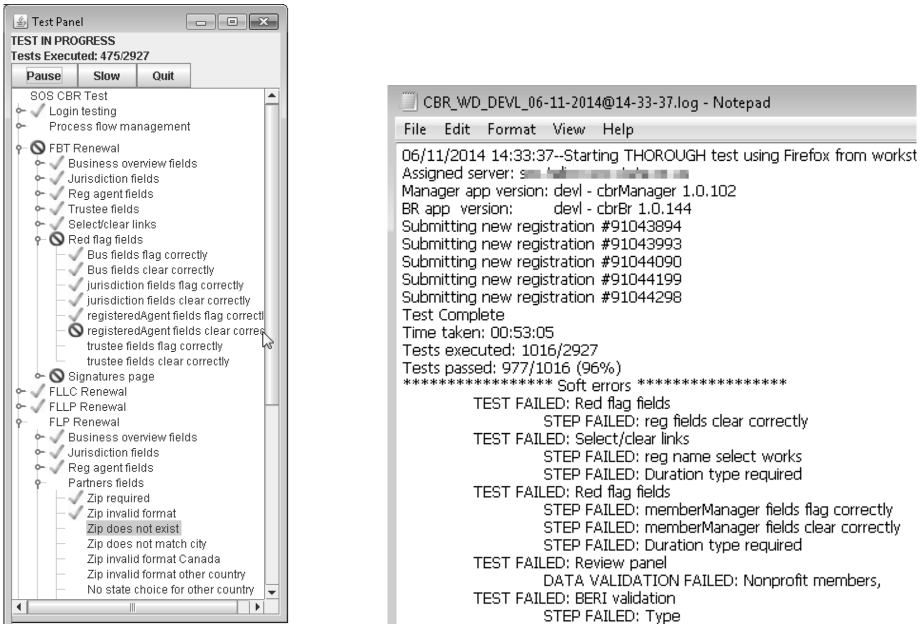
```
routine to put biz na...
put at start
storeEval
storeEval

storeEval
storeEval
storeEval     (${end} - ${start}) / 60000                                   dur
storeEval     Math.round(${dur}*10) / 10                                    dur
open          h[redacted].
verifyEval    alert("${currentName}   ${regNo}  Start: ${dateAndTime}  End: ${endD...   true
echo          ${currentName}   ${regNo}  Start: ${dateAndTime}  End: ${endDateAn...   true
```

[JavaScript Application]

ABN-CER QUICK TEST    ${regNo}  Start: 06/12/14 09:56:09  End: ${endDateAndTime}   Duration: ${dur}  Server: ${server}  UserId: abnbren001

Figure 3: *Selenium IDE script showing a test result dialog*

Although this is somewhat open-ended and up to the developer, the IDE has no native ability to write to files.  The WebDriver API, when extended into java code, allows much more control over data storage, tracking, and reporting.  Using Java Swing elements, we built a test harness into our scripts that would track and update test results in real time.  We could also write those results and other metrics to a log file for later analysis.  This allowed much more control and organization of defect tracking and development progress.



TEST IN PROGRESS
Tests Executed: 475/2927

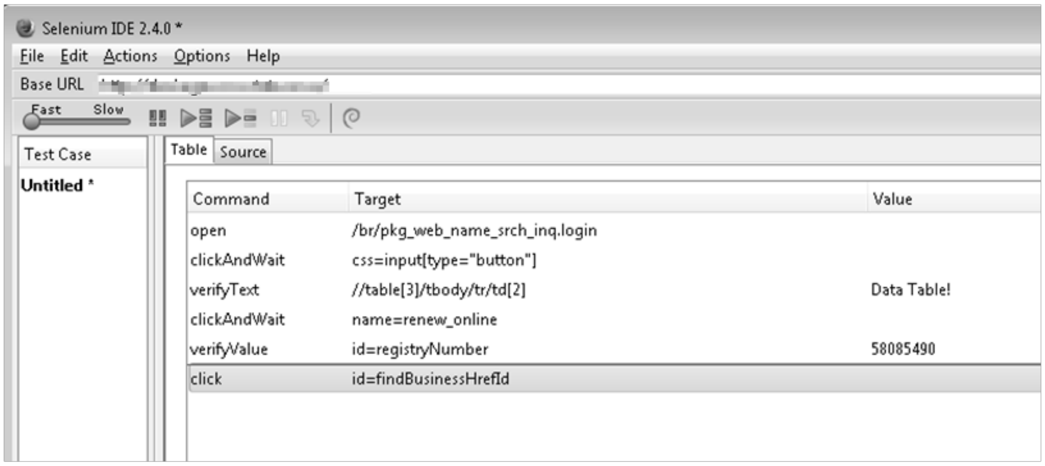
```
SOS CBR Test
  ✓ Login testing
    Process flow management
  ⃠ FBT Renewal
    ✓ Business overview fields
    ✓ Jurisdiction fields
    ✓ Reg agent fields
    ✓ Trustee fields
    ✓ Select/clear links
    ⃠ Red flag fields
      ✓ Bus fields flag correctly
      ✓ Bus fields clear correctly
      ✓ jurisdiction fields flag correctly
      ✓ jurisdiction fields clear correctly
      ✓ registeredAgent fields flag correctl
      ⃠ registeredAgent fields clear correc
        trustee fields flag correctly
        trustee fields clear correctly
    ⃠ Signatures page
  ✓ FLLC Renewal
  ✓ FLLP Renewal
    FLP Renewal
    ✓ Business overview fields
    ✓ Jurisdiction fields
    ✓ Reg agent fields
      Partners fields
      ✓ Zip required
      ✓ Zip invalid format
        Zip does not exist
        Zip does not match city
        Zip invalid format Canada
        Zip invalid format other country
        No state choice for other country
```

```
06/11/2014 14:33:37--Starting THOROUGH test using Firefox from workst
Assigned server: [redacted]
Manager app version: devl - cbrManager 1.0.102
BR app  version:       devl - cbrBr 1.0.144
Submitting new registration #91043894
Submitting new registration #91043993
Submitting new registration #91044090
Submitting new registration #91044199
Submitting new registration #91044298
Test Complete
Time taken: 00:53:05
Tests executed: 1016/2927
Tests passed: 977/1016 (96%)
***************** Soft errors *****************
          TEST FAILED: Red flag fields
                    STEP FAILED: reg fields clear correctly
          TEST FAILED: Select/clear links
                    STEP FAILED: reg name select works
                    STEP FAILED: Duration type required
          TEST FAILED: Red flag fields
                    STEP FAILED: memberManager fields flag correctly
                    STEP FAILED: memberManager fields clear correctly
                    STEP FAILED: Duration type required
          TEST FAILED: Review panel
                    DATA VALIDATION FAILED: Nonprofit members,
          TEST FAILED: BERI validation
                    STEP FAILED: Type
```

Figure 4: *WebDriver custom test result panel and log file*

## 2.5 Script developer skill set required

One advantage of Selenium IDE is the ease of use and speed at which someone can develop a test script.  The IDE itself is fairly straight forward and easy to use, and there is a vast knowledge base of documentation available to assist script developers.  By far the biggest advantage is the record function.  When enabled, you browse the target application as you would normally, and the IDE will automatically generate script code based on what you are clicking on, interacting with, etc.   Users can also right click and select test commands from a menu.  The recorded script not only navigates the application, but can run various tests as it goes as well.  The script can be played back at any time and at any place to watch how it interacts with the application.



Base URL [redacted]

| Command | Target | Value |
|---|---|---|
| open | /br/pkg_web_name_srch_inq.login | |
| clickAndWait | css=input[type="button"] | |
| verifyText | //table[3]/tbody/tr/td[2] | Data Table! |
| clickAndWait | name=renew_online | |
| verifyValue | id=registryNumber | 58085490 |
| click | id=findBusinessHrefId | |

Figure 5: *The Selenium IDE recording a script*

There are some downsides of recording; Scripts generated in this way tend to be fairly ‘raw’ and need additional refinement; they may not be very readable by testers without comments/documentation. Verifying results of your actions aren’t built by the recording process, and need to be added manually. (Verifying that a checkout screen includes all the items you added to a shopping cart, for example) There could also be timing issues, where the script sends a command before a web element has loaded in the browser. A major task in refining an IDE script is adding pause or waitfor commands to resolve timing issues. Recording is dependent on the IDE’s locator strategy. Although the IDE can normally record elements using the ID, name, XPath or CSS, occasionally elements are hidden from the IDE’s locator within Ajax code or they are dynamic so the recoded element names are useless. We have used JavaScript to overcome these issues, but this requires programming experience so that skill must be available or the scripter needs to search the Selenium blog.
To address these issues, experience with Selenium IDE is required so scripts can be fine-tuned with pauses/waits, validations, and comments that can help it run completely and consistently.

An experienced test engineer can create a detailed IDE script in a matter of hours. None of our developers were familiar with Selenium initially, but learned it quickly and easily. Perhaps even more importantly, a non-technical user can use the IDE to create a script fairly quickly as well. This rapid prototyping can be highly advantageous when a quick script is needed for debugging. This is very useful since both the browser session and script are both active after a failure, so the page can be restored and script can be restarted immediately before the point of failure. It’s also useful for scripting utility tasks: create a new user in all test environments; finding and deleting old test data; and creating large transactions or batches to test reports.

While a non-technical user can start scripting quickly, the scripts will be somewhat limited. The scripts we have listed above have many gotoIfs for flow control and some are nested. (These are very difficult to read and debug) The locator strategy for recording elements on a page is good, but sometimes JavaScript is needed. We have probably pushed the IDE beyond its intended purpose, but have overcome limitations with experience using the tool and some help with JavaScript. The key point is that all have been developed by an experienced tester without programming skills.

Selenium WebDriver, on the other hand, has built a package of API’s that are extended and built into several established programming languages. There is a higher barrier of entry to these scripts. Test developers need to have some programming experience in addition to testing analysis/quality assurance experience. We chose to use Java as our language, so as a result needed a java programmer to be able to adapt and build WebDriver scripts.

It is also very helpful for a WebDriver developer to have some experience using Selenium IDE. Some of the testing concepts are similar, and it’s also possible to convert/translate scripts from IDE into Java/WebDriver code.

## 2.6 Higher Maintenance and Complexity due to IDE's limited instruction set

We have pushed the envelope of the IDE by using plugins (See Selenium site for a list of the plugins) and JavaScript. This has enabled us to build extensive regression scripts without a programmer except for some help with JavaScript to perform operations outside the IDE's capabilities. The downside is the resultant complexity and high maintenance. WebDriver and Java provide the opportunity to use the power of the API and Java to perform complex functions natively and to build well-structured code. Some examples will show the contrast.

- **Flow Control**
  The IDE uses a plugin that provides goto, gotoIf and While commands for flow control within a scrip. We have used these for subroutines, looping, skipping tests that fail awaiting fixes and conditional navigation. These scripts can be difficult to follow and to debug. Java provides the ability to create methods, has more powerful flow control capabilities and it is native to the language.
- **Date and Time functions**
  In IDE you need to execute JavaScript to create dates and times and use JavaScript for date and time calculations. Again these are more powerful and native to Java.
- **Data Manipulation**
  There are no native commands for data manipulation or formatting. You need to use JavaScript to convert case, pad, slice, etc. Again these are native to Java.

## 2.7 Selenium IDE's editor is limited compared to a full development language IDE

Selenium IDE's editor is designed for online scripting and inserting test commands and comments. As such, the IDE is very easy to create and run test scripts. It has a very interactive run time environment and it has supported plugins primarily focused at run time and logging. It has a native test suites build facility. The editor does have a command auto-complete capability and it does provide the ability to cut copy and paste commands. However, it is a very limited editor, e.g., it does not have a find or replace. It is not a complete development environment compared to Eclipse for Java. Full function development environments offer full editing capabilities designed to build navigate and manipulate code. One result is the ability to build one WebDriver script to perform all of the testing done with twelve IDE scripts.

## 2.8 Load testing

Selenium IDE is not designed for load testing. We have done light load testing by writing tight scripts focused on iteratively testing an interface, a query or database update and then running the script on multiple PC's or VM's. This is awkward and in our agency getting ten PC's is about the limit. WebDriver provides a much better approach by creating jar files that can be replicated and run. We have run seven concurrent sessions on one PC and could run more until memory is maxed. There is also a Selenium project, Grid, which is designed to run and manage multiple sessions across several servers. Grid has been merged into the current Webdriver. We do not have experience with this facility, but this will be our choice for future load testing.

# 3 Conclusions

After adding one and a half java developers to our test team and building and running WebDriver scripts we have formed some conclusions.

- The IDE scripts that we have built over the years continue to be useful for several reasons. For older applications in maintenance mode, these scripts function well and require minimal maintenance.
- For applications where we are upgrading or adding new functions, the IDE scripts can continue to test the portions in maintenance mode.
- For those applications where we are continuing to use the IDE scripts, converting one or more of the IDE quick scripts to WebDriver provides multi-browser/platform testing and load testing. It also provides a portable JAR file for any user.
- The IDE scripts are still useful for positioning, i.e., getting to a particular page and field consistently in an application is valuable for debugging.

- When converting an IDE script to WebDriver, the existing quick scripts serve as templates to design the new. The IDE regression scripts can be mined to capture all relevant test cases.
- Building a WebDriver script from scratch takes about the same time as converting an IDE script to WebDriver.
- For new software development, we are using WebDriver to develop more advanced scripts. The ability to test multiple browsers, manage the code base, display and save test result metrics and portability outweigh the ease-of-use benefits of the IDE.

For now at least, Selenium WebDriver fits the development and testing needs of our agency best.

# References

Selenium Project website (http://docs.seleniumhq.org/)

JMeter (http://jmeter.apache.org/)

Eclipse (https://www.eclipse.org/)

# MetaAutomation:
# A Pattern Language to Bridge from Automation to Team Actions for Quality

**Matt Griscom**

matt@wisewillow.org

## Abstract

Regression testing automation provides an important measure of product quality and can keep the quality moving forward during the SDLC[1]. Unfortunately, automation can take a long time to run, and automation failures generally must be debugged and triaged by the test automation team before any action item can be considered or communicated to the broader team. The resulting time lag and uncertainty greatly reduces the value of the automation, and increases time cost and quality risk.

MetaAutomation is a language of five patterns that provides guidance to new and existing automation efforts, supplies fast and reliable regression testing of expected business behavior for a software solution, speeds quality communication around the team, and reduces latency and resource cost.

The five patterns, Atomic Check, User Pool, Parallel Run, Smart Retry and Automated Triage, form a sequence, representing an order in which the patterns would apply, and a network of dependencies from the more dependent to the less dependent patterns.

For an existing automation project, the least dependent pattern, Atomic Check, can be applied in whole or in part to run the automation faster and create results which are more actionable. If enough of Atomic Check is followed, the dependent patterns can then be applied to further speed, direct and enhance the value of communications resulting from the automation.

The patterns are language-independent. The talk that presents this paper provides and demonstrates a platform-independent sample implementation of the Atomic Check pattern.

## Biography

*Matt Griscom has 20 years' experience creating software including test automation, harnesses and frameworks. Two degrees in physics[2] primed him to seek the big picture in any setting. This comprehensive vision periodically puts Matt in the vanguard. Matt loves helping people solve problems with computers and IT.*

[1] Software Development Life Cycle

[2] BA in physics from Wesleyan University, BS in Astronomy from University of Washington

# 1 Introduction

In the worlds of Test and QA, automation is about making a software product do stuff. Automation authors answer the question "Have you automated this test?" by making the product do steps of the test automatically. They might add on a verification or two, especially if the test is on an API or service rather than a GUI. If the automation fails, then the automation author must manually determine what is not working, usually by reproducing the failure and stepping through the code. Until then, the test failure is not actionable, other than a vague warning to the team that something is not working. It might be serious, but they do not know yet. In the meantime, part of the product quality goes unmeasured because a test case that would measure it is marked as "automated," and while the automation itself is failing to deliver value for the bit of target quality, manual testers focus their efforts elsewhere. N pieces of automation that fail for an unknown reason thus represent N items of product quality going unmeasured for a time that scales with N, causing project risk that scales as N *squared*.

MetaAutomation includes and enriches automation itself, and it offers a higher-level perspective, a much more expansive horizon. Using techniques of the patterns of MetaAutomation, verification of correct product behaviors will run faster, scale better, and deliver important information *with no deciphering or wait time needed*. The team-wide quality process becomes more robust and interactive, delivering better quality information and making sure that the business logic of your application is solid, so the quality of your software project can always go forward, never backward.

This paper presents the first pattern in the language, Atomic Check.

## 1.1 Why a Pattern Language?

The book *Design Patterns*[3] is the most famous of books describing a set of meta-solutions about common problems in software engineering. As the preface states,

> *Design patterns capture solutions that have developed and evolved over time*.[4]

The book's simple and elegant solutions don't tell you what language to use or what you should name your classes, but they are extremely useful high-level answers to the questions that software developers face when they are choosing designs to build or refactor software systems or subsystems.

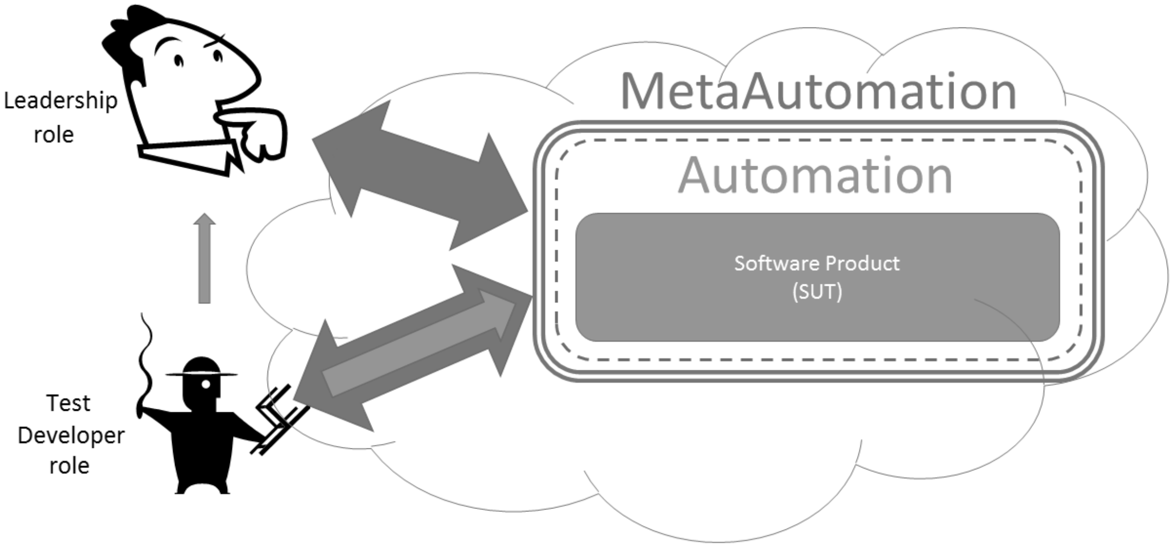
MetaAutomation presents a set of five patterns that address turning product automation into action items and decision points. The patterns have components that many teams have implemented with success, and some new properties that bind them together for the greatest complementary strength. There is a strong and ordered dependency between the patterns, and as a group, they address important issues. For example: If automation is about running the product through its paces in a controlled environment, how does this deliver quality information to decision-makers on the product development team?

MetaAutomation is more than a set of patterns; it is a pattern *language* with dependent and interdependent patterns to address the problem space including software product automation on one side, and decision-makers and decision-points on quality on the other.

The following diagram expresses the relationship between traditional, basic automation and MetaAutomation, with the arrows to represent channels of communication and information flow: the blue arrow for traditional automation, and the green arrow to show what MetaAutomation can do.

[3] Gamma et al., 1977

[4] Ibid, p. xi



This figure shows the relationship between traditional automation and MetaAutomation. One of the benefits of this book is disintermediation: with MetaAutomation, the test developer no longer needs to interpret the results of test automation for others on the team.

## 1.2 Testing vs. Checking

Given that software testing is about measuring, communicating and promoting quality, leadership often sees automation – that is, making your software product do things automatically – as a way of doing all of the above *faster*. Unfortunately, it does not work that way.

People are smart, but computers and computing power are *not* smart. People running user stories, test cases or doing exploratory testing are very good at finding large numbers of bugs, within the limits of attention, getting tired, bored, etc. People are great at spotting things that are not as they should be, for example, a flicker in an icon over here or a misalignment of a table over there, or a problem with discoverability.[5]

Automated product testing, done well, has huge value. Automation is excellent at preventing regression of product quality issues quickly, repeatedly, and tirelessly. Automation does not get tired or bored. Computers are very good at processing numbers and repeating procedures, and doing them fast and reliably, and doing it at times like 3:00 AM local when your team members are home sleeping.

However, automation is *not* good at finding product bugs or anomalous issues like the flickering icon. You need good human testers for that.

Atomic Check is the first of five patterns in the language. It describes ways to do automation well and effectively, run faster and with better scale potential than traditional automation, and with actionable results. The artifacts from following Atomic Check, the detailed, strongly typed, hierarchical results of a

[5] “Discoverability” answers the question: Can an end user discover how to use the product, using only cues from within the product itself (excluding help documents)?

check run are so powerful, they open the quality process to faster, directed and more focused communications with all the data for quick follow-up or even correlation with existing bugs.

This paper uses the term “checking,” proposed by James Bach et al.[6] to define automation that drives tests. A single automated procedure that measures a defined aspect of quality for the SUT is a “check.” The term “check” applies where more commonly a professional in the space might use the term “automated test” but since testing is an intelligent activity done by humans, the term “automated test” becomes an oxymoron; once a test is automated, it is no longer a test in the same sense. If done well, it is fast, reliable, tireless and highly repeatable, but the value is very different from the same procedure run manually by a testing professional.

If “testing” generally is about measuring the quality of the product, then “checking” is a very important, specialized subset of testing. Designing and developing checks takes significant testing skill.

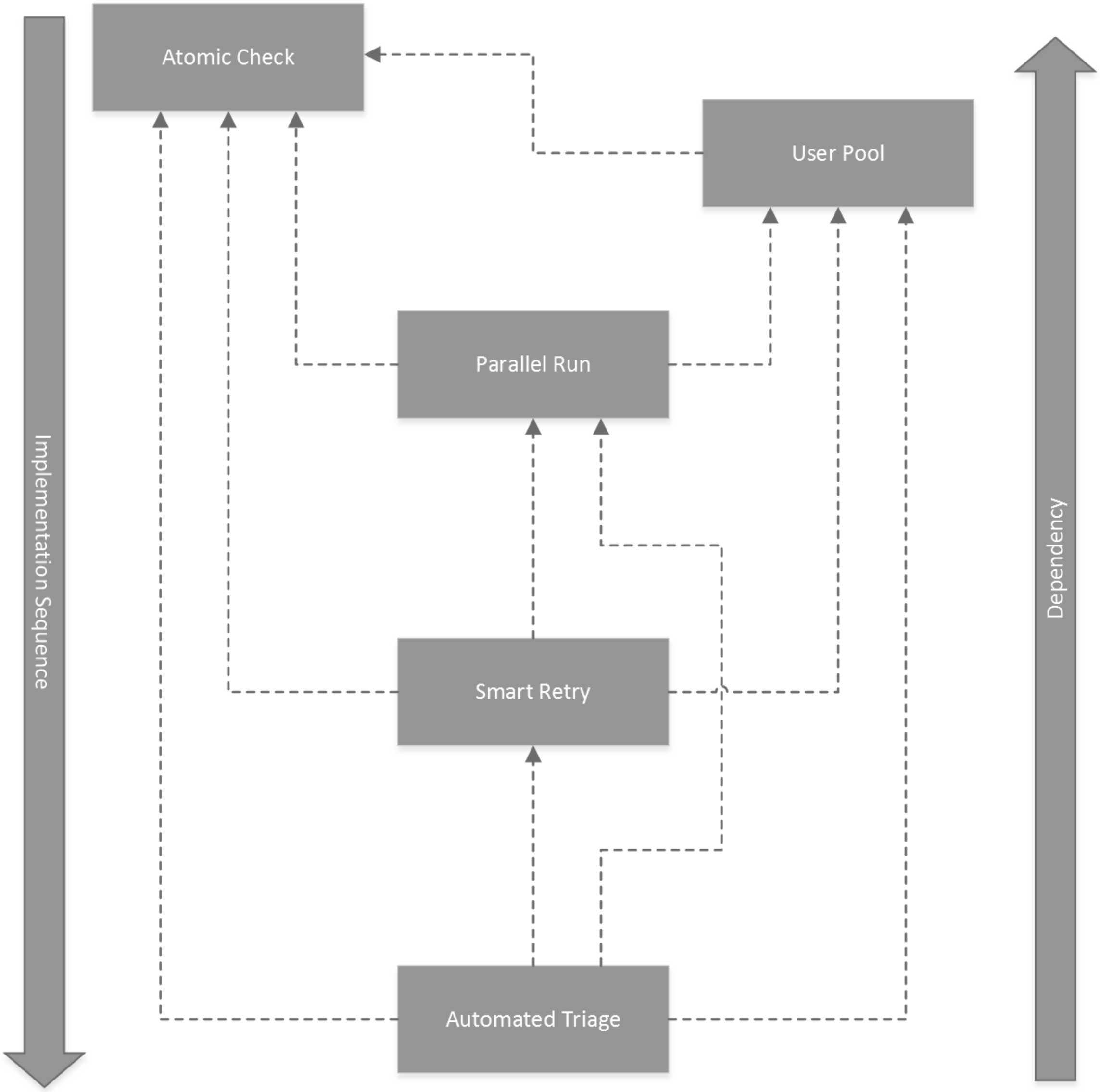
The term “check” emphasizes an important reality of developing modern, impactful software: the team needs both manual testing and automation. One does not substitute for the other. However, if the checks are well defined and implemented, and they are passing, they obviate manual checking of the targets of the checks.

The name of the pattern described in this paper, Atomic Check, reflects the shift away from using the term “test” to reflect automation. However, this paper uses both “test” and the deprecated term “automated test” because they are still common in the vernacular of the industry.

## 1.3 Relationships of Atomic Check to Dependent Patterns

Due to limited time and space, this paper and the associated talk focus on the first pattern in the MetaAutomation pattern language: Atomic Check. The following diagram illustrates the sequence and dependency network between all five patterns.

[6] Bach et al., “Testing and Checking Refined”



## 1.4 Business Continuity

There are many reasons for which your business might suddenly find itself without the team member who is most knowledgeable of the automation project.  This is a well-understood risk; what if it happens to you and your team?

Applying the Atomic Check pattern, along with the four other patterns of MetaAutomation, reduces your risk in these ways:

- Checks are as short and as simple as possible, minimizing maintenance time by reducing the number of potential failure points and reducing the length of time a check run requires.
- Check failures are highly actionable, reducing dependency on automation authors as intermediaries.

- Automation should use the same language as the business logic of the product where possible. This greatly reduces the mystery associated with the finer points of non-standard languages and assures that there is in-house expertise if needed.

- Coding the automation with an IDE and in-house code, just as the software product does, is a natural approach. This lowers the incidence of blocking behavior limitations or bugs that cause the team to wait on external factors. Mysteries can be resolved, if needed, by an investigation or debugging session by experienced in-house talent.

- 3rd-party test automation tools are probably unneeded. This reduces licensing costs and/or external dependencies which can cause mysterious failures or external blocking bugs. Mysteries can be resolved, if needed, by an investigation or debugging session by experienced in-house talent.

- Self-commenting coding techniques as part of creating actionable failures for the pattern. This creates readable code and largely eliminates the need for traditional code comments which can become misleading or incorrect.

- Applying the full pattern language allows all interested team members to interact with the automation, which distributes knowledge of the tools and capabilities more evenly around the team.

# 2. Atomic Check

The Atomic Check pattern describes how to design and write a check that measures a simple and important aspect of product behavior. The pattern is a guide to maximizing the following values:

- The actionability of a check failure, so that decision makers will know what to do about it
- Scaling the check run across machines, virtual machines or operating system instances
- The speed of verifying the business behavior

While minimizing these aspects:

- The number of debug sessions required before a given check failure becomes actionable by any team other than the one that owns the check code
- The duration and scope of any part of product quality going unmeasured due to a check failure
- Latency in communicating product quality issues

Checks following the Atomic Check pattern are end-to-end and may include integration, regression or performance checks.

## 2.1 Basic Requirements of Atomic Check

The Atomic Check pattern requires that each check verify just one thing about product behavior, plus any steps or product behaviors that are required to support the verification that is the focus of the check.

Each check must be as repeatable as possible in how it drives the SUT. There is no place for random, pseudo-random or quasi-random values.[7]

Each check is designed to be independent of all of the other checks. Checks can be run singly or in sets in any order, in series or parallel.

“Fail soon, fail fast” summarizes these requirements of the Atomic Check pattern:

- Detect a failure condition for the check as soon as possible. This is important for root cause. For any statement or line of code in the check that could fail, if earlier or simpler statement could detect the failure and report it, then there must be such a statement in the code. For example, your code could assert that an object is not null or that it can be found, with an informative message formed and reported for the case of assert failure.
- Make the check fail as fast as possible on detecting a failure condition. The most common pattern for making this happen is to create and throw an exception that fails the check. For example, the Assert idiom does this by throwing an exception.

The check must fail with detail and accuracy. All check failures need to be traceable to their root cause, at least in the context of the check code or harness.

The check logs – A.K.A. flight-recorder logs of what the check is doing - must be separate from the record of failure for many reasons, starting with the fact that if the logs are useful at all, they are only useful at check development time. Logs do not otherwise get stored past the check run, but in case of check failure, the harness running the check will archive the artifact of the check run for later perusal and automated triage.

Just from these basic requirements, it follows that Atomic Check requires either deviation from product coding styles and requirements, or another set of coding styles and recommendations.

## 2.2 Automation is for Checks, rather than Tests

Automation for measuring product quality is indispensable for an impactful software product. Without it, the quality risks are too great:

- A business logic failure or incorrect behavior can impact the workflow of the team.
- An incorrect behavior can cause additional downstream bugs, as developers “work around” the failure.
- The development team can get distracted (A.K.A. “randomized”) by the need to go back and fix an issue that was introduced some time before.
- If the issue impacts a customer before the team is aware of it, crisis management can take over to fix the issue, with its own dissonance, costs, risks and delays.

Checks, written and run well, will notify the team quickly of quality regressions, thus avoiding these and other risks.

However, one limitation with automation is that, other than the default point of verification (did the application *not* crash?), the only points of verification are what is written in the code, implicitly or explicitly. Most points of verification must be written explicitly, with reporting that describes root cause for the failure case.

[7] Testers use these sometimes in an attempt to increase test coverage, but are really only suitable to fuzz testing, and even then, the random values must be persisted somehow to make the tests repeatable.

The term “check” reflects the target verification of the automated test. For example, if the team automates a verification that a user can change the preferred color scheme for a client app and that this change is persisted successfully, then this piece of automation “checks” that the change persists. The preliminary steps – logging in, going to a settings page, changing the preference etc. – have preliminary assertions.[8]

## 2.3 Check Setup and Teardown

Modern automated test frameworks provide for a test setup (initialization) and a test teardown (clean up), separate from the main body of the test. If these methods are implemented, they run just before and after the main test, and on the same thread.

Atomic Check does *not* allow these methods because analysis of a check failure would be difficult or impossible if there was a failure in the test setup or test teardown. Atomic Check only details and persists failure in the scope of the check method, not in setup or teardown. Using a class setup (a setup for a set of checks) might also interfere with the ability of checks to run independently of each other on different client processes.

Move all setup steps in line with the main procedure of the check. Setup steps are treated just like any other preliminary step of the check, and the artifacts related to check failure are treated the same way; they may be actionable bits of information there for people or automated processes to follow up on, to analyze patterns of failure or just get the checks running again.

Using a setup method pattern would make it impossible to apply the Smart Retry pattern later. Given that a check failure happens with an exception thrown out of the check method scope, a failure in a setup method cannot generate artifacts in the same system that will enable root cause to be determined. The check would fail, but the artifacts would be missing failure cause information.

One could argue that the historical reason for using the synchronous setup idiom is that a failure “aborts” the automation, rather than fails it. The reason to have an “abort” is that it clearly signals that the failure is not an action item for the product developer team or role; the automation author or authors must fix it. However, the team no longer needs this signal: with the Atomic Check pattern, the artifact answers the question of who needs to follow up on a failure (if anybody). With complete MetaAutomation, if there were an action item for a person on that team, that person would receive notification (e.g., by email) with a link to an intranet site with all of the information needed.

Another historical reason for the setup idiom might be the idea that using setup and teardown methods will ensure that a failure of the main check method always has failure in the SUT as root cause. This would simplify triage; the idea is that if the main check method fails, it is clear that the product developer has to fix it. This reason simply is not valid anymore; with modern software applications, dependencies are much more complex than they used to be. For end-to-end tests, generally there are many factors outside the SUT, which might cause a check to fail in the main method of the check.

If your automation requires a significant clean-up after a check where any issues that might happen during the clean-up are *not* related to the SUT, consider using the User Pool pattern or other asynchronous method if that can free up the thread (and associated resources) more quickly.

In summary, although many automated test frameworks offer setup and teardown methods, the historical reason for offering these methods is no longer valid, and the Atomic Check pattern requires that you not use them.

[8] See section 2.8.1

## 2.4 Outputs from a Check

The runtime process of MetaAutomation starts with what information comes from a check.

If the automation drives an application GUI, it is possible that a knowledgeable tester staring at the screen during the entire automation run can get some useful information out of the automation run. (I have seen this done.) However, this is very expensive, error-prone and risky, and any information coming out is not likely to be of very high quality or complete.

For an application GUI, the harness can take screenshots, and for some specific applications, these can be useful, but automated parsing of screenshots into something actionable is something I have never seen done successfully.

An important part of MetaAutomation is focusing the information from a check on maximizing value for the consumer of this information. Consumers, i.e. customers, of this information fall neatly into two roles: automated test developers who are in the process of developing or maintaining the checks, and decision-makers[9] who receive the information resulting from the check run.

The Atomic Check pattern focuses information flow and value for the different needs of these two roles.

Everybody who has an interest in quality information about the SUT from a check will want to view and act on the artifacts, i.e., the results of a run of that check. This information includes whether the check passed or failed, and in case of failure, enough information to identify the root cause of the failure.[10]

At check development time, the developers of the automation might need more details from the procedure as it is executed, i.e., what steps are completed and when. This “flight-recorder” log helps correlate automation code and what is happening as it drives the product. Stepping through code in an IDE can provide this information, but for a non-trivial check the steps are useful in the artifacts should the check fail. The sample project provided gives an example implementation for this purpose and supports check steps in a hierarchical dependency of any depth.

Logs and artifacts are two distinct streams of information, as shown in the following table:

| | **Logs** | **Artifacts** |
|---|---|---|
| **Creation time** | The harness streams logs as the check runs. | The harness persists artifacts at check completion. |
| **Source of the data** | Logs stream actions that the check performs as it drives the SUT. | Artifacts hold information on the check, plus failure and specific detailed context to express root cause for a failed check. |
| **Role of the data customer** | Logs can be used by automation authors to manually follow up on failures. | Artifacts are focused and concise enough to present with an XSL stylesheet to any decision-makers on aspects of product quality, and they are |

[9] E.g., managers, leads, and product developers

[10] Please see section 2.6 for more information

| | | |
|---|---|---|
| | | strong enough for automated parsing and comparisons. |
| **How the data is presented** | This is flat but usually readable text in a log file or an output window in an IDE. | Artifacts are data only, no inline presentation. |
| **How the data is persisted** | This data is not used after a development session is over, so can be discarded. | This data is persisted at least for the duration of the SDLC, possibly longer. |
| **How the data can be used** | People create logs to be readable by people, but they tend to fill with lots of information that is not useful. | Artifacts support fast and robust automated triage and analysis, and an XSL stylesheet can present the data very nicely. |
| **The importance of the data** | This can be an aid for developers at test development time, but it is not required. | This is a basic requirement for MetaAutomation, so without this data, the value of test automation to your software project is limited. |

In case of a check failure, in order to make the failure actionable, the stage of the check that failed is useful information that will help in failure analysis (whether automated or not) or correlating check failures. However, in order to be useful for humans and automated processes (as seen with the other MetaAutomation patterns) to triage, it is better to leave out the flight-recorder log of every step that the check performed in order to get to the point of failure; that quickly gets too verbose for automated parsing. For applying the Smart Retry and/or the Automated Triage patterns, it is especially important to keep information about the failure concise; unnecessary information might cause either of these patterns to fail. To support automated parsing and succinct logging for a complex check, consider expressing sections of the check procedure in hierarchical steps[11].

To support long-term storage of the check result artifacts and automated parsing, querying and comparison, the Atomic Check pattern considers logs and artifacts to be two separate entities.

## 2.5 Logs

The logs from a check are useful at check development time and maintenance time.  The Atomic Check pattern helps create simple checks that minimize maintenance time; however, product changes that require changes to the check are still possible.  For a complex check, logs might help at development time.

[11] The sample implementation demonstrates support for hierarchical steps that go into the structured artifact in case of check failure.

Because logs are streams of text, they tend to be verbose and difficult, expensive or impossible to parse reliably. The Atomic Check pattern therefore recommends that they be discarded at completion of any check run.

The working sample of an Atomic Check implementation provides for hierarchical check steps that output to a log *and* are stored in a structure in the artifact in case of check failure. The check developer can use steps like these instead of simple log statements, and it will help at both check development time and at artifact triage and analysis time.

Atomic Check recommends discarding simple log streams when the debug or development session is finished.

## 2.6 Artifacts

Artifacts are the check results that are stored for some length of time for later review, manual analysis or automated analysis. They record the name or unique identifier for the check, the “pass” or “fail,” start time and duration, but much more than that as well. In case of failure, the harness records the point of failure with enough information to identify root cause.

Running many checks, and running them frequently, generates large numbers of artifacts for the check result data. Keeping that data compact allows you store more of it and do more subsequent analysis on the data. The sample project provided uses XML.

In the case of storing the artifact data in a flat file system, XML is ideal because it is:

- Text-based
- Compact[12]
- Extremely scalable
- A well-known and well-supported W3C standard

With XML element names and attributes, the data is automatically future-proof and can be extended as needed without loss of information or risk of data being unrecognized due to schema changes.

XML is also a W3C[13] standard, unlike JSON, and has the standard presentation language XSL to make the artifact data accessible for human viewers.

Mostly, the data that goes into the log for use at development time is different from what the harness saves as an artifact because the uses and purposes of the data are so different. However, there is one bit of information that serves well in both destinations: what the step (or hierarchical steps) of the check are at point of failure.

## 2.7 Dependent Operations of an Atomic Check Implementation

The focus of a check that follows the pattern is an atomic verification, which usually consists of one or more assertions. That is not all that the check does though, because in any dynamic system there will be dependent operations.

[12] And compressible

[13] The World Wide Web Consortium

For example, if we are testing a REST service for the ability of a certain account to log in, the focus of the check is successful login, according to the criteria we specify: the message received *and* a given session cookie.  The message and the cookie are both requirements for success.

The dependent operations in that case include:

- Test infrastructure
- The client opening a network connection to the service
- The client creating, serializing and sending an HTTP request
- The server successfully generating a response
- The response has a compatible HTTP status (probably a 200 OK)
- The response body is successfully deserialized

The check includes assertions which might be implicit, that is, not directly expressed in the code.  Check developers might choose to make the assertions explicit to ensure that in case of failure at a given dependent operation, the artifacts are informative enough to identify root cause in the context of the check run.  An alternative is to supplement the artifact related to a check failure with server-side log data around the failure, to help determine or disambiguate root cause.

The Atomic Check pattern still applies, though, because the focus of the check is still one thing: successful login.  Testing that one thing cannot be done in a simpler way or with fewer potential failure points.

## 2.8 Points of Verification for Atomic Checks

The target of the check probably requires some steps to get there.  Most such steps are prone to failure - or, if you take potential coding errors in the automation code into account, all of the steps could potentially fail.

The term “assertion” is useful here because it refers to all assertions that happen in the check, including ones that could be missed on casual code review, for example, null-reference exceptions (null-pointer exceptions for a native runtime) or required properties that do not default.  The assertions that aren’t visible in code that is owned by the team or the project doing software checks are “implicit” assertions; they aren’t coded for explicitly in code authored by that team or project.

Preliminary steps lead up to the target of the check.  For example, opening a network connection (or some higher-level object that abstracts this out) is probably a preliminary step.  The target of the check verifies a piece of important business behavior, and is the purpose for which the check exists.

### 2.8.1 Preliminary Assertions

On check setup or initialization, please see section 2.3 on Atomic Check Setup and Teardown.

There are two types of preliminary assertions:

Implicit preliminary assertions are part of the preliminary check steps and do not require any extra code from the check author, and explicit preliminary assertions that throw an exception with a failure message on check failure.

For example, if a network connection made as a preliminary step times out, then the network timeout exception that fails the check is a result of an implicit assertion, i.e., that the network connection can be made and does not time out.

Explicit preliminary assertions are assertions that help the check fail faster than it would otherwise, and with better information.

For a simple example, if a preliminary step requires creating an object of type User, and if the User object creation might fail returning null, the check would fail by default at the following step, which did something with that User object. In this default case, though, it would fail with a null reference exception or something similar (depending on your programming language and framework). This failure of the check could even be ambiguous, too, with only that information, and therefore not actionable because root cause is not clear. This would require follow-up with a test-automation-and-product debug session by the check author or someone else on his/her team.

To implement the Atomic Check pattern, this example needs at least one explicit preliminary assertion. The check asserts (or uses equivalent test logic) that the User object is not null, with informative error information about that to throw with the exception, including the User object name or ID and any other useful information.

The check requires another preliminary assertion if the User object has some property that is required for the check to succeed. For example, if the check would fail later in case the User object is not currently active, then put in an explicit assertion for that property, with error information.

Explicit preliminary assertions must never cause the check to fail if the check were not going to fail anyway at a later step. However, if the check is in a state of imminent failure, and this is measurable in the context of the code, Atomic Check requires adding an explicit preliminary assertion to fail the check faster and with a more actionable message. Otherwise, there is only the implicit preliminary assertion, which might fail occasionally with neither information closest to the root cause of failure nor an informative message with the exception object. In the latter case, the team might have to wait on the automation author to follow up, try to reproduce the failure, and find root cause before it is determined whether an action item exists for anybody other than that author or whether there is some unmeasured part of product quality.

The case where a check might fail with incomplete information for the customers of the check automation is exactly the one where the check needs an explicit preliminary assertion. An explicit preliminary verification helps the check fail sooner (i.e., faster, in the scope of running many such checks) and with more specific information to help speed communication around your Atomic Check implementation and make the dependent patterns of MetaAutomation possible.

### 2.8.2 Target Points of Verification

Each check following the Atomic Check pattern has one target verification. A target verification answers the question, “Has the SUT passed the check?”

This might just be the final assertion of the check, but may also be a little more complex. Maybe it has to be two assertions rolled into one, depending on the bigger picture into which your check fits.

At check design time, you will have to ask this related question: “For the target behavior of this check, and from the point of view of clients of this behavior, what does success mean?”

The target verification might be just one assertion, or it could be two or more.

Two assertions - call them A and B - cluster together as part of the target verification of an atomic check if they meet all of the following criteria:

1. A and B are both important to business behavior (or, client needs)
2. They are *not* dependent on each other in terms of the verification steps
3. They *are* logically linked together. If A should fail, the state of B at check failure time must be recorded in the artifact as well because this affects the action item that might result.
4. Between the measurement of A and the measurement of B, there is no interaction with the product or any external entity over the network (and hence no transient failures that might be eliminated from the check result summary with the Smart Retry pattern).

The clients of the subject behavior of the target verification would need the behavior in order to continue a scenario, beyond the scope of the check. Therefore, an assertion on something that is not required to continue the scenario (if only hypothetically) beyond the scope of the check is *not* a good candidate to include in the set of assertions in the target verification.

The Atomic Check minimizes parts of the product that go untested. This means that the target verification might require reporting more than just the result of a simple Boolean verification, even while at the high level, the result of the check is Boolean: pass or fail. The following example clarifies this:

Suppose it is a real estate web site, and you want to automate a check that verifies that search results on the public-facing site will show properties for sale. The Atomic Check pattern requires that the check be as simple as possible and that it verifies something about the site that is a prerequisite for customers to proceed.

The check has these steps:

- Start a web browser with the URL of the site
- Click as needed to get to the property search page
- Change criteria as needed to get search results
- Verify that at least one real estate property is shown for sale

The target verification includes ensuring that the page shows at least one real estate property.

When implementing this check, a developer needs much more detail to make it happen. People are smart, but computers aren't, so to get from the people-friendly four steps listed above to something that the automation needs, some steps are added (in italics):

1. Start a web browser with the URL of the site
2. Click as needed to get to the property search page *translates to*
    a. *Find the menu item on the page that shows the choices needed to proceed*
    b. *Click the menu item*
    c. *Find the "Property Search" submenu item*
    d. *Click the submenu item*
    e. *Wait for the Property Search page to load*
3. Change criteria as needed to get search results *translates to*

    a. *… (maybe nothing is needed to be done here)*

4. Verify that at least one real estate property is shown *translates to*
    a. *Find the HTML item that shows the number of properties that result from the query*
    b. *Read the InnerText of that HTML item*
    c. *Parse the text for the desired number*
    d. *Assert that the number is greater than zero*

Any one of these steps could fail, due to check dependency failures, check code failures, or of course failures from the SUT.

One of the goals of the Atomic Check pattern is to minimize the need for testers to follow up on a failed check with an intensive debug session just to answer the question “What happened to cause the failure? Is this a test or a product bug, or something else?” Debug sessions on failed checks are very expensive and time-consuming, and as a result, they might not even happen at all. Even if they do happen, they cause a delay and some disorder in the productivity of the team. The result is reduced value in the checks: they can still sometimes verify correct product behavior, but the team will tend to view failed checks as a test-owned failure rather than a product issue, and the manual debug session that might turn a failed check into a fixed product bug is a barrier to the product team.

Therefore, it is important to ensure that the artifacts from a failed check contain the information needed to know *if* someone needs to follow up, or at least maximize the chances that the artifacts are sufficient that the importance and impact of the failure is known, without a manual debug session.

Suppose, for example, that with our check on the real estate site, the check failed at step 4c: the parse failed for some unknown reason. If the target verification is step 4d, then the target verification did not happen. It appears that an implicit preliminary assertion failed, and the artifacts that result from this check failure include a format exception and a stack trace that points to a file and line number in the source code. With that information alone, there’s no way of knowing whether the root cause of the failure comes from the SUT or the check code, so the check result will be ignored until and if somebody on the test team can follow up. If there is a product bug, it is still undiscovered, although if there is a pattern of this check succeeding before, that change in the pattern of success is significant.

For this check on a real-estate web site, if step 4d – verifying that there is at least one property returned in the ultimate search – succeeds, then that is sufficient to pass the check. The end-user could continue in theory from that point with a further scenario because there is at least one property to look at. Step 4d is the target verification.

Consider the relationship between steps 4a, 4b, and 4c; if step 4a fails, step 4b is meaningless, and if 4b fails, 4c is meaningless. It is clear that steps 4a, b, and c are preliminary steps, along with any part of steps 1-3.

However, the implicit preliminary assertions of those steps as written are not sufficient. Back to the example of a failure at step 4c: if the check fails with an exception from trying to parse the text, there still is not nearly enough information to decide whether the check failure is actionable or who needs to take what action.

Suppose the text value is “four,” and that is consistent with design of the product. The failure to parse that string into an integer would then be a bug with the check code.

Suppose the text value is a zero-length string. That might be a significant and actionable product bug, but the check result does not show that, so the check result itself is not actionable and the product bug is unknown for now.

This check needs explicit assertions so that enough information goes into the artifact associated with the failure, such that the results of the check are actionable. Another way of looking at this issue is that, when there are a large number of check results and many failures, it is important that the check artifacts discriminate between the zero-length-string issue and the “four” issue above.

To cover the possibilities for steps 4b and 4c:

Add an explicit assertion to step 4b that the inner text of the HTML element returns a string with greater than zero length. This may be browser, or framework-dependent, but it is probably safe that accessing the inner text of an HTML tag will not by itself fail, e.g., with a null-reference exception.

Add an explicit assertion to step 4c so that if the parse should fail, the harness reports this as part of the check run artifact with the actual string. We need to know if the string that is expected to contain the integer we want is “four” (which would be a check bug, if the product design allows for that kind of behavior for the SUT) or perhaps “1,0,01” (which would represent a product bug).

Consider again the example from above: verifying successful login over a REST service. The final conditions for a successful login include:

1. The HTTP status of 200 OK
2. Verification that the session cookie is the correct type
3. Verification that the body of the response includes the expected indication of login success

The Atomic Check pattern allows a set of grouped target assertions if all these conditions are true:

1. This is no interaction with the SUT or any outside system that could introduce latency or another point of failure after the first assertion in the set and before the last.
2. There is a logic reason related to functionality of the product to group the assertions together.

In case of check failure resulting from any assertion in the set, the failure report must include the result of all assertions in the set. This is part of what makes the check atomic: assertions in the set fit together as one, and in case of failure, the harness reports all product information resulting from that assertion set, no matter the results of any individual assertion in that set.

This type of login verification can still be an atomic check, even if it has multi-part assertions at the check target - that is, that login is successful. But, the correct cookie and the login success status both depend on the response coming back with an HTTP status of 200 “OK”; any other status is not compatible with a successful login, and in addition, the HTTP status is (or should be) measured before an attempt is made to stream the body of the response off the server. Therefore, the verification of HTTP response status of 200 must be a separate assertion. The cookie and the response message, though, meet the criteria for a set of grouped assertions in the target verification.

Therefore, the final assertion set of the check includes a check on the login cookie and a check on the body of the response. The cookie assertion and the body assertion must both be done, and then the check fails if either or both of these checks fail, and in any of these failure cases, the report generated at check failure includes information about both assertions in the set.

The focus of this check is the final assertion set, which answers the question: was the login successful and correctly formed according to the designed product behavior? Given that there are two Boolean assertions in the set, there are four possible outcomes:

1. Login was successful and correctly formed, so the result for this positive check run is "true."
2. Login was successful according to the body of the response, but the cookie is absent or does not meet some criterion for success. The check run result is "false," and the artifacts include this information from the target verification:
    a. That the body was as expected
    b. That the cookie did not meet the success criterion
    c. Information about the cookie including details on why the criterion were not met
3. Login was not successful according to the body of the response, but the cookie looks correct. The check run result is "false" and the check run artifact includes this information:
    a. The fact that the body did not meet success criteria
    b. Details on how the body did not meet criteria
    c. The fact that the cookie met the success criteria
4. Login was not successful, and the body and cookie were not as expected for the run of this positive test. The check run result is "fail" and the artifacts include all of the information about the assertion results in that final verification set.
    a. The fact that the body did not meet success criteria
    b. Details on how the body did not meet criteria
    c. That the cookie did not meet the success criterion
    d. Information about the cookie including details on why the criterion were not met

In case of check outcomes 2, 3, or 4, the code for the target verification places all of the information described into the exception and thrown to fail the check. In case of check fail, the artifact generated for this check run would contain information on the state of the cookie and of the response body. For a check that verifies login details for a custom REST service, analysis needs all of this information to describe the login failure and follow up at triage and analysis time or at runtime for the Smart Retry pattern.

In addition, if the check code and harness implement hierarchical steps as with the sample project, a failed check will include hierarchical steps leading up to the failure to clarify the context of the check failure event.

Given that the server side of the login is also part of the SUT for this example, what is going on with the service is important as well. If information from the server side (from a log or custom instrumentation) is available synchronously (or even *nearly* synchronously, with a short wait on the information) then those details can go into the artifact as well as a separate, strongly typed unit of information (e.g., in an XML element, as is done with the sample project).

The resulting artifact has sufficient information to make root cause apparent, with no debugging sessions needed. The techniques outlined here scale to more complex tests, to enable actionable results and application of the four following patterns of MetaAutomation.

# 3 Conclusion

Atomic Check is the first of the five patterns of the MetaAutomation pattern language.[14]

Applying this pattern, or adapting current practices and infrastructure to the pattern, will bring greater effectiveness to the quality team and value for the software product. Automation will run faster, scale better, and give decision-makers for quality what they need to know.

Looking forward to the following four patterns of the language – User Pool, Parallel Run, Smart Retry, and Automated Triage – offer potential paths to further acceleration of communication around the team, greater perspectives and broader horizons on current automation practices around software quality measurement and regression testing.

# References

## Books

Alexander et al., *A Pattern Language*, 1977

Binder, Robert V., *Testing Object-Oriented Systems: Models, Patterns and Tools*, 2000

Gamma et al, *Design Patterns*, 1977

Garg & Sharapov, *Techniques for Optimizing Applications - High Performance Computing,* Prentice-Hall 2002

Graham, Dorothy and Fewster, Mark, *Experiences of Test Automation: Case Studies of Software Test Automation*, 2012

McCaffrey, James, *Software Testing: Fundamental Principles and Essential Knowledge*, 2009

Meszaros, Gerard, *xUnit Test Patterns*, 2007

Myers, Glenford J., *The Art of Software Testing*, 1979

Page, Johnston, and Rollison, *How We Test Software At Microsoft*, 2009

Riley & Goucher, *Beautiful Testing*, 2010

Whittaker, James, *How Google Tests Software*, 2012

Kaner, Cem, *Exploratory Testing*, Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando FL, November 2006

## Periodicals

International Software Testing Qualifications Board, Standard glossary of terms used in Software Testing: Version 2.2, 2012

## Blog Posts

James Bach and Michael Bolton, “Testing and Checking Refined”
http://www.satisfice.com/blog/archives/856

[14] See the book MetaAutomation by Matt Griscom, to be published in 2014, for more information

# Enabling ETL Test Automation in Solution Delivery Teams

**Subu Iyer**

subu.iyer@yahoo.com

## Abstract

Most companies today deal with lots of data. Typically, this data is in several databases and across multiple applications. Companies also have to work with data from vendors and clients. To integrate data from multiple sources, Extract Transform and Load (ETL) systems are used. These ETL projects are handled by Solution Delivery teams, where each project has its own specific requirements and constraints. This makes it very hard for teams to build automated tests and processes around the ETL solutions that would help them consistently deliver high quality solutions on time and on budget. These and a set of outdated tools and practices make testing ETL jobs a time consuming and labor intensive manual process. The lack of automated testing also makes it very difficult to build and deliver incremental changes and this hurts productivity.

This paper describes how our team, the Quality Engineering and Specialized Testing (QuEST) team, is driving a change in the way teams build and test ETL jobs at Cambia Health Solutions. We are doing so by engaging with the Solution Delivery teams, setting up automated tests and providing teams hands-on training so they can continue writing automated tests. This paper will also talk about the benefits of building and growing the test framework organically in an incremental fashion. We'll see how our team was able to overcome initial challenges and improve our ETL testing.

## Biography

*Subu Iyer has been working in Software Quality Assurance for 10+ years. Subu believes in building quality upfront and setting up processes that help us deliver quality products. At Cambia Health Solutions, Subu works with teams to build automated tests and sets up automated processes that make teams more productive.*

*Subu has a Masters in Computer Science from the University of North Texas, Denton, TX. He co-owns PDX FitNesse and Test Automation Users Group to discuss FitNesse as a tool to drive development and improve productivity.*

# 1 Introduction to ETL and ETL Testing

Extract Transform and Load (ETL) processes are done to integrate data from different applications and vendors. ETL is a process where data is extracted from a source system, transformed and loaded into a target system. ETL jobs import data from flat files into databases, copy data from one database to another or export data from databases to flat files. Before the import or export, there are usually a few data transformation steps required to derive the target data from the source data. Some examples of ETL projects at Cambia Health include importing employee data from flat files, copying claims data from one system to another and exporting member information from a database onto flat files. The challenge in these projects is to accurately move data from one system to another while maintaining or improving data quality. In some cases, complex data transformation rules make the implementation complicated. To add to these, typical ETL projects handle huge volumes of data and this means that these solutions have to be efficient and scale well.

The target applications depend on the data load for their functionality. In other words, testing of ETL jobs is very essential to ensure that the target applications continue to function correctly. Conceptually, ETL testing tests the functionality of the components built to perform the extract, transform and load steps. Testing these components consists of setting up test data, running the ETL process and verifying the actual results with expected results. Automating ETL testing is exactly the same – set up test data, run the ETL process and verify the results.

At Cambia Health Solutions, the Quality Engineering and Specialized Testing (QuEST) team is driving the change to help Solution Delivery teams be more productive. Most ETL projects in our company lacked formal testing. We are changing this by collaborating with the teams on test automation, building a common framework for ETL testing and conducting training sessions.

# 2 ETL Solutions and Testing

Once the implementation plan is laid out, ETL development tools with their intuitive and simple graphical tools provide an easy way to build ETL components and work flows. ETL tools also come with a powerful set of data transformation capabilities. The graphical tools and built-in libraries make it easy to put together the ETL functionality. However, the lack of testing tools within the ETL development tool kit makes unit and functional testing of these jobs very difficult. In many cases, during the project planning phase, the testing effort is not fully understood and this leads to an inaccurate estimate of the testing tasks. The lack of test planning and the lack of formal tools result in the lack of systematic automated test processes. This reduces testing to visual inspection and spot-checking of data.

With no automated unit and functional testing, any defect fix or enhancement can only be verified by manual end-to-end tests. Manual tests are time consuming, error prone and don't provide quick feedback about the product being tested. Manual tests can't be used as a tool to aid incremental development.

# 3 Challenges to ETL Testing

The concept is very simple, but when it comes to actually building automated ETL tests, there are a few challenges. Let’s go over some of these.

### 3.1 Testing Tools

Typically, ETL development consists of defining data components, mapping systems and setting up jobs using commercial software. These commercial development kits with their GUI tools make it easy to build complex ETL components quickly. The code for the ETL job is auto-generated by these development kits. ETL testing consists of setting up test data to test the logic in the ETL jobs, executing the ETL jobs and running SQL queries to validate results. In most cases, ETL jobs map a hundred or more fields from one system to another. Setting up test data is a time consuming activity. In addition, complex data transformation rules can make it very challenging to set up test data. The lack of tools to automatically generate good test data also makes testing a more time consuming activity than development.

### 3.2 Technical Skills

ETL testing requires a good set of technical skills. There is no visual front end for testing ETL as with most functional testing. While the user interface in target applications can be used for spot-checking data, such testing is insufficient and ineffective. ETL testing requires a good understanding of the data model, SQL skills and scripting skills to generate test data.

### 3.3 Development Model

At Cambia, ETL development still follows the traditional waterfall model. Development and testing tasks are scheduled in sprints, but they are in sequence. In a typical ETL development cycle, the components are built in a sprint and tested in the next. Ideally, we'd like development and testing in the same sprint with testing providing immediate feedback about issues in the components and requirements. The lack of automated tests and no quick feedback loop between developers and testers make true agile, incremental development very difficult.

### 3.4 Manual Testing

As testing lags development by a sprint or two, defects are found late. Running manual regression tests following defect fixes can be very time consuming, but necessary. This manual testing activity also prevents the team from working on automated tests.

### 3.5 Change is Difficult

In general, humans tend to resist change and hide behind excuses like - “Our ETL is too complicated to test” or “We have no time to test” or “Our ETL testing can't be automated”.

## 4 Misconceptions about ETL Testing

There are many widespread misconceptions about how ETL jobs should be tested. Let us look at a few of these.

### 4.1 ETL Tests are Complicated

There is a common misconception that automated ETL tests need to reproduce the ETL job in test code. This might work in an ad hoc test when spot checking data. However, such test queries can get very complicated and unreadable. Using such queries in automated tests only leads to unmaintainable tests.

### 4.2 ETL Testing is Data Testing

Checking data quality is a good audit step, but this is not the same as testing the ETL functionality. This misconception might be due to the fact that testing starts after the ETL job has been built, run and the target data has already been loaded. Now the tests try to make sure that the target data is right. This might work as an audit step, but without full functional testing of the various components in the job, testing is incomplete. Most ETL jobs operate on at least hundreds of thousands of rows of data. It is impossible to validate all of this data every time the ETL job is run.

### 4.3 ETL Projects Don't Need Regression Testing

ETL projects are built as custom solutions to very specific problems. This leads to the belief that once they are built, they don't need to be revisited or tested for regressions. As a result, ETL projects rarely plan for test automation. However, there will always be changes to the requirements even after the product is released. There will always be defect fixes. And even if we magically built a perfect product, things around the product will change and such changes will require changes in our product too. Without automated regression testing, making any changes to the existing product is very risky.

These misconceptions make it very difficult for anyone to get started on ETL test automation. Two years ago, when our newly formed team engaged with ETL Solution Delivery teams, we realized that these teams were struggling with the basics of ETL test automation. We realized that our team couldn't possibly automate all tests in all the ETL projects. Instead we could pair with the ETL teams to write a few automated tests, build a test fixture and lay out processes for testing. Doing so would enable them to do their own test automation in the future and better equip them to build and scale their solutions in the agile world.

## 5 Benefits of ETL Test Automation

The benefits of automated testing are well known. Automated tests can be run easily and frequently identifying issues as soon as they happen. ETL test automation has the same benefits. The initial cost and effort to automate tests can be high, but over time and with an established framework, these tests can be implemented quickly and run cheaply. With these tests running in the background, the team can focus on more important issues like “how do we make our job more efficient?”, “how can we extend this to work for files from other vendors?” or “how can we simulate and test error conditions?”

Automated tests provide a safety net for the development process. With these tests in place, developers can edit existing components for enhancements and bug fixes with the confidence that any new issues they introduce will be caught. Conversely, without the automated tests, it is very difficult for developers to make changes to existing components. The fear of introducing issues which go unnoticed due to the lack of automated testing, is enough to deter anyone from making changes. Typically, this results in multiple copies of components, copies of components with subtle differences and components where defects never get fixed - a maintenance nightmare.

## 6 Enabling ETL Test Automation

At Cambia Health, the QuEST team works on building and improving processes. We focus on functional test automation and automated jobs to build, deploy and run these tests. These automated tests are one set of systematic Quality Assurance procedures that will help us deliver high quality products consistently on time and on budget.

The Solution Delivery teams, on the other hand, work on date driven projects to build new ETL jobs and enhance existing ones. On such date driven projects, the entire team is focused on building, testing and delivering the ETL functionality. It is difficult for such teams to switch gears and work on building better processes and improving their current practices. While these teams may realize that such improvements are very necessary for continued success, they often can’t prioritize them within their tight deadlines.

Our team fills this gap - we collaborate with the Solution Delivery teams to design and develop automated tests, set up processes to build, deploy and run these tests in a fully automated fashion and help them learn and adopt these process changes that can make them more productive. We do this by pairing with one or more members of the Solution Delivery team to implement the changes. For example, when writing an automated test to verify target data, the Solution Delivery team identifies the target objects and formulates the query. Our team creates the test project, plugs the query into the ETL test runner and sets up automated test runs. We do this together for the first few tests. Then, the Solution Delivery team is able to add and update tests themselves. In other words, we enable the Solution Delivery teams to build automated tests and improve their processes. These changes reflect positively in the quality of their product.

## 6.1 Best Practices Exchange

Any time significant process changes are introduced the biggest barrier is the resistance to change. We have seen this when transitioning from waterfall to agile development, when requiring unit tests for all new code written or when introducing the concept of using test cases as a substitute for detail requirements.  We expected to face similar resistance to ETL test automation. In our initial discussions with the Solution Delivery teams, we were bombarded with questions on complicated edge case test scenarios where test automation wouldn't work, challenges posed due to out-of-sync test environments and insufficient privileges to systems under test. These issues definitely needed to be addressed in order to be completely successful in running fully automated tests, but were irrelevant at the outset. Our initial discussions went almost nowhere due to teams that were unaware of the benefits of test automation or had never seen successful test automation.

At the time, our company had a Software Quality Assurance Best Practices Exchange (SQA BPE). The SQA BPE is a forum to discuss anything related to software quality, demonstrate tools and processes or just talk about issues that need to be fixed. However, it was a closed network of managers and some senior quality engineers. We changed this. We opened up the SQA BPE to everyone, set up weekly discussions and published a public calendar of events and topics. Now, anyone in our company could set up a discussion, post questions or present their work. We used this forum to discuss problems seen with ETL testing practices, suggest possible solutions and demonstrate a prototype test automation tool. The wide audience included quality engineers, system analysts, developers and a few managers who helped us with productive feedback. Later, we presented automated ETL tests that we built with some teams to demonstrate success.

Our discussions do not always attract good participation, go smoothly or end favorably, but we do have a forum. We get to pitch ideas, participate in discussions and drive change. We realize that driving change is not easy and that it can take years for us to implement some changes and see the benefits of some others, but we believe that the change to adopt ETL test automation is very essential to increase productivity.

## 6.2 Build a Better Tool

Before we embarked on building a framework for automated tests and a test runner, we weighed existing options. We looked at add-ons for the ETL development tool we used at Cambia Health. We considered other commercially available ETL test automation tools. These solutions were either prohibitively expensive or required the use of proprietary techniques to automate tests. We realized that following these paths might require teams to learn another new and specialized skill. Such efforts are usually not cheap and typically don't scale well beyond one particular tool. We also looked at existing test automation projects. Both these projects required the adoption of a new platform for test development work. In addition, they also needed QA people to be proficient in coding. While our QA team members were very strong in SQL, they lacked the ability to program in languages like Java and Python. These in-house solutions were limited to the two teams that had QA members proficient in these languages.

We realized that any new tool we introduced would have to satisfy the following requirements -

- Getting started is easy. Automated tests can be built using SQL.
- No disruption to existing processes.
- Operating system and database independent.
- Tests would be available for everyone to read and understand. This would help all team members to participate in test development.

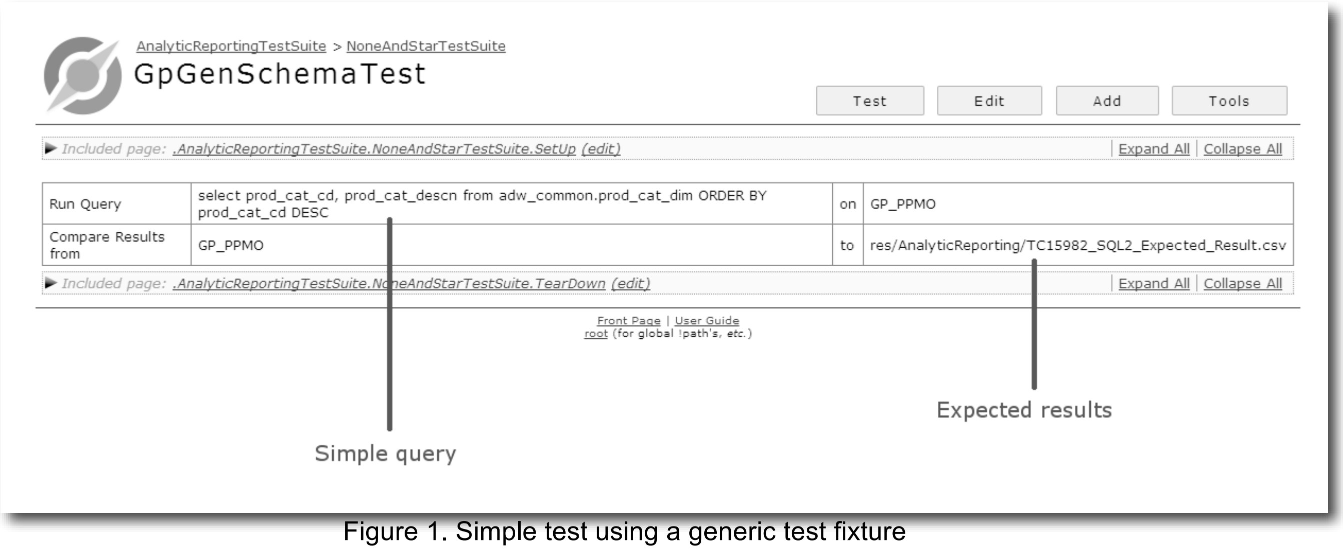
In addition to these, we recognized the need to build the tool in an incremental fashion. We did not have exhaustive requirements for this tool and we did not have the luxury to spend several months working on a perfect tool. We decided that building this tool incrementally was the best option. This method would allow us to try out features as soon as they are built, accept feedback from teams and grow the tool in an organic fashion.

We did not intend to build all the components of this new tool from scratch. Ideally, we would like to assemble a tool using a set of proven libraries. We wanted a mature platform and language that were widely used by development teams in our company. The advantage in picking a widely adopted language was that we could use help from other teams in our test development effort. We could also use all of the existing build tools, processes and infrastructure to build, deploy and run our tests. Since many development teams in our company use Java, this choice was easy. We'd use Java as the language to build test fixtures. These fixtures would have the ability to run SQL queries and interpret the results. The tests written by ETL testers would be in plain SQL which these Java test fixtures would execute. This way, ETL testers would never have to write or even look at the Java code. Java also has a good set of libraries for database access, which is really the most important feature of this tool.

The other major requirement was the ability to share tests with the entire team. Every member of the team would be able to read the tests, run them and interpret the results. This would let anyone on the team run tests at will. This also freed up the QA members to focus on more important tasks rather than running existing tests. We looked at a few test runners like Cucumber, Robot and FitNesse. All these test runners let us define the tests in plain text, which makes them readable. In the case of Cucumber and Robot, these tests are in text files in the file system. FitNesse, on the other hand, serves these tests as wiki pages. These tests can be easily accessed from anywhere using a browser. FitNesse tests can be run from anywhere by clicking buttons on the wiki page. FitNesse is a wiki and a test runner. Now everyone on the team can easily access the tests using a browser, read them and understand them. Once we decided on Java and FitNesse, the next step was to write a fixture class in Java.

### 6.2.1 Test Fixture Development

Our mantra here is, “Start simple, start small”. The reasoning behind this was that we had seen many test framework development efforts fail. These failed efforts were an attempt to build the perfect tool upfront and the end result was a tool with a clunky interface and proprietary syntax that no one wanted to learn. We kept the design simple and our code was simpler - just enough to make the first test work. Our first test connected to a test database, ran a query and compared the results with an expected result set (see Figure 1). We were able to whip up this functionality in a couple weeks and demonstrate it at the SQA BPE. By presenting this functionality, we wanted to let people know that they could try out simple tests against their databases by writing plain SQL. We also wanted to gather any feedback from them to continue adding functionality to the test fixtures.



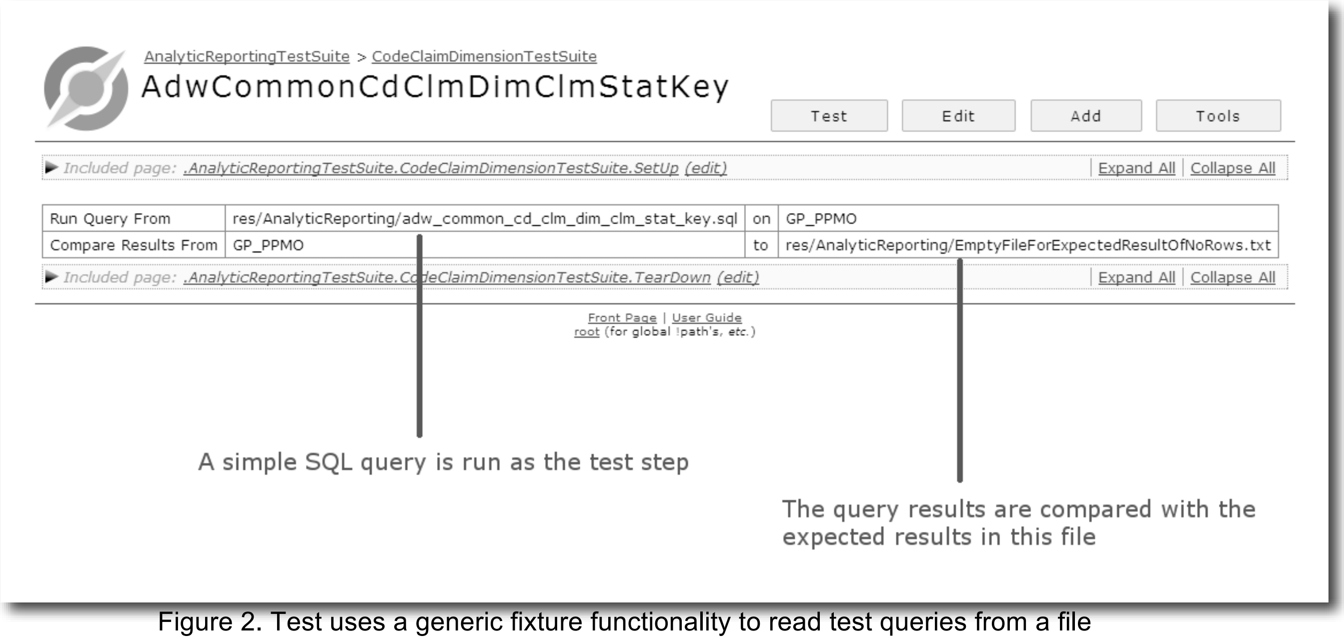
▶ Included page: .AnalyticReportingTestSuite.NoneAndStarTestSuite.SetUp (edit) | Expand All | Collapse All

| | | | |
|---|---|---|---|
| Run Query | select prod_cat_cd, prod_cat_descn from adw_common.prod_cat_dim ORDER BY prod_cat_cd DESC | on | GP_PPMO |
| Compare Results from | GP_PPMO | to | res/AnalyticReporting/TC15982_SQL2_Expected_Result.csv |

▶ Included page: .AnalyticReportingTestSuite.NoneAndStarTestSuite.TearDown (edit) | Expand All | Collapse All

Figure 1. Simple test using a generic test fixture

The most common criticism we got was that the tool and the testing concepts were too simple and would never work for real ETL jobs. But the simplicity was its strength - simple things work well, they grow and scale easily. A couple teams understood this concept. They wanted to know more about the tool, build a few tests themselves and understand how they could use it in their projects. We were not ready to hand over the tool to these teams. We decided instead to work with them to build their own tests. We paired with the QA members of the team to write the FitNesse wiki tests with the SQL queries in them. As the test queries got lengthy, we added a functionality in the test fixture to read queries from a file. So now, the wiki tests read lengthy, complex queries from files (see Figure 2). Logical file names for the query files also made it easy for other members on the team to understand the intent of the query.



AnalyticReportingTestSuite > CodeClaimDimensionTestSuite

AdwCommonCdClmDimClmStatKey

▶ Included page: .AnalyticReportingTestSuite.CodeClaimDimensionTestSuite.SetUp (edit)

| | | | |
|---|---|---|---|
| Run Query From | res/AnalyticReporting/adw_common_cd_clm_dim_clm_stat_key.sql | on | GP_PPMO |
| Compare Results From | GP_PPMO | to | res/AnalyticReporting/EmptyFileForExpectedResultOfNoRows.txt |

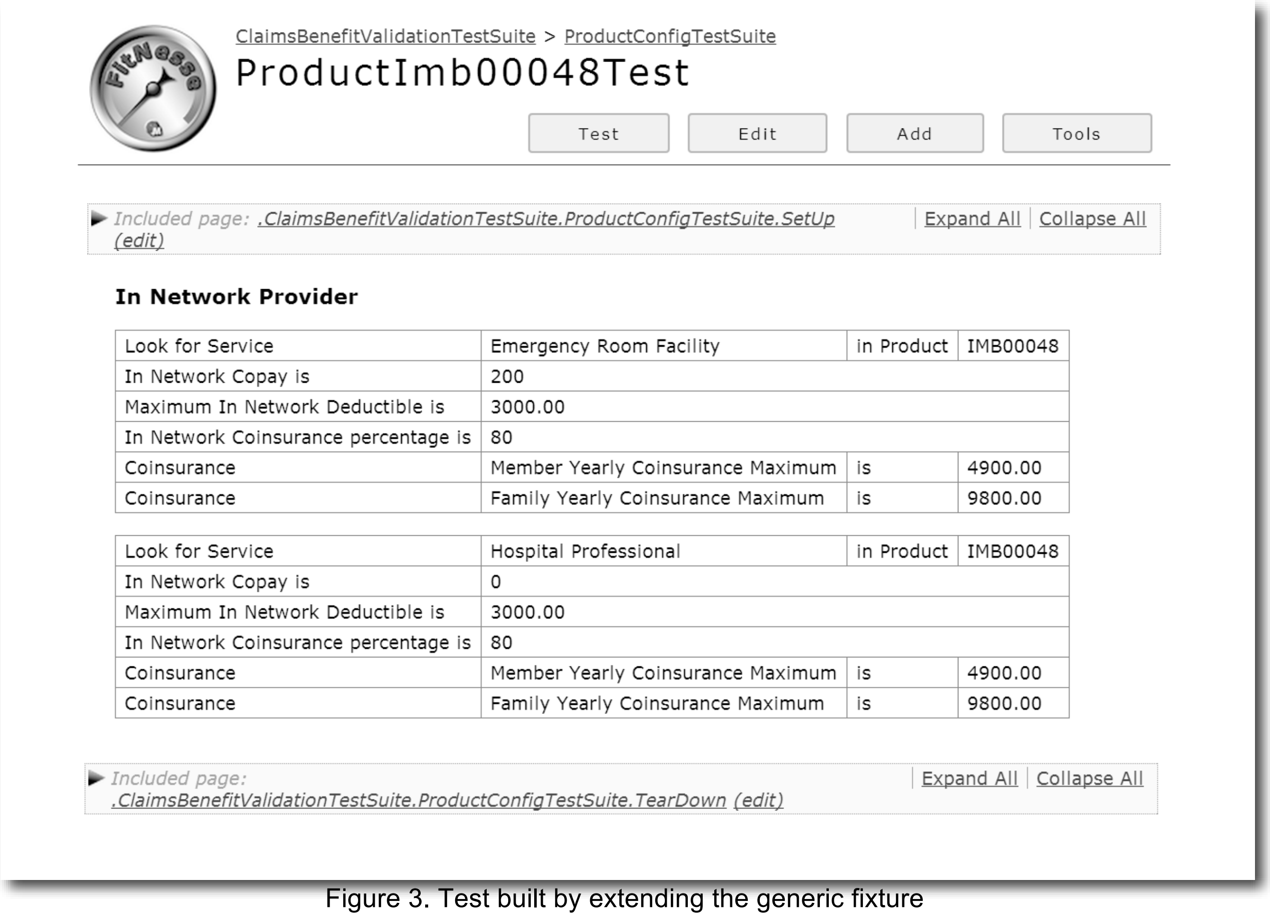
▶ Included page: .AnalyticReportingTestSuite.CodeClaimDimensionTestSuite.TearDown (edit)

A simple SQL query is run as the test step

The query results are compared with the expected results in this file

Figure 2. Test uses a generic fixture functionality to read test queries from a file

We have also recently developed tests with a product owner. These tests are more readable and express the functionality in plain English (see Figure 3). We were able to achieve this by extending the base functionality and adding a few wrapper methods to the fixture code.



ClaimsBenefitValidationTestSuite > ProductConfigTestSuite

ProductImb00048Test

▶ Included page: .ClaimsBenefitValidationTestSuite.ProductConfigTestSuite.SetUp (edit)

**In Network Provider**

| | | | |
|---|---|---|---|
| Look for Service | Emergency Room Facility | in Product | IMB00048 |
| In Network Copay is | 200 | | |
| Maximum In Network Deductible is | 3000.00 | | |
| In Network Coinsurance percentage is | 80 | | |
| Coinsurance | Member Yearly Coinsurance Maximum | is | 4900.00 |
| Coinsurance | Family Yearly Coinsurance Maximum | is | 9800.00 |

| | | | |
|---|---|---|---|
| Look for Service | Hospital Professional | in Product | IMB00048 |
| In Network Copay is | 0 | | |
| Maximum In Network Deductible is | 3000.00 | | |
| In Network Coinsurance percentage is | 80 | | |
| Coinsurance | Member Yearly Coinsurance Maximum | is | 4900.00 |
| Coinsurance | Family Yearly Coinsurance Maximum | is | 9800.00 |

▶ Included page: .ClaimsBenefitValidationTestSuite.ProductConfigTestSuite.TearDown (edit)

Figure 3. Test built by extending the generic fixture

More than a year later and with five teams using this ETL testing tool, the core functionality of the tool remains much the same - connect to a database, run a query and compare the query results with a set of expected results. We have added some utility methods and helper classes, but the core functionality remains the same. This shows that testing tools and frameworks can be built successfully starting with just enough functionality and adding more incrementally.

### 6.3 Educate

Educating teams about the benefits of test automation and the need to constantly improve existing processes is an ongoing task. So is learning the needs of the delivery teams to be able to build better tests. There are many opportunities to learn from each other, improve the tools we build and our skill set. So it is very important to engage with the teams and continue discussions.

Education can be done simply by sharing success stories at the SQA BPE or by holding elaborate training sessions. We are doing both. By sharing success stories, teams get to see and hear how others use the ETL testing tool in their projects. This acts like an endorsement and also helps people see similarities between their projects and advantages of using a common tool. At the time of writing this paper, our team had completed a two part training session on how to use the ETL testing tool.
We conducted workshop style training sessions on the following -

1. Introduction to Test Automation where we covered the basics of the FitNesse wiki, reading tests, running tests and editing them. Attendees worked on a sample test project we built for this workshop session.
2. Basics of ETL Test Automation - here we went over examples of tests that ran queries against a test database.

We are planning to continue delivering the same sessions a few times a year. We are also planning to deliver at least one more course on advanced ETL testing that will focus on extending the existing framework.

## 7 Some other Simple but Effective Techniques

Along the way, we discovered several techniques that are natural, but aren't really obvious until you do them. Simple things like informal discussions over coffee, celebrating successes, talking through pain points together go a long way in finding solutions and building collaborative relationships. Similarly, telling people they are wrong and pointing out flaws are not constructive at all. This seems obvious now, but we learned it the hard way.

## 8 Conclusion

Two years ago, the QuEST team started this focused effort to improve the quality of ETL products. We recognized that lack of automated testing and lack of skills to build automated tests as areas that needed significant improvements. Our work on building automated tests and a framework for testing have enabled teams to build automated tests and making these tests part of the development process have shown good results. Below are some preliminary results -

- Five out of the eight ETL teams have some automated tests. These tests are scheduled to run daily. They can also be run by anyone on the team by clicking a button. On one particular project that generates member benefits documents, testing consisted of end-to-end tests with visual inspection of documents. This was a time consuming process and defects were found very late in the development cycle. Now, automated ETL tests have been implemented to validate member data loads and document configurations. Issues are now found as soon as the ETL jobs have finished running and are quickly fixed. These automated tests have also drastically reduced the time and effort to run tests. Previously, the testing cycle took 4 to 6 weeks and followed development. Now, many tests are run in parallel with development and the final end-to-end tests can be completed in one week.

- Setting up automated tests is now easy and quick. Teams know how to get started and how to get help. Recently, a team was able to set up automated tests for almost 60% of the test cases even before the functionality was built. These tests caught defects, issues with the environments and identified gaps in requirements early.
- We have had a few enhancement requests to the ETL test runner from System Analysts. One such enhancement was to display the query run in the test. Looking at the query would give them more confidence in the tests. A few Product Owners are thrilled at the prospect of using FitNesse for test automation because they now understand exactly what the tests do. We now have non-QA team members participating in testing activities.

ETL test automation is real and can be done with simple tools and techniques. ETL test automation can be implemented in an agile fashion – one can start with simple tests and over time add tests to build a comprehensive test suite. ETL test automation, much like any other functional test automation, helps development by finding issues as soon as they happen and tightening the feedback loop. Test automation can be used to drive the development of ETL components. While it was a struggle to get early adoption from teams, we have seen good progress recently in teams accepting a common framework and embracing sound testing concepts. We believe that over time we'll continue to hone our test runner, improve tests and test coverage and these changes will result in a better product.

# Example Driven Architecture

**Moving beyond the fragile test problem once and for all**

**Gerard Meszaros**

**FeedXL Pty Ltd**

pnsq2014@gerardm.com

## Abstract

By using effective automated testing (A.K.A. checking) applications can be evolved more safely as developers add functionality. But the structures of most software systems make it hard to automate tests in a way that doesn't lead to fragile tests.

By preparing automated tests before building the functionality, we can force testability/checkability into the software architecture thereby making test automation both faster and less fragile. As a side benefit, it improves our software design by making it more modular (by making each part of the application testable in isolation).

Modular automated tests have worked well on projects where we have used this approach. It has led to less rework (because the automated tests describe what a successful outcome looks like) and less effort spent on automation. With legacy systems we have been able to drive the evolution of the architecture in a favorable direction by using automated tests for new (or even existing) functionality.

Example Driven Architecture is a people and process solution that can be applied to pretty well any system or application as long as we start early enough to influence the architecture.

## Biography

*Gerard Meszaros is an independent software development consultant and trainer with 30+ years experience in software and over a decade of experience in agile methods. He started doing eXtreme Programming in 2000 and quickly discovered that close attention to test craftsmanship was essential to keep the cost of change low. He described his key learnings in his book: "xUnit Test Patterns – Refactoring Test Code", which was published in May 2007 by Addison Wesley in the Martin Fowler Signature Series and it won a Jolt Productivity Award in the Best Technical Book category. Since then he has turned his attention to applying the same concepts to organizing the automated acceptance tests as executable examples. He is based near Calgary, Canada and has coached teams and taught courses as far afield as China, India and Europe. He is also the CTO of FeedXL.com which provides a web based diet optimization tool for horse nutrition.*

*Gerard has a BSC (Hons) in Computer Science from the University of Manitoba.*

# 1 Introduction

This paper describes our experiences driving development using executable examples and how this helped drive testability into the architecture of our system. We start by describing our motivation and the problems we typically encountered. This is followed by a description of the process of specifying behavior by using executable examples.

## 1.1 A Note on Terminology Used

While the use of the term “automated testing” is widespread in the testing community, there are those in the community who insist that this is actually “automated checking” since testing requires the presence of a “sentient being”. [Johnson 2011] But the term “check automation” is not widely in use, therefore I will continue to use “test automation”; if you prefer to call them automated checks, feel free to do a mental global change from “test” to “check” as this does not change the key message of this paper.

To distinguish a new style of automated tests that avoids many of the pitfalls of traditional test automation, I introduce the terms “example” and “executable example”. In the agile community, there is much debate about whether we should call these self-verifying examples “acceptance tests”, “specifications”, “executable examples” or something else. For the purpose of this paper, I tried to standardize on calling them “examples” but I hope you’ll forgive me if the odd “test” or “check” or “spec” slips in. The reframing of these automatable tests as examples and how that change in terminology changes everything is the essence of this paper. So I’ll be calling them “examples” and referring to the automation as “automated checking of executable examples”.

# 2 Motivation

Effective automated testing of the behavior of a software intensive system can simplify evolution of the software by making it safer to embark upon changes. Automated testing of the behavior can quickly and cheaply detect any regression bugs while new functionality is being added or bugs are being fixed.

# 3 Problems With Traditional Approaches to Test Automation

Anyone who has tried to implement automated testing (or “checking”) of a software-intensive system has probably encountered many issues, not the least of which is the “fragile test” problem. These issues result in tests that are expensive to build, slow to execute, tests that frequently break and that require frequent expensive fixing.

## 3.1 The Fragile Test Problem

Fragile tests occur because the structures of most software systems make it hard to automate tests in a way that doesn’t lead to fragile tests. The logic we want to verify is typically tightly coupled to the surrounding “plumbing” (database contents and access mechanism, run-time environment, interacting applications) and, because automated testing efforts typically start after the code has been built, it is very difficult to decouple the logic to facilitate automated testing.

Also, there are many dependencies that tests depend upon which may be outside the control of the automated tests. These include: dependencies on data in the database; dependencies in other systems; and dependencies on properties of the run-time container including permissions and the current system time and date. These are described in more detail in [Meszaros, Gerard. 2004].

## 3.2 Simply Automating Manual Tests Doesn’t Work

But it is not enough to just automate the tests or checks that human testers would execute manually. Manual testing is very labor intensive and some of the standard practices used to reduce execution cost do not work well for fully automated tests. For example, a manual tester may execute a series of tests where each test uses the leftover state from a previous test. If the previous test gives unexpected results, the tester can adjust their approach on the fly to repair the state of the system, execute a different follow-on test or abandon testing. An automated test cannot be expected to display this level of flexibility; to do so would make the tests/checks unreasonably complicated and even harder to maintain. Therefore, a different approach is required.

## 3.3 Automating Tests Through the User Interface is Problematic

Another problem with automating manual tests is that they typically interact with the user interface of the system and therefore result in very long, detailed test scripts when automated. This makes the automated test scripts very difficult to understand because they are dominated by the details of the interactions with the user interface elements (buttons, links, fields, etc.) and very prone to widespread breakage as every test that refers to a changed element in the user interface will break when that part of the UI is changed.

## 3.4 The Root Cause is the Software Development System

The true root cause of this problem is the way we organize our software development organizations: the people who build the system live over there and the people who test it live over here. And there is very little true collaboration between the two groups (or with the specifiers) during the actual development process.

Use cases are too abstract because they enumerate a range of possibilities. (See [Cockburn 1995].) Even use case scenarios are abstract enough to be completely misinterpreted. On the other hand, most automated test cases are too detailed to be easily understood; their primary audience is the computer software that executes them.

Software is inherently difficult to test automatically. This is because each distinct community builds what they think is required (specs, tests, software) and there is little to no cross checking between them as they work. So the three kinds of artifacts “drift” farther and farther apart. And the test automaters are stuck trying to hold everything together with the predictable result of fragile tests. Improving the technologies available to the test automaters may make them more efficient at bridging this gap but this is a band-aid solution that tries to cover up the underlying cause: the system we typically use to specify, build and verify software is dysfunctional.

# 4 Specifying Behavior Using Examples

Because the root cause is how we organize the system for building software systems, we need to change that organization to effect a meaningful change. We need to break down the (metaphorical or physical) walls between the specifiers, the builders and the testers, each with their own bosses and goals, to facilitate true collaboration. We need one team (definition: a group of people with a single common goal of specifying, building and testing the software intensive system). By preparing automated tests before building the functionality, we can force testability into the software architecture thereby making test automation both faster and less fragile. As a side benefit, automated tests improve our software design by making it more modular (by making each part of the application testable in isolation).

## 4.1 Single Cross-Functional Team

The specifiers need to specify the expected behavior of the system using potentially executable examples. They probably don’t have experience doing this, so they should collaborate with the testers in transforming vague statements of requirements into concrete examples that can be automated. But since most tests are too detailed to act as clear examples, the specifiers need to stay involved as the examples/tests are created and not just hand off the specs to the testers.

## 4.2 Using Examples to Specify Behavior

Concrete examples (that can later be automated as tests) need to be prepared before the corresponding production software is built. This has the following benefits:

- First, the developers have concrete examples of how the system should behave; they can use the real data in these examples to better understand the expected behavior.
- Second, developers are expected to build the necessary adapters to read the examples and use them to execute the relevant parts of the software and verify the correct results are obtained. This forces the developers to design the software to be testable. The examples define both the functional behavior and the testability of the software.

## 4.3 Managing Scope vs. Detail

Unfortunately, a naive or simplistic approach to automating the testing of a complex system results in very long, complex and detailed test scripts that are difficult to understand and maintain. Since the amount of information a person can hold in their head is widely recognized as 7+/-2 items [Miller, G. A. 1956], we need to strive to keep all examples short and to the point. This requires that we actively manage the level of detail in our examples with this limit in mind. Since we cannot describe all essential details in an example this short, we need to break our specification into different kinds of examples, some of which provide an overview of the functionality (the "big picture") and others that provide details about various parts of the logic (the "zoomed in" view).

The exact nature of the system being specified will influence how many kinds and levels of examples we require. A typical business information system will have at least three such levels:

1) One or more specifications of the overall workflow
2) Specifications for each major transaction type (e.g. for each use case)
3) Specifications for each business rule or algorithm

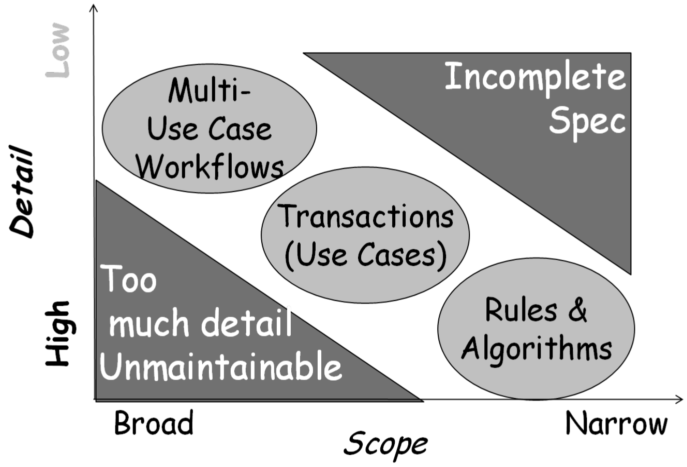


**Figure 1 Scope vs. Detail of Tests**

We often use a keyword or *action word* based approach [Buwalda et al, 2001, Zylberman et al, 2010] for the first two kinds of specifications but with different keywords for each level. Business rules and algorithms are frequently specified using tabular tests with each row describing an independent example.

The following sections expand and elaborate on this approach.

## 4.4 Workflow Specification

To provide an overview of how the system is used in the business, we need to provide some examples that illustrate the overall workflow of a business transaction as it flows from actor to actor. The actors may be humans (such as the originator of a request and each person who works on it) or machine (automated processing of workflow steps).

The workflow examples should have one step for each step of the workflow; that is, each actor involved in the workflow should have a single statement describing what they do. Any more than a single statement would result in workflow examples that exceed the 7+/-2 rule for any but the most trivial workflows.

To achieve the single statement per step goal, we need to abstract ruthlessly; we need to remove all non-essential details. To paraphrase Einstein, “If a detail is not essential to the understanding of the workflow, it is essential to exclude the detail.” Therefore, we define keywords for each step that take the fewest possible arguments.

When transitioning from manual test scripts or refactoring existing automated test scripts, we can remove the extraneous details by selecting multiple steps and replacing them with a summary of why they are done. Any data used in the script that is not material to differences between workflows can be “pushed down” into the underlying keyword definitions. By necessity, we must omit any references to user interface in these examples.

Even if an actor goes through multiple steps to execute a single transaction, we capture this as a single keyword with the end result of that series of interactions. For example, the user might use multiple pages to populate their shopping cart, enter their delivery and payment details and confirm the transaction. But for the purpose of defining the workflow illustrating how the purchase will be fulfilled, we would capture all this as a single keyword as in:

1. User makes a purchase and provides payment and delivery details
2. System checks payment info and reserves funds
3. System decrements stock on hand for selected items and produces stock list
4. Fulfillment staff assemble package based on stock list
5. Fulfillment staff mark order as complete
6. System transfers waybill to shipping company to prepare for pickup
7. System processes payment
8. System notifies user of order status

We don’t need to show any of these details in the examples unless what was purchased or how it was paid for affects the fulfillment process. For example, if some items are to be drop shipped from the manufacturer, we might say this as part of the relevant step keywords:

1. User makes a purchase of an item requiring drop shipping
2. System checks payment info and reserves funds
3. System sends drop shipping request to manufacturer
4. Etc.

We can hide the details of what makes an item “drop shippable” in the implementations of the keywords as that is not what we are specifying here.

## 4.5 Transaction (Use Case) Specification

When defining the expected behavior of a single transaction, we want to show the back and forth interactions between a single actor and the system in question. We want to focus on the essence of the conversation while avoiding the details of the mechanism. That is, we want to show the essential data being communicated but avoid the details of the user interface used to communicate it. We typically define one keyword for each action the user takes and another for each response the system provides.

For example, we might define the transaction behavior as:

1. User provides the name and details of the recipient
2. System constructs a preview of the item
3. User confirms the preview
4. System saves the item

Each of these list items is a single keyword. Note that in this example we do not provide any variable data but it is possible provide arguments to any of these keywords.

### 4.6 Business Rules Specification

The business rules and algorithms embedded in a software system tend to be complex and require extensive testing. The rules may be hard-coded or they may be defined through configuration (or "meta") data. There tend to be a lot of cases to be tested based on any one set of rules. Therefore, business rules are best described through common dependency configuration followed by a set of individual examples that can be verified one-by-one. The most common way of doing this is via a data driven approach wherein the same test execution logic is applied to many rows of data, each of which represents an example to be tested.

For example, we could define how a person's pay is calculated using the following Fit [FIT] table:

| HourlyPayDue | | | |
|---|---|---|---|
| Total Hours | Hourly Rate | Overtime Hours() | Total Pay Before Deduction() |
| 40 | 20 | 0 | 800 |
| 41 | 20 | 1 | 830 |

**Table 1 Simple Column Fixture used for Business Rules Example**

This table uses a FIT "ColumnFixture" [FIT] to enumerate a set of independent test scenarios or examples, one per row. This table contains two such scenarios: the first 40 hours are paid at the hourly rate and that any additional hours are considered "overtime hours" and are to be paid at time and a half. Note how we pick numbers at the boundary conditions to make it easy to see where the cutoff point is. The first row indicates the name of the "fixture" to be used to interpret this table: HourlyPayDue. This could be a Java class, a C# class or even a Ruby, Groovy or Python class, depending on which version of FIT we are using. FIT allows tables to be surrounded by arbitrary text that can be used to explain the logic or rules which the examples in the table illustrate.

# 5 Example Driven Architecture

After the specifiers and the testers (traditional role names) have prepared the examples, they are shown to the developers as part of the iteration planning process. Once the team has decided that a particular user story is to be built, the related examples are examined and the high level design of the software can commence.

### 5.1 Automating Execution of the Examples

The first thing to consider is how the examples can be used as automated tests. Ideally, the examples will not require any changes to do this beyond using a common vocabulary (itself good thing) and syntax; instead, we want to "connect" the examples to the appropriate part of the software being built. Since the example already exists, we know what it needs access to and this helps us define the high level design of the code to facilitate automated execution of the examples.

There are many open source and even some commercial tools available to facilitate connecting the examples to the system. (See [AAFTT 2009] and the section "Tools for Automating Execution of Examples" later in this paper.) The good ones allow us to define multiple levels of keywords to allow us to hide unnecessary details "in plain sight" within the keyword definitions. This keeps the examples short enough while still allowing the curious to see what data is actually being used without having to resort to reading code.

## 5.2 Automating Workflow Specifications

When automating the workflow specification, we define the tests in terms of keywords that describe what the actor is trying to achieve.

### 5.2.1 Defining Workflow Keywords

We push the non-essential (but technically necessary) details down into the keyword definitions. If several keywords definitions need to access the data, then we use “variables” to hold the data so we don’t have to repeat them.

Even if an actor goes through multiple steps to execute a single transaction, we capture this as a single keyword with the end result of that series of interactions. For example, the user might use multiple pages to populate their shopping cart, enter their delivery and payment details, and confirm the transaction. But for the purpose of defining the workflow illustrating how the purchase will be fulfilled, we would capture all this as a single keyword as in “user makes a purchase and provides payment and delivery details”. We don’t need to show any of these details in the examples unless what was purchased or how it was paid for affects the fulfillment process.

### 5.2.2 Implementing Workflow Keywords

The keywords are implemented by making calls into the software system passing the necessary data. This data may have come from within the example (as arguments of a keyword) or from within the keyword definition (if it is necessary but not essential to what we are describing in the specification). Ideally, we do not need to interact with the user interface when interacting with the system; instead we make calls to the same API that the user interface interacts with. If the user interface includes business logic, we may want to interact with the UI in some cases. Some of the “functional test” frameworks (e.g. Robot Framework) provide the means to use different keyword definitions in different test runs; we can use this capability to run the same examples either through the UI or bypassing it simply by providing two sets of keyword definitions.

## 5.3 Automating Transaction Specifications

The examples for each transaction tell us what information the user will exchange with the system. Therefore we must define a programming interface that corresponds to the requests the user makes and how the system responds. The good news is that the user interface often requires the same set of requests and types of data to function. If this is the case, we can define a common API used by the UI and the transaction specification keywords. The main job of the keyword implementations is to map the format provided by the automation framework to the specifics of the API. For our sample transaction example, these might look like this:

### 5.3.1 Keyword: user provides the name and details of the recipient

This keyword takes the name and account information provided as arguments, or generates appropriate values that are known to be acceptable. This avoids having to visit a number of examples should the rules for what is acceptable change in the future. It then submits the request to the system under test and stashes the returned result for use by the next keyword.

### 5.3.2 Keyword: system constructs a preview of the item

The keyword takes the stashed result from the first keyword and validates the contents based on what was submitted, whether provided as arguments by the example author or generated by the keyword. If the stashed result doesn’t match what the keyword expects, it marks this step as failed.

#### 5.3.3 Keyword: user confirms the preview

This keyword calls the API to confirm the information provided by the system. If the system is “stateless”, this will involve sending back the information provided by the system in the previous request as part of the confirmation API.

#### 5.3.4 Keyword: system saves the item

This keyword needs to verify that the system has indeed saved the item. This may be done by interacting directly with the system’s database, or preferably, by interacting with the API used to query the existing items in the system. The latter is preferable since it is less tightly coupled to the implementation in the system. If this results in very slow tests, it is acceptable to interact directly with the database inside a keyword implementation. This is more acceptable than it would be within the example steps since the keyword implementation is a single place that can easily be evolved as the database structure evolves.

### 5.4 Automating Business Rule Specifications

For business rule specifications, we typically have one row for each scenario we want to illustrate. We need to be able to invoke the software component that implements the business rule, passing it all the data in the row. The component’s API should return the result so that the test execution framework can compare it with the expected result provided in each row.

One testability consideration is where the data on which the example depends comes from. Most often this will be sitting in a database. The example needs to provide the depended-on data as part of its preconditions (prerequisites), so that the example is not dependent on pre-existing data which when changed would result in failing tests. It is less than optimal to have to load the prerequisite data into a database each time the tests are run. A better solution is to architect the component such that it is passed either the prerequisite data or a “data provider” when it is constructed. When used from a test, the data can be passed directly to the component as a “test stub” [Meszaros 2007] containing all the data. This avoids any database access and makes it possible to execute hundreds of examples per second rather than many seconds per example. This is a good example of how “Example Driven Architecture” improves the testability of the design.

## 6 Tools for Automating Execution of Examples

There are many tools available for automating the execution of the examples.

### 6.1 Criteria for Tools Selection

The main criteria for selection of a tool for expressing and automating examples are two-fold:

- The ability to express the requirements clearly in the language of the domain experts.
- The ability to connect the examples to the underlying implementation logic as easily as possible.

Most commercial functional test automation tools were built without these as specific goals. Any attempt to use such tools to implement Example Driven Development is doomed to failure.

### 6.2 Sample tools

The tools listed here are merely examples of tools that have been used successfully with this process. As tools can and do change regularly, the goal of providing these tools as examples is to help the reader understand how to evaluate tools (“teach a man to fish”) rather than to provide a list of tools to be used (“give a man a fish”). The Functional Test Tools project of the Agile Alliance maintains a spreadsheet of testing related tools intended for use on agile projects [AAFTT 2009].

### 6.2.1 FIT/Fitnesse

FIT (the Framework for Integrated Test) [Mugridge R, 2005] uses tables to represent examples. Fitnesse is a wiki based version of FIT. Each table names a “fixture” class that contains the code used to interpret the table. Implementations of FIT are available for many popular languages. Most include fixture styles that support all three styles of examples described here. The ColumnFixture is the easiest way to implement a business rules test. ActionFixtures and FlowFixtures can be used for keyword driven workflow tests in FIT but they are a bit more complex to create than Robot Framework keywords.

### 6.2.2 Robot Framework

Robot Framework [Robot Framework] (A.K.A. RF or just “Robot”) can be used to express examples in a keyword driven style. RF Tests can be parameterized with a list of examples to use the data driven style although this is not quite as clean as Fit’s ColumnFixture. Keyword definitions can be stored in libraries, which can be selected at runtime. Keywords may be defined in terms of other keywords or in the underlying programming language. RF is implemented in Python, but can also run on Jython or IronPython allowing keywords to be implemented in JVM languages or .net languages. RF includes a large collection of connectors enabling RF examples to interact with the system under test directly via API calls, via the web browser or many other ways.

### 6.2.3 Cucumber

Cucumber [Cucumber] is a Behavior Driven Development tool that implements the Given-When-Then style of examples. The specification language is called “Gherkin” and is also implemented by a number of other tools.

# 7 Sample Specifications

To help illustrate the difference between a traditional, naive approach to test automation and the use of well designed examples at multiple levels, we have a fully worked set of examples to describe how a bank might allow its customers to configure thresholds for when they want to be notified of transactions on their bank and credit card accounts.

## 7.1 Overview of Functionality

The bank wants its customers to be able to configure thresholds for when they want to be notified of transactions on their bank and credit card accounts.

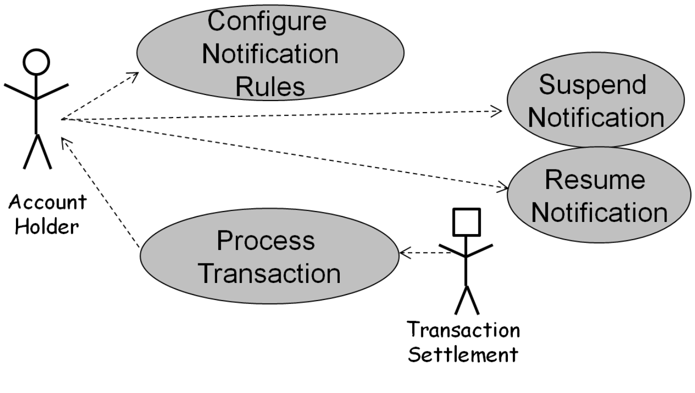


**Figure 2 Use Cases of Sample Application**

The account holder can initiate one of three different use case transactions. *Configure Notification Rules*, *Suspend Notification* and *Resume Notification*. The Transaction Settlement system initiates the fourth use case, *Process Transaction*, which results in a notification being delivered to the Account Holder.

The business rules of interest are configured in the *Configure Notification Rules* use case and exercised in the *Process Transaction* use case. Therefore minimal workflow we could use to verify the proper implementation of the rules consists of having an Account Holder *Configure Notification Rules* and then having the Transaction Settlement System *Process Transactions*. A more complex workflow would also include the *Suspend Notification* and *Resume Notification* use cases.

## 7.2 A Traditional Automated Test

The following is an example for the scenario “Configure and Test a Simple Notification on a Single Account” written in Fitnesse using a traditional test automation style.

Because it is written in Fitnesse, it already abstracts away from the user interface, focusing on the user’s actions and the system’s responses. In Fitnesse, vertical bars (|) are used to delimit table cells and each group of lines separated by a blank line represents one table. Note how this example has imposed the testability requirement of being able to advance the system time, something that would not typically be provided but which is essential to effective testing of such systems.

```
| Customer    | bobma    | logs in    |

| System lists all available accounts for the authorized customer   |
|     account          |     type          | notifications               |
| 10035692877          | chequing          | disabled                    |
| 10035692890          | savings           | disabled                    |
| 20010928892          | credit line       | disabled                    |

| Customer sets notification threshold for | all | transactions from | all |locations to |
    $10,000.00    | on account    | 10035692877 | via | email | to |!-bobma@live.com-!  |

| ensure    | No system messages     |

| ensure    | System log contains | "Customer bobma set notification threshold for all
    transactions from all locations to $10,000 on account 10035692877"|

| System lists all available accounts for the authorized customer   |
|   account        |     type        | notifications                  |
| 10035692877      | chequing        | enabled                        |
| 10035692890      | savings         | disabled                       |
| 20010928892      | credit line     | disabled                       |

| Notification settings for account                  | 10035692877                              |
| transaction type | location where initiated  | threshold amount| via    |    address          |
|        all       |              all          | $10,000.00      | email | !-bobma@live.com-!|

| Time now is | 9:30AM, 03/18/2008    |

| Bank processes  | debit      | to| 10035692877    | in the amount of| $15,000.00   |
| Bank processes  | debit      | to| 10035692877    | in the amount of|   $9,000.00  |
| Bank processes  | debit      | to| 10035692877    | in the amount of| $11,000.00   |
| Bank processes  | debit      | to| 20010928892    | in the amount of| $12,000.00   |
| Bank processes  | credit     | to| 10035692877    | in the amount of| $13,000.00   |
| Bank processes  | credit     | to| 10035692877    | in the amount of|   $9,999.99  |
| Bank processes  | charge     | to| 10035692877    | in the amount of|   $9,999.99  |
| Bank processes  | charge     | to| 10035692877    | in the amount of| $11,000.00   |

| New notifications sent to customer               | bobma                                     |
| type   |       account     |   timestamp          | amount        | via    |     address       |
| debit  | 10035692877| 9:30AM, 03/18/2012| $15,000.00   | email | !-bobma@live.com-!|
| debit  | 10035692877| 9:30AM, 03/18/2012| $11,000.00   | email | !-bobma@live.com-!|
| credit | 10035692877| 9:30AM, 03/18/2012| $13,000.00   | email | !-bobma@live.com-!|
| charge| 10035692877| 9:30AM, 03/18/2012| $11,000.00   | email | !-bobma@live.com-!|
```

**Table 2 Overly Detailed Workflow Test for Threshold Based Transaction Notification**

This test uses a style of Fit fixture called an Action Fixture. Blank lines separate tables with different functions. Some tables consist of a table type line (e.g. New notifications sent to customer | bobma) followed by a row of column headings followed by a series of table rows. Other tables consist of a single line starting with a verb plus some arguments. Other test automation frameworks may use a different syntax but the intent is the same: to describe the behavior expected in a particular scenario using human readable text.

The problem with this test is that it contains too much detail for the scope (workflow) thereby making it too long and hard to understand. It takes conscious effort to match the notifications to the transactions so that we understand which ones should be notified. And the excess detail is not helping us understand the overall workflow.

Because this is the simplest workflow in the system, we need to find ways to simplify the example so that the more complex workflows won't be even more complex and verbose.

## 7.3 Changing Scope or Detail

The workflow test contains lots of useful information but too much for the workflow example. Much of it will be useful for the transaction example focused on configuration. But for the workflow example, we want to reduce the level of detail. And for testing the business rules, we need a more compact and easily traceable format of example.

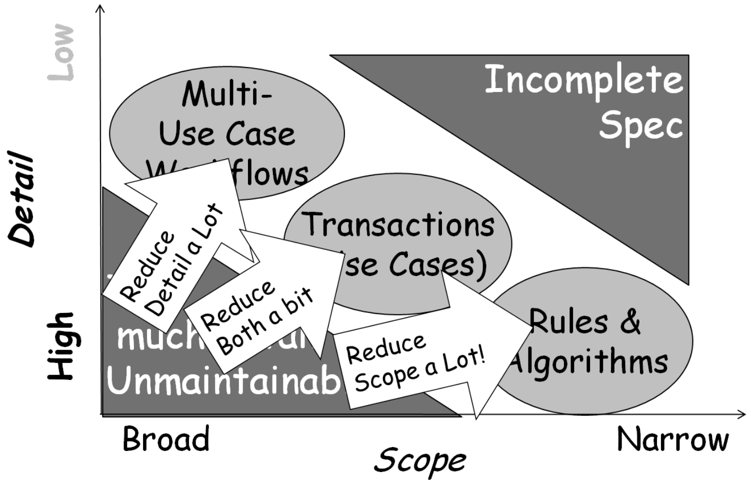


Figure 3 Reducing Detail, Scope or Both

Given that we have too much detail, we are down in the bottom left corner of the diagram and we either need to move up (less detail in a workflow example), to the right (reduce the scope to just the rules component) or both (focused on a single scenario of a single use case).

## 7.4 Well Factored Workflow Specification

The overall workflow needs to focus on what each actor does and what the expected outputs of the system are.

```
| at some time     |
| customer sets notification threshold to | $10,000 | for all transaction on bobs-checking |

| at a   | later  | time   |
| Bank processes  | debit  | to| bobs-checking    | in the amount of | $10,000  |
| Bank processes  | debit  | to| bobs-checking    | in the amount of |   $9,999  |
| Bank processes  | credit| to| bobs-checking    | in the amount of | $10,000  |

| new notifications sent to customer      | bobma          |
| type        | account              | timestamp    | amount   |
| debit       | bobs-checking   | later            | $10,000  |
| credit      | bobs-checking   | later            | $10,000  |
```

**Table 3 Minimal Workflow Specification Example for Threshold Based Transaction Notification**

This example contains much less information but it still conveys the overall workflow. Some irrelevant details have been omitted entirely while others have been replaced by meaningful placeholders like "bobs-checking". We could include more sample transactions but we can do a much better job of illustrating the notification decision logic in the examples for the notification business rule.

## 7.5 Well Factored Use Case Specification

We can take the naïve workflow test and use the first half of it to describe how the user interacts with the configuration interface to set up the notifications.

```
| Customer     | bobma     | logs in     |

| System lists all available accounts for the authorized customer    |
|       account          |       type               | notifications                        |
| 10035692877    | chequing             | disabled                               |
| 10035692890    | savings               | disabled                               |
| 20010928892    | credit line          | disabled                               |

| Customer sets notification threshold for | all | transactions from | all      | locations to   |
     $10,000.00 | on account | 10035692877     | via | email | to | !-bobma@live.com-! |

| ensure    | No system messages      |

| ensure    | System log contains | "Customer bobma set notification threshold for all
     transactions from all locations to $10,000 on account 10035692877" |

| System lists all available accounts for the authorized customer    |
|    account          |       type          | notifications                                |
| 10035692877    | chequing        | enabled                                       |
| 10035692890    | savings          | disabled                                      |
| 20010928892    | credit line     | disabled                                      |
```

**Table 4 Minimal Use Case Specification Example for Configure Notification Threshold**

Here we have included more detail as we are watching for the change in state on the chequing account and we want to make sure all the information that would be shown to the user on the screen is correct.

## 7.6 Well Factored Business Rules Specification

Now that we have show how the notification rules are configured, we can illustrate how the notification decision is made. First we need to describe the rules via the CustomerAccounts and CustomerThresholds FIT tables. Then we ask the question "Is Notification Required?" for each of a representative set of example transactions carefully chosen to illustrate how the notification logic should work.

| CustomerAccounts | | | |
|---|---|---|---|
| Customer | Account | Label | Added() |
| bobma | 100373 | Checking | ok |

| CustomerThresholds | | | | |
|---|---|---|---|---|
| Customer | Account | Charge Type | Threshold | Added() |
| bobma | 100372 | All | $10,000 | OK |
| bobma | 100372 | Travel | $1,000 | OK |
| bobma | 100372 | Restaurant | $100 | OK |
| bobma | 100372 | Groceries | $264.23 | OK |

| NotificationRequired | | | |
|---|---|---|---|
| Account | Charge Type | Amount | Notify? |
| bobma | Travel | 999.99 | No |
| bobma | Travel | 1,000.00 | Yes |
| bobma | Restaurant | 99.99 | No |
| bobma | Restaurant | 100.00 | Yes |
| bobma | Groceries | 264.22 | No |
| bobma | Restaurant | 264.23 | Yes |

**Table 5 Minimal Business Rules Specification Example for Notification Threshold by Charge Type**

Each of the rows in the final table describes one transaction to be processed. The first 3 columns represent the key data in the transaction. The final column expresses whether this should result in a notification. When executed, the final column's contents will be compared with what the notification component returns and the cells will be color coded green (for pass) or red (for fail). When the row fails, both the expected and the actual results are shown to help us understand what went wrong.

## 7.7 Example Driven Architecture for Sample *Examples*

In this section we'll describe how the examples in the previous section are automated and how their presence influences the architecture of our system.

### 7.7.1 Workflow Steps API

The workflow example requires easy access to an API that corresponds to each step in the example. Providing such an API (as opposed to just the user interface) makes automating the example checking nearly trivial. Since automation of the examples is the developer's job, they are incentivized to provide the API, since the total work to implement the API plus the automation will be less than the work to implement the automation without the API. There should be a strong synergy between automation of the examples and development of the user interface as the API provided for the examples should be a subset of that needed by the UI.

### 7.7.2 Transaction Steps API

The keyword definitions for the transaction examples should be able to use the same APIs as the user interface. But if this is not the case, the presence of the examples and the need to automate their checking should motivate the developer to refactor the APIs to meet the needs of both the keywords and the UI.

### 7.7.3 Notification Rules Component

The rules example will motivate the development team to define a NotificationRequired component with a simple API to ask the question "is notification required for a user transaction given the user's notification rules?" In code this would translate to:

```
rulesComponent.isNotificationRequiredFor(aUserTransaction,theUsersNotificationRules);
```

By having the need to automate the example checking, we must make the logic accessible. By passing in the rules with each request, the component becomes stateless making it much easier to test. It is the presence of the examples that drives this componentization of the architecture and the use of “data injection” into the component instead of accessing the rules directly from a database.

# 8 Results

This testing process outlined in this paper has worked well on new development projects where we have used this approach. It has led to less rework (because the automated tests describe what a successful outcome looks like) and less effort spent on automation. In *Specification by Example*, [Adzic] presents a number of case studies where this process has resulted in good business results.

It is essential to implement all aspects of the automated testing approach, as “cherry picking” of practices can result in less than optimal results. For example, in one case, a business team prepared extensive examples for use by an outsourced development team. But because the outsourcer had their own development process, they chose not to automate the examples and therefore much of the benefit (automated testing, improved testability) was lost.

## 8.1 Applicability to Legacy Systems

When dealing with legacy systems, we can slowly improve the testability of the architecture by first using keywords that interact with the system through the UI. This makes it safer to refactor parts of the system to make it more testable. Once that is achieved, the relevant keywords can be reimplemented via API calls without having to change the examples themselves because the difference is hidden in the keyword definitions.

# 9 Conclusions

The use of concrete, automatable examples to drive development is a people and process solution that can be applied to pretty well any system or application as long as we start early enough to influence the architecture. The examples not only help the developers understand what needs to be built, they also help the specifiers clarify their own thinking about what they want built. Furthermore, having the automatable examples on hand while they are building the system encourages the developers to make the architecture amenable to automated execution of the examples as this reduces the effort and uncertainty involved in developer testing of the software.

It is essential to make both process and organizational changes to be successful in this approach. It is the shared responsibility that motivates the various parties to play their part in making test automation simple to construct and maintain. If specifiers and testers are not required to provide the examples before development starts, there is no incentive for developers to define a test friendly architecture. If the developers are not responsible for implementing automated checking of examples, they are unlikely to provide the necessary APIs to make the automation easy to implement.

# 10 Key Terminology

**Use Case:** An abstract description of the various ways a user may achieve a particular goal while interacting with a particular system or application. By definition the goal must be achievable in a single sitting. A use case may be part of a larger workflow or business process.

**Use Case Scenario:** An abstract description of one way a user may achieve a particular goal.

**Work Flow:** A series of interactions between users and systems as part of a business process. Each user would be executing one use case scenario, likely from different use cases.

**Feature:** A set of functionality that will deliver value to some stakeholder. Often built incrementally as a series of user stories. May involve one or more use cases. May add scenarios to previously defined use cases.

**User Story:** The smallest unit of verifiable functionality; used as the increment of software delivery planning on agile projects. Typically involves implementing one scenario each of several use cases.

**Test Scenario:** An executable sequence of steps to verify that a system implements one or more use case scenarios.

**Example:** A concrete example that illustrates the behavior the system is expected to exhibit. It may illustrate the behavior describe in one or more use case scenarios. An example can be used as a test scenario, but not all test scenarios make good examples. Unlike a use case scenario, it is concrete, complete with the necessary data. Unlike a test scenario, its primary focus is communication of the desired behavior rather than verification of it. As a result, examples tend to be much simpler with most non-essential details encapsulated behind the fixture.

**Executable Example:** An example that can be checked automatically by running it via an example execution tool.

**Fixture:** The code and infrastructure required to enable an example (or test) to be executed. Not really part of the example, much like the store shelving fixture supports the merchandise but is not for sale.

# 11 References

AAFTT 2009, *Agile Automated Testing Tools Spreadsheet* https://docs.google.com/spreadsheet/ccc?key=0Apag_J97I3CTdHR0WS1sZGFVaFA0dEpXYWRqLXBxV3c&usp=drive_web#gid=1

Adzic, Gojko, *Specification by Example: How Successful Teams Deliver the Right Software.* Manning Publications

Buwalda, Hans. Undated. *Key Success Factors for Keyword Driven Testing* http://www.logigear.com/resources/articles-presentations-templates/389--key-success-factors-for-keyword-driven-testing.html

Buwalda, Hans, Dennis Janssen, Iris Pinkster. 2001. *Integrated Test Design and Automation: Using the Test Frame Method,* Addison-Wesley Professional

Cockburn, Alistair. 2001. Writing Effective Use Cases. Addison-Wesley Pearson Education

Cucumber, *Behavior driven development with elegance and joy.* http://cukes.info/

FIT, *The Framework for Integrated Tests*, http://fit.c2.com/ and http://en.wikipedia.org/wiki/Framework_for_integrated_test

Fitnesse, *The fully integrated standalone wiki and acceptance testing framework*, http://www.fitnesse.org

Johnson, Dion, *Checking Vs. Testing Is Hot! Cyber-Dueling Over 'Check' Vs. 'Test' and Other Semantics,* Automated Software Testing Magazine. June 2011

*Keyword-driven testing*, http://en.wikipedia.org/wiki/Keyword-driven_testing

Zylberman, Ayal and Aviram Shotte, *Test Language -Introduction to Keyword Driven Testing*, Summer 2010 issue of Methods & Tools http://www.methodsandtools.com/archive/archive.php?id=108

Meszaros, Gerard. 2004, *The Fragile Test Problem*. Proceedings of Agile United 2004 conference.

Meszaros, Gerard. 2007, *xUnit Test Patterns – Refactoring Test Code*, Addison Wesley Professional.

Miller, G. A. 1956. *The magical number seven, plus or minus two: Some limits on our capacity for processing information*. Psychological Review 63 (2): 81–97. Also: http://en.wikipedia.org/wiki/The_Magical_Number_Seven,_Plus_or_Minus_Two

Mugridge, Rick & Ward Cunningham 2005 *Fit for Developing Software - Framework for integrated Tests*, Pearson Education.

Robot Framework. *Generic test automation framework for acceptance testing and ATDD,* http://robotframework.org/

# Five Symptoms Your Test Automation Is Dying

**An Doan**

an.doan@outlook.com

## Abstract

Are you thinking about starting an automation project? Are you chest deep in test automation? You might not know it yet, but your automation project may be on a collision course with an undodgeable Hadouken Fireball. Certain telltale symptoms can help identify problems early. This paper discusses the key symptoms, what causes them, why they are terminal, and what you can do to diagnose and treat them. These anti-patterns, if left unattended, will degrade the quality and efficiency of your test automation, and may cause hardship and eventual failure of the automation project. As with all illnesses, having a symptom or two does not mean imminent failure. Awareness of the symptoms, however, including how to identify and treat them, can help avoid automation pitfalls.

In the context of a software test automation project, this paper looks at five key symptoms that may be evident, and could cause automation projects to fail.

## Biography

*Developing test automation by day, speaking and organizing automation meet-ups by night, An Doan is an 8th year "Dark Developer" specializing in writing dark code, code that breaks other code. He develops test automation frameworks and quality processes. As co-founder, and idea incubator of the Portland Selenium and Test Automation Users Group he helps the community share and develop creative ideas in the realm of testing and test automation.*

# 1 Introduction

Testing early and testing often is the key to delivering a high quality piece of software. That makes automation the most important testing technique in allowing QA to test early and often. Unfortunately, it is also the most difficult get right, and often fails to be of any true value. Think about the following questions:

- Does the team support test automation? The company?
- Do you have resources at your disposal to procure automation software, hardware, and people?
- Are you trying to train manual testers?
- Are you trying to convert your manual test cases into automated test cases?
- Does your primary concern revolve around making automation business readable?
- Are you using automation to fulfill a business need or solve a business problem?
- Are you dependent on record and playback tools?
- Does your automation framework seem clunky?
- No automation guru on your team?

If you answered 'yes' to more than a few of these questions, your test automation may be dying. In this paper, you will learn about the five symptoms that result in test automation failure, how to identify these automation anti-patterns, and how to treat them.

## 1.1 Test Automation

Test Automation is the art of using software to test software. Anywhere where there is any kind of test against a product, it has the potential to be automated. A great example of upper limits of automation is the Tapster Mobile Automation Robot made by Selenium contributor and Sauce Labs co-founder Jason Huggins. It automates the interaction of a human finger with a mobile touchscreen. The task is daunting, but the rewards are well worth it.

# 2 The Symptoms

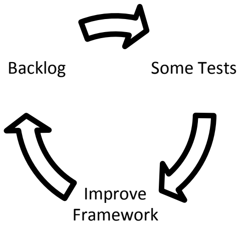
## 2.1 The Missing Project

You have been brought into an Agile team working on a new mobile application. The manager wants to do automated testing, but it is not captured anywhere in the Agile backlog and workflow. No one is bothered by how it happens or what the results look like, they just want test automation. There are other teams within the company that might be able to help but they all seem to be doing the similar things, solving similar automation problems, and reinventing the same wheel repeatedly.

### 2.1.1 The Symptom

Creating great software requires planning and organization. Likewise, test automation is software development, and as such requires the same level of planning and organization. Test automation is complex, sometimes difficult and requires careful planning and precise execution.

You have this symptom if your automation framework is not designed, developed, and maintained as a project.



For small teams test automation should be a part of the main development project. For an environment with multiple development teams test automation should be its own project with its own team. Its mission would be to create automation and automation tools to support the other teams.

### 2.1.2 The Cure

Treat this symptom by finding a leader, treating automation as an Agile project, taking small steps, and making iterative improvements.

#### 2.1.2.1 Leader

First, you need to find a leader. This person need not be an automation guru. However, this person needs to be able to solve the problems of today, while anticipating the solution to tomorrow's problems. Solving the problems will involve learning new tools, seeking out new resources, as well as new technologies. In order to track and solve problems effectively, we will borrow some proven practices from the Agile world.

#### 2.1.2.2 Project backlog

The second step in getting your automation organized is to create an automation backlog. You should use your development team's backlog if you can, or use a separate backlog. Getting your automation visibility within the team is important but do not let it get in the way at this point. What is important right now is to get tasks down on paper.

In Agile, an Epic "captures a large body of work. It is essentially a large user story that can be broken down into a number of smaller stories." (Atlassian), and a User Story "captures what a user does or needs to do as part of his or her job function" (Wikipedia). In this situation the user is the team's automation engineer, the person or persons responsible for writing automated tests.

The following Agile Epics should be written, prioritized, broken down into smaller Stories, and solved one by one. As these Epics are interrelated by the automation project as a whole, remember to think about each Epic and how they would be solved in relation to each other. For example: sharing test results is related to seeing test results, the audience to which the reports are shared influences the format in which they are presented.

- "I want to write an automated script"

  This Epic is about choosing a test tool. Who will write automation? What tool or programming language will be used? How will the automation be written; i.e. what framework and design patterns will be used? When will the tests be written? Before, during, or after the application's development?

- "I want to see the test result"

  This Epic is about test reporting. "If a tree falls in the forest, and no one hears it, did it fall?" How will the team report automation progress and success? How does the automation report failure? When it fails, it should be easy to determine the failure. Who will read the reports?

- "I want the script to run on a remote machine"

  This Epic is about how the tests run. The automation does not give much value if it takes over the user's computer and the person has to sit there and watch it run. If that's the case you might as will test manually. The automation should run on another machine, so that the user's time is freed up for more development, complex testing, or exploratory testing. The chosen test toolset should have the ability to run on a remote machine. It may come with this feature out of the box, or be added through customization.

- "I want to run the automated script automatically"

  This Epic is about the mystical unicorn that is Continuous Integration (C.I.) and Continuous Deployment (C.D.). A few have it, we all want it, but we are not too sure if it exists or how to get there. Having automation, and having the ability to run it automatically and repeatedly is a key step in achieving C.I. and C.D.

- "I want to share the result"

  This Epic is about publicizing the test reports. What good is our success if we do not celebrate it loudly? Are the test results sent out in email? Are failures automatically entered as bugs? Is there a big dashboard on the wall that turns from green to red? If a developer causes a test to fail, which flavor of doughnuts do they need to buy?

#### 2.1.2.3 Take Small Steps

Now that the automation road map is laid out, plan to accomplish these stories in small steps. Start with the 5 or 10 most important smoke tests. Automate those, get a simple report out of it, get it running automatically, and share your success. Think simple, but expect continual improvements. The tools used in the beginning might not be the correct tools for the job, so be open to change and improvement.

#### 2.1.2.4 Iterative Improvements

The last step is unfortunately not the last step but an "advance to go, collect $200." An Automation Project is continual. Just like the way your application is continually improved, and your team continually grows, your automation must improve and grow. When new features are added to the application it may be necessary to add or change the tool or framework used for automation. Upgrade your toolset continually, leave your good tests in the old framework or tool, and as you add automation for the new features implement them using the new framework or tool. Eventually as your application is improved, so will your automation. More tests may mean more test hardware. Longer reports may mean reporting improvements. The automation should have the ability to improve as needed.

## 2.2 Training Manual Testers

The company has just started a new training program aimed at improving testing and thus improving the quality of the products delivered. A new automation title is created and any manual tester who wants this new title needs to attend programming and automation training. The training program was well received, and initial impressions were positive. Tests were automated left and right and all seemed well. Almost every team had at least one automated test. As the easy tests were completed the newly trained automated testers struggled to automate the more complex test cases. The automation had grown to the point where it was beyond of the scope of the original training. Only a few truly knew what they were doing, and the rest followed like lost sheep.

### 2.2.1 The Symptom

Another symptom of a dying automation project is the prospect that a manual test team can be turned into an automation team through training. Borrowing a line from Liam Neeson's character in the film "Taken"; automation engineers have a "particular set of skills; skills … acquired over a very long career." This does not mean that a manual tester will not ever become an automation engineer. It means that not everyone will succeed. On top of that, it will take experience to be successful. There will be a low success rate when training manual testers.

#### 2.2.2 The Cure

A manual tester trained in a dozen automation examples will most likely only be able to write tests similar to those dozen examples. A select few will have the particular skills, to become an automation engineer.

- Programming background – Automation engineers should have a background in various forms of programming and programming languages. Learn enough recipes until you can create your own.

- Creative Nature – Automation engineers need to be creative. It requires a bit of problem solving and a bit of out of the box thinking. This comes with experience.

- Inquisitive – Automation engineers are inquisitive by nature. They are always learning, always seeking out new technologies, attending conferences and meetups; and learning from others.

There are two types of training, externally motivated and internally motivated. The first one creates manual testers that try to automate, and the later creates automation engineers.

##### 2.2.2.1 Externally Motivated

Externally motivated training, is training that comes from an ulterior agenda. It is something the company wants, not necessarily what the tester wants. It is the company realizing too late that exclusive manual testing is costing them too much. It is that one requirement that will get them the next career bump.

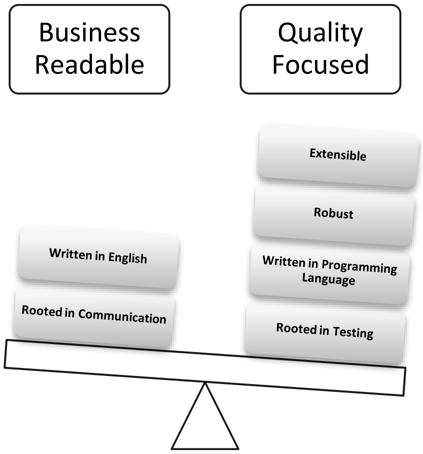
##### 2.2.2.2 Internally Motivated

Internally motivated training comes from the tester. It is born from the desire to test, the desire to test better and faster. This desire will eventually lead the tester to automation as a superior supplement to manual testing. The tester will seek out books, classes, meetups, and conferences.

### 2.3 Business Readable Trap

The QA director has just recommended the use of BDD and Cucumber. All development teams are expected to write their acceptance criteria in Gherkin, and all test automated is expected to be written in Cucumber. The stakeholders however had little time on their hands, so the testers were left to write the Gherkin on their own. The developers just wanted to write code. They did not think testing was their responsibility so they gave no regard to the Gherkin. At this point, the automation engineer not only had to write the test automation, they had to write the acceptance criteria in Gherkin. It only mattered to the team that the build was red or green, and the Gherkin was just some text that was recorded in a log. Cucumber was supposed to make automation easier and faster, but it just felt like just extra work.

### 2.3.1 The Symptom



A business readable domain specific language will not make automation easier or better. Are you automating for the primary purpose of quality or are you automating for the primary purpose of fulfilling business requirements? If it is the latter then the automation project has fallen into the business readable trap. Automated testing is a solution to a testing problem to help prove that the application works well and correctly. The primary purpose of automation is not to solve business problems but to prove that business requirements have been fulfilled. Tread lightly when trying to solve business problems with automation, it is a symptom that can lead to automation death. Testing is about quality and automated testing should remain deeply rooted in quality.

There are tools out there, like Cucumber or FitNesse, that allow English, in the form of business requirements, to be used as a programming language for writing automation. In the ideal situation, the business owner or the product owner would write these. However, that is usually never the case and the QA team member has to write the business language in addition to coding the automated test. This is where automation starts to fall apart. Over reliance or over requirement of business language can lead the team to the point where automation is about making the requirement green and no longer about testing the application.

These types of test tools tend to become brittle as test cases become more complex, and will fail to scale when there are many tests. Their primary focus is on the presentation of the business requirements and making them green. As a result, they fall short when it comes to the actual automation and testing. Being written in plain English the tools often lack fully featured code development environments that help with writing and debugging the test. These tools also have poor support for parallel test execution, the ability to run multiple tests at a time to speed up overall test execution times. This makes it difficult to incorporate the tool into a Continuous Integration workflow. At this point, more resources are being spent trying to cope with the tool’s shortcomings rather than creating actual automated tests. This is a sign that it might be time to upgrade to a better toolset.

### 2.3.2 The Cure

The only way out of the Business Readable Trap is to upgrade the toolset to one that is rooted in quality and is about testing first. Do not just use a business readable automation tool for the sake of using the tool. Think about testing first and foremost. Identify a testing problem, then select a tool to solve that problem. Business readable tools are often solutions looking for problems.

## 2.4 Record and Playback

The team knows the value of automated testing, but there is a lack of expertise on the team. Testers get their feet wet by trying out some record and playback tools. They seem simple enough: start the tool, perform the manual test, it is recorded, and the test can be repeated by playing it back like an old-school cassette tape. It was an immediate time saver. As the number of tests grew it took longer and longer to complete the test, and even longer to fix the test when enhancements were made to the application. The solution was obvious, run tests in parallel to test faster and create reusable pieces of automation to reduce repetition and maintenance, but it does not seem possible with the record and playback tool.

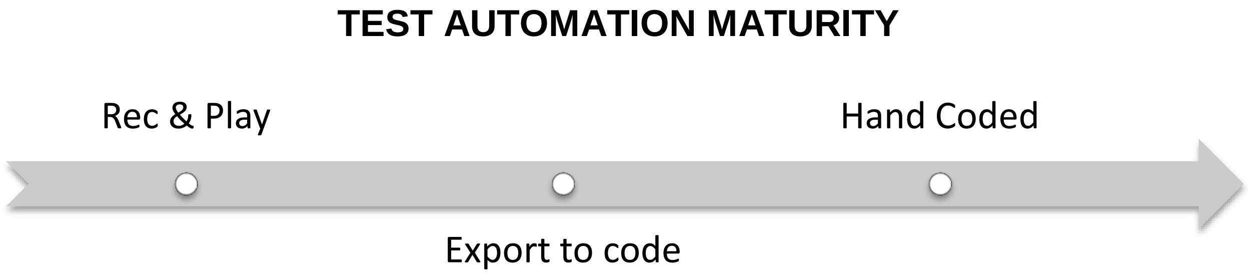
### 2.4.1 The Symptom

Strong reliance on record and playback tools will limit what you can automate. You have this symptom if you depend on record and playback tools to automate.

While record and playback automation tools are great for beginners, their utility quickly diminishes in more advanced situations. As a beginner's tool, it works well for a small number of simple to mediocre test scripts. Once you advance to dynamic websites, advanced interactions, or parallel test execution even MacGyver will not have enough duct tape around to hold the tool together.

### 2.4.2 The Cure

There are three levels of maturity that an automation framework should have the ability to advance through, and should advance through. Before creating too many tests at any level, forethought should be given in how the framework can be advanced to the next level.



#### 2.4.2.1 Rec & Play

The most basic form of test automation is record and playback. These tools record the user's interaction with the application. The recording can be played back to repeat the interaction. These are simple to use and creating a test is as simple as performing the manual test.

However, these tools can only record basic tests. Record and playback tools will most likely not support applications with dynamic elements, or advanced user interactions. When it is time to execute the tests, most will not support Continuous Integration tools like Jenkins or TeamCity. As your test suite matures and the number of automated test grows, you will want to run your automated tests concurrently in parallel. Again, most recorded and playback tools will fall short of your needs.

#### 2.4.2.2 Export to Code

As an added feature, some record and playback tools will allow you to export the recorded test out to code. This allows you to modify and used the recorded test in a native programming language rather than the proprietary language used by the tool. This opens up the support for Continuous Integration tools, and concurrent testing, but you are still limited by the type of interactions that the tool supports. If your application is dynamic or complex, corrections or heavy modifications will need to be added to the exported test. Even then it might not work.

#### 2.4.2.3 Hand Coded

The most advanced form of test automation is hand coding, where the bulk of the automation is coded by hand. Test automation is customized to the needs of the team and the characteristics of the application under test, in order to maximize utility and re-usability. This form of test automation is best, and can automate almost any type of application. It is also the most complex and needs to be treated like software development. All test automation should strive to achieve this level of maturity.

## 2.5 Tumors

As the automation guru, you have been asked to assist another team with automation. The previous automation engineer left behind mountains of documentation and a comprehensive suite of test cases. Investigation reveals that the automation takes hours to run, and when it finishes the reports are difficult to understand. Trying to incorporate parallel testing and improved reporting into the existing framework, turns out to be too difficult and time consuming. It is like trying to fit a square peg into a round hole. The automation toolset just did not support parallel testing, and the reporting improvements would require too much customization make it right. There are better tools out there, but what about the last two years worth of automated tests? Are they to be rewritten?

### 2.5.1 The Symptom

A poorly architected framework can grow out of control. This results from putting too many common functions into a single common library. It can also occur with keyword driven frameworks that have too many duplicate keywords or phrases. Test suites take too long to run, or we get reports that are just a wall of words. Technical debt can grow into tumors if the automation framework is not regularly evaluated, maintained, and refactored. This can result from poor initial design, lack of forethought, or due to team member turnover.

### 2.5.2 The Cure

Fight this symptom by using a Page Object design pattern, and a clean framework API.

#### 2.5.2.1 Page Objects

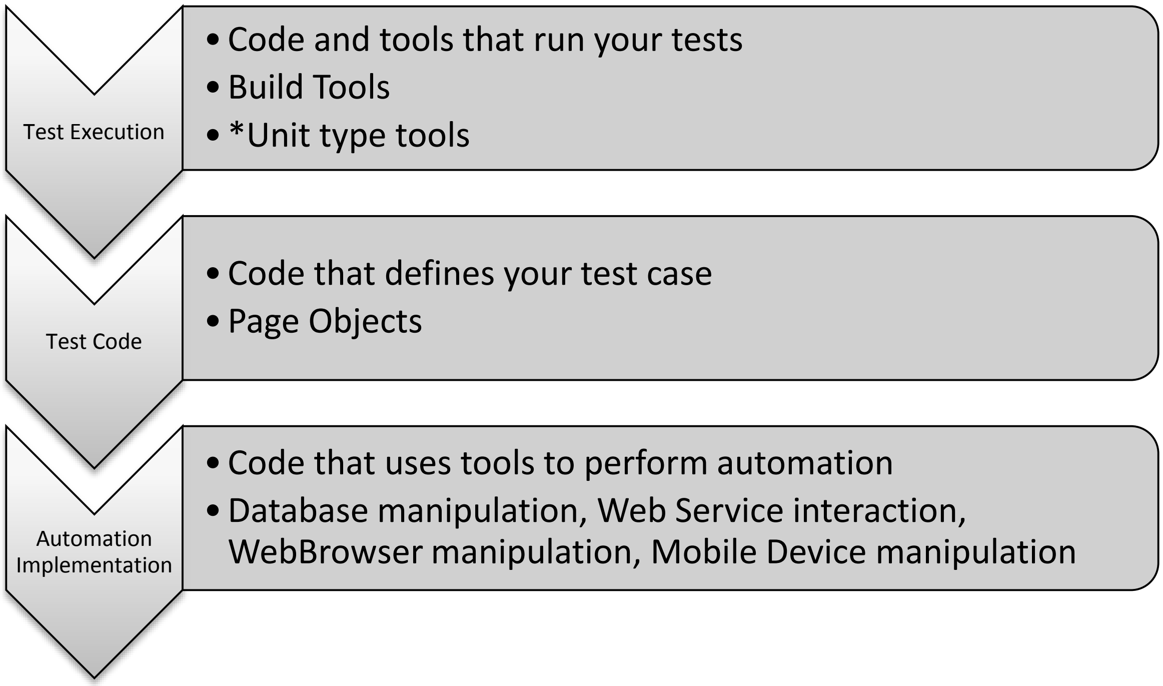
Your automation tests should follow the Page Object design pattern. The Page Object design pattern is an object oriented design pattern where each page or sub-section of a page in your application is represented by your test code as an object. The things on the page, known as the “nouns”, are the pieces of code used to identify the things on each page. The actions you can perform on the page, known as the “verbs”, are the things you can do on each page. That way the code is clearly organized and when enhancements or new features arrive in your application under test, updating the automation is as easy as going to the appropriate page object and updating the “nouns” and “verbs”. This is a widely used automation technique, mostly used by the web automation community, but it can be applied to all types of automation.

#### 2.5.2.2 Framework API

Whether you inherited it or created your own test automation framework, it needs to layered, modular, and the reports need to be actionable.

##### 2.5.2.2.1 Layered

To maximize longevity and maintainability of your test automation separate your test case code from your implementation code. Create layers between your test code and your test automation tools. Depending on your chosen development language, the layers will be implemented as separate files, objects, or classes. This will make your code easy to maintain and allows you to enhance, upgrade or change your test tool without drastically affecting your test code and test execution. These layers are not hard and fast, and the lines can blend, but the more you can separate these layers, the better your framework will be and the longer it will last.


Test Execution
• Code and tools that run your tests
• Build Tools
• *Unit type tools
Test Code
• Code that defines your test case
• Page Objects
Automation Implementation
• Code that uses tools to perform automation
• Database manipulation, Web Service interaction, WebBrowser manipulation, Mobile Device manipulation

**Test Execution:** This layer encompasses the code and tools that are used to run your automated tests. They are tools like Gradle, Maven, JUnit, TestNG, NUnit, and Rspec. These tools usually sit on top of your test cases and are responsible for running your test cases. Keep this layer restricted to test parameterization and execution. Keep automation out of this layer as much as possible. For example if you need to set up a database prior to running a test, create the database setup in the Automation Implementation layer and make method calls to it from the Test Execution layer. That way if your application upgrades to a different type of database, only the Automation Implementation layer is affected.

**Test Code:** This layer should contain only your test code and it should use the Page Object model described above. The test case will make method calls to the Page Object, and the Page Object will make method calls to the Automation Implementation. The benefit of this is that you have a single entry point for multiple automation tools, and you can swap out automation tools without affecting the test case.

**Automation Implementation:** This layer performs all of the automation. It pulls in many tools like web, mobile, web services, and databases and provides a single common interface for the test code. For example, you can use Selenium for web browser automation, and Appium or Robotium for mobile automation, but from the perspective of your Test Case, it is abstracted away and they are the same. The Test Case does not need to worry about how the automation is implemented; a click is a click regardless of the platform.

#### 2.5.2.2.2 Modular

The code that performs the automation should consist of many tools and should be modular. For example, a single API can represent both desktop web browser automation and mobile web browser automation; that single API can leverage different modules to perform the appropriate automation on the different web browser platforms. When it comes time to upgrade, the desktop module can be upgraded independently of the mobile module while maintaining a consistent entry point for your test case writers through the single API. The result is a lightly coupled automation framework that can change to meet the testing needs of the constantly changing application under test.

#### 2.5.2.2.3 Actionable

When a test fails, the report that results should be readable and actionable. What good is a test if it fails and no one knows about it, or no one understands why it failed? It should be as specific as possible and attempt to identify the location of the error. Use English whenever possible and avoid stack traces. If testing a UI, include screenshots in the report. The report should also be saved somewhere to allow historical reference to make it easy to compare a failed report to a previous passing report. To increase its visibility the report should also be compatible with Continuous Integration and build tools.

# 3 Conclusion

As detailed in this paper, there are many ways for a test automation project to die. Each one is treatable. The automation should be treated as a project, a never-ending project that undergoes constant evolution like google.com. There should be an experienced lead that keeps the automation framework two steps above the needs of the team. Instead of trying to train manual testers, try to encourage them to attend conferences and meetups; it is the best way to learn automation if they truly want to learn. Understand that it is a long learning process, so it will not happen overnight. In the end hiring more specialized automation resources will still be required. Unless the business is on board and is asking for business readable automation, do not do it. It is just an extra layer of work that will slow down automated test development. Record and playback tools are good starter tools and a good learning resource, but do not stop there. Record and playback tools are not made for serious enterprise level automation. Keep an eye on technical debt caused by automation. It can poison the automation from the inside out if it is left for prolonged periods.  Monitor and treat these symptoms and the automated tests will be more successful and easier to work with.

# References

Atlassian. “Working with Epics.” https://confluence.atlassian.com/display/AGILE/Working+with+Epics (accessed August 22, 2014).

Huggins, Jason. “Tapster from hugs.” Tindie https://www.tindie.com/products/hugs/tapster/ (accessed July 27, 2014).

Luc Besson and Robert Mark Kamen, Taken, Film. Directed by Pierre More. France: Europa Corp, 2008.

Tapster. “tapsterbot.” Twitter, https://twitter.com/tapsterbot.

Wikipedia contributors. “User Story,”Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/User_story (accessed August 22, 2014).

# Continuous Delivery: Bridging Quality Between Development and Customers

**John Ruberto**

(Johnruberto@gmail.com)

## Abstract

We were not moving fast enough for our business stakeholders. Development and quality was constantly the bottleneck for delivering the cool ideas from our product managers and designers. Our release cycles took three weeks, with more than half of the effort dealing with technical debt. We typically needed a full hardening sprint for each iteration. Our business team wanted to move faster. We had to bridge the gap between development and deployment.

What our business team wanted was a Continuous Delivery system, or even better a Continuous Deployment system. As soon as a story is complete, it's released to production. Each story does not have to wait for other stories, and our regression testing is completely automated.

We took this on as a challenge, and in six months implemented a Continuous Delivery system – where we push a story live on the day that story is completed. This paper will describe the changes we made with technology, mindset, and processes to achieve Continuous Delivery. It will also describe the principles, practices, and tools/methods which made this possible. Finally, a playbook is provided that can be used by others to make the same transformation.

Since we've implemented Continuous Delivery, our delivery velocity has doubled, the scope of the team has grown, and the engineering team is leading the charge on innovation.

## Biography

*John Ruberto has been developing software in a variety of roles for 28 years. He has held positions ranging from development and test engineer to project, development, and quality manager. He has experience in Aerospace, Telecommunications, and Consumer software industries. Currently, he is a Director of Quality Engineering at Intuit, Inc. He received a B.S. in Computer and Electrical Engineering from Purdue University, an M.S. in Computer Science from Washington University, and an MBA from San Jose State University.*

# Introduction

“Three weeks is not fast enough. We want to get these designs out now!”

Our team thought we were moving fast. We had a three-week release cycle and used Scrum to manage our projects.  However, our Product Management team didn’t think we were releasing fast enough. Our team also supported multiple project teams, with their own independent release schedules. Aligning their cycles with our three-week cycle was frustrating.

We decided to move to a Continuous Delivery model, where we could release any story as soon as that story was complete. Continuous Delivery eliminated the need to wait for system/regression test cycles, thus speeding up the delivery of individual stories. It also allowed us to more easily align with our partners release schedules.

This paper will tell the story about how we made this transition. The story involves changes in people’s mindset as well as changes in processes, technology, and architecture.

# Definitions

Consider this life-cycle view for the following definitions. Figure 1 shows an iterative Software Development Life-cycle that results in code deployment after System Testing. The “Define” stage is comprised of requirements analysis & definition, as well as iteration planning. For Continuous Integration systems, the Unit Test & Integration Test activities are fully automated. System Test in this context includes Regression Testing, System Testing, and User Acceptance Testing. With Continuous Delivery, these System Tests are automated. The Deployment phase includes staging, deploying to production, and deployment validation testing.

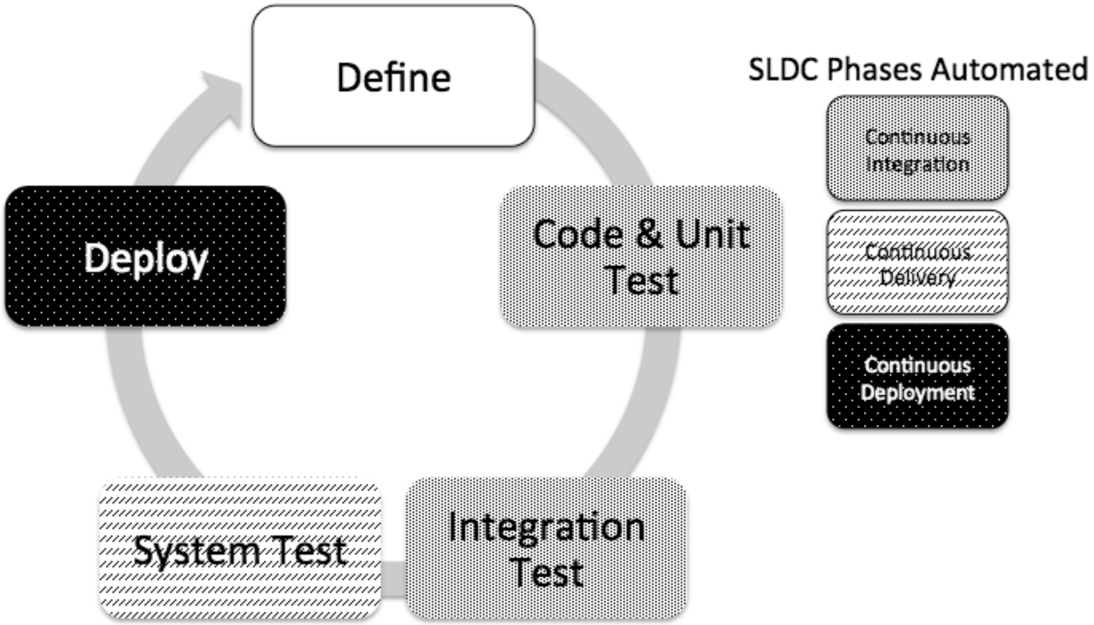


Figure 1. High level Iterative Software Development Life-Cycle with phases impacted by Continuous Integration, Continuous Delivery, and Continuous Deployment. Continuous Integration automates unit tests and integration tests, Continuous Delivery involves automating System test, and Continuous Deployment automates deployment.

## 1.1 Continuous Integration

According to Wikipedia, “Continuous Integration is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day”.

In practice, Continuous Integration (CI) means to us a set of automated tests that are triggered with each build. The tests are intended to give confidence that the new code developed for each build functions as expected and does not cause breakages in the existing code.  Non-functional tests are also executed, including performance tests and static analysis.

The intent behind Continuous Integration is to give rapid feedback to the development team and to keep the code healthy at all times.

Continuous Integration is a necessary pre-condition for Continuous Delivery.

## 1.2 Continuous Delivery

With Continuous Delivery (CD), we extend the concept of Continuous Integration to include customer acceptance tests with the regular automated test suite.

According to Wikipedia, “Continuous Delivery is a design practice used in software development to automate and improve the process of software delivery. Techniques such as automated testing, continuous integration, and continuous deployment allow software to be developed to a high standard and easily packaged and deployed to test environments, resulting in the ability to rapidly, reliably, and repeatedly push out enhancements and bug fixes to customers at low risk and with minimal overhead.”

In our practice, while the Customer Acceptance Tests are fully automated, the decision to deploy to production environments is still a decision made jointly by the project team and business stakeholders.

The value of Continuous Delivery, as practiced by our team, is to break the dependence on a regular release cycle (every three weeks) and instead deliver a user story or bug fix when ready.

Paul Duvall *[Duvall, 2014]* defines Continuous Delivery as “With Continuous Delivery (CD), teams continuously deliver new versions of software to production by decreasing the cycle time between an idea and usable software through the automation of the entire delivery system: build, deployment, test, and release.”

## 1.3 Continuous Deployment

Continuous Deployment extends the concept of Continuous Delivery by fully automating the production deployment of the application as well. Once code is checked-in, it is automatically tested and if the tests pass, is automatically deployed. No humans are part of the deployment decision.

For our application and business context, we didn’t see the need to use Continuous Deployment. The rest of the paper will concentrate on our experiences with Continuous Delivery.

# Benefits of Continuous Delivery

Building a Continuous Delivery system is quite a significant investment in technology and time. This section describes the benefits of Continuous Delivery and why a team might want to undertake this investment.

## 1.4 Increase Velocity of Innovation

For the project described in this paper, the primary benefit expected from Continuous Delivery was an increase in the delivery velocity of innovation. Our product management team had a long list of features and enhancements they wanted to build. Each one of these enhancements was thought to improve customer value and reduce our costs. So, delivering at the fastest pace possible was a strong desire. Our baseline release cycle called for a release every three weeks, but the team wanted to release, and benefit from, these features as soon as possible. Continuous Delivery allows for stories to be delivered when the stories are finished with code, testing, and acceptance – eliminating the wait for a "bundle" of stories to be delivered.

## 1.5 Planning Flexibility

Another benefit of Continuous Delivery is planning flexibility for release planning. The ability to release each story independently, and when that story is complete, takes away the need to plan which bundle of stories work best together. The team can simply work on the highest priority stories and release when ready.

Many of our stories are experimental in nature. We often implement two versions of a solution and monitor which solution performs better with customers. This practice is called A/B testing and is intended to identify the best solution to a given problem. Having the ability to learn which solution is superior soon, and deliver that solution to all customers rapidly adds to the value of our application.

## 1.6 Avoiding Technical Debt

One of the strongest benefits for building Continuous Delivery is that managing technical debt becomes vital; technical debt is kept to a minimum. One of the principles of Continuous Delivery is that the code base is always ready for production deployment. Each new story has to come with its tests, and code is only promoted to the next stage when all of the tests pass.

This discipline forces the team to rapidly fix bugs, keep automated tests running at high coverage levels, and exposes very quickly any gaps in the test coverage.

## 1.7 Teamwork Benefits

David Farley and Jez Humble wrote an excellent book on Continuous Delivery *[Farley, Humble, 2011],* where they described several benefits of CD:

- Empowering Teams
- Reducing Errors
- Lowering Stress
- Deployment Flexibility
- Practice Makes Perfect

# Continuous Delivery might not be appropriate in all cases

Continuous Delivery does have many advantages, but fully implementing CD might not be appropriate for all applications. Some situations may make it difficult to fully implement CD.

## 1.8 Requires a low cost of deployment

A key benefit of CD is frequent deployments. This advantage is only appropriate where the cost of deploying is low. “Software as a Service” (SaaS) applications seem to be natural fit for CD, where the developer is fully in control of the deployments, and deployment can be automated.

CD might not be the right choice for delivered applications, where the customer has to install or update the application. Frequently updating desktop and mobile apps might send the wrong message to customers, and cause customer frustration with having to apply frequent updates.

Other considerations for the cost of deployment include documentation updates, bandwidth for distributing code, manual installation, and training costs.

## 1.9 Regulatory environment might not be compatible

In CD, much of the accountability for quality revolves around the team instead of individual decision makers. The practice has few “signoffs” and the documentation trail might not clear. The regulatory environment for your application/industry should be considered before making a choice to use CD.

## 1.10 Difficult to Retrofit into existing applications

It was much easier to build CD into the new application, instead of retrofitting into the existing application. The technology choices and lack of testability in the older application made easier to start fresh.

## 1.11 Applications that operate on data

When moving fast with release cycles, and relying on fully automated tests, sometimes problems escape to production, making it necessary to back out a release. With applications that operate on data, this is often difficult to accomplish. If a new enhancement changes the way data used, and customer actually use the new functionality, it might not be easy to roll back the application to the previous state. Roll back is possible, but must be designed into the application accordingly.

# Context for our application

## 1.12 Self-help platform for multiple products

The project described in this paper is the Intuit Unified Self-Help Platform (USH). This is a web application that provides our customers with help information drawn from a variety of sources and content management systems. The information is provided through a web site, in-product help-viewers, and guided workflows. The intent of the system is to help customers find the right information they need to run their business without having to call technical support.

Providing the best information, which is easily digested by end customers, is very valuable to customers and to our company. Our customer gains value by finding the information they need quickly, without having to call support. Our company benefits from not having to field those calls. Also, our research has found that customer satisfaction is much higher for customers that do not have to call support.

With millions of customers, releasing an enhancement a few days earlier has very tangible benefits to the business.

### 1.13 Release Process

Before undertaking this project to refresh the platform, our team worked with a three-week release cycle. The first two weeks were allocated to planning and development, with the final week dedicated to system testing, fixing bugs, and user acceptance testing.

Figure 3 illustrates our release life cycle.

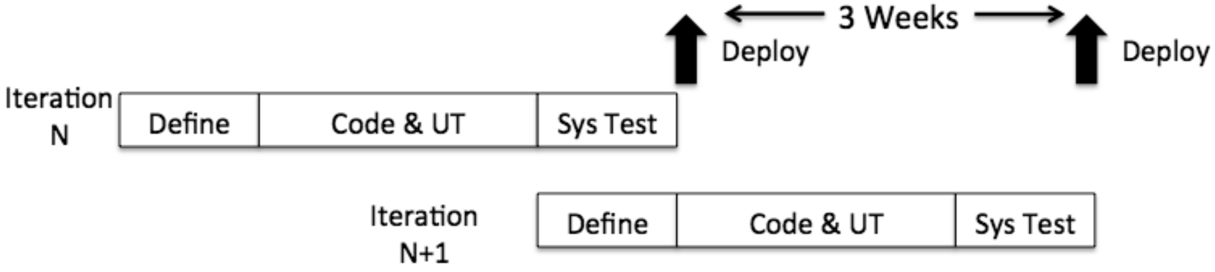


Figure 3. The original release life cycle before Continuous Delivery, with three weeks between deployments. Code & Unit Testing took two weeks, while System Test took 1 week.

### 1.14 Challenges

Even though a week was dedicated to testing and fixing bugs, our system still contained a lot of technical debt from previous projects. Approximately half of the effort for each release was spent fixing bugs and refactoring code. This extra overhead slowed down our ability to release customer-facing stories.

Most of the testing was conducted at the system level, with manual tests. We had very little test automation to aid the release. Executing these tests took time and effort, and any defects found during this time were found late in the release cycle, giving our team very little time to correct them before the release date. Many times, a story was scratched off the release because of bugs found late in the cycle.

Our customer base was growing and the complexity of small business accounting was growing. These factors drew many more customers to the self-help application, causing our costs to increase. Our business teams wanted to mitigate these costs as soon as possible.

### 1.15 Attributes beneficial for Continuous Delivery

This application had a few attributes that were beneficial for building a Continuous Delivery system. First, this was a web application; we controlled the deployments. We were free to release code when ready without customers needing to install any updates, take any action, or even know if we deployed an enhancement. Deployment was fully in our control.

The application also contained very little state information or business logic. These factors made it easier to implement a roll back, if that proved to be necessary.

# How we changed

This section tells the story of how we changed from a three-week delivery model to a Continuous Delivery model. Starting with the business rationale for making the change, the change leadership steps, technology investments, and the tools and practices that helped make this project successful.

## 1.16 Built the case for change

The business case for making this change started with conversations with the product management team. They were asking for certain stories to be delivered “off-cycle”, to be delivered earlier than the full release. This conversation occurred enough times for the development team to see the trend.

We did a thought experiment. Looking at the current process, code-base, and state of test automation, we asked how we could cut the test duration in half (this include unit, integration, system, and acceptance testing). The resulting brainstorm showed that we would have a very large investment and gain just one-week reduction in the release cycle. Continuing the thought experiment, we realized that it would be very difficult, and a long time, to incrementally improve our code & testing practices to create the planning flexibility required by the business.

Another factor that went into the decision was the state of our technology used for the self-help site. The platform contained many home-built components that had to be maintained. These components also lacked the automated tests that would lead to speedy release. Newer technologies were available, with platforms/frameworks that contained much of the needed functionality. Using this opportunity to refresh our technology and adopt an existing platform added to the business case.

Our engineering division also had some technology goals, namely to deliver more functionality in the cloud and to build our capability for Continuous Integration/Continuous Delivery. Our existing platform did not lend itself to hosting in the cloud, nor was it easily updated for Continuous Delivery.

We built the case for change based on the following factors:

- Faster time to delivery of each story (helps our customers faster, which reduces costs sooner)
- Technology refresh to a new platform will allow our developers to spend more time developing customer facing innovations, and require less time maintaining the platform. Over the long term, the productivity gains would offset the switching costs.
- Our hosting costs would be much lower by hosting in the cloud, and this prepared our system for global growth

## 1.17 Set CD as a goal

Once we had a good business case, we added Continuous Delivery to our annual goals. We made sure that business team set a goal for CD as part of their goals. We also made sure that our technology goals (CD, hosting, platform) were part of the larger engineering division goals.

Having our project on the goals for the business VP and engineering VP gave us the visibility to make this change happen. We were also given some extra budget to build the new system while maintaining the existing system.

## 1.18 Investments in people

We needed a team of people that shared the values of CD. Both the quality engineering and the development team had to share these values.

Since we were maintaining the existing site, in addition to build the new site, we were able to hire a few extra engineers to enhance the team. This gave us the opportunity to find people with both the technical skills required but also the mindsets described in the following table.

| Traditional Software Mindset | Continuous Delivery |
|---|---|
| **Structured hand-offs between functions (PM -> Dev -> Quality -> Operations)** | Collaboration between all functions. |

| | |
|---|---|
| **Build then test** | Test first (Test Driven Development and related practices) |
| **Quality team responsible for testing** | Everyone tests |
| **Tests reveal problems** | Accountability with each build (builds fail if tests fail, coverage falls, or static analysis detects poor code) |
| **Manual Deployments** | Deployment automation built in |

Table 1. Summary of mindset changes that helped foster Continuous Delivery.

Building quality in from the beginning is very important. Several of the new developers were “Software Engineers in Test”, and were therefore very adept at creating tests & they were the leaders of the “test first” mindset.

In addition to hiring new people, and transferring people from test roles, we also invested in training the existing team. We brought in a professional training company to give a one-week training for Ruby on Rails, our chosen technology.

## 1.19 Investments in technology

One of the constraints for releasing faster with the legacy system was the state of our technology. We had built the application several years prior, without designing in testability. Also, we build many of the components used, such as A/B testing framework, ourselves – without unit tests built in.

Here are the factors used for choosing the new technology stack:

- Productive programming language: ability to implement functionality with efficient amount of code
- A large library of components available to integrate, which increases developer productivity, and increases quality by reusing code.
- Popular with developers, to find a good labor pool and entice developers to learn the platform
- Framework that provides the Model View Controller (MVC) pattern, internationalization ready, abstracts the database, and lends itself to test driven development.
- Compatible with other technology initiatives in our company, to leverage learning from other teams.

We chose Ruby-on-Rails for these reasons, with the following major components and tools

| Component/Tool | Purpose | Link |
|---|---|---|
| **Ruby-on-Rails** | Main development framework | http://rubyonrails.org/ |
| **RSpec** | Test/Behavioral Driven Development, Unit Tests | http://rspec.info/ |
| **Rubymine** | Integrated Development Environment | http://www.jetbrains.com/ruby/ |
| **Git and gitflow** | Source Control and Branching | http://git-scm.com/<br>https://github.com/nvie/gitflow |
| **RubyGems** | Component Library | http://rubygems.org/ |

| | | |
|---|---|---|
| **RailsCasts** | Training and Learning | http://railscasts.com/ |
| **Jenkins** | Continuous Integration | http://jenkins-ci.org/ |
| **Chef** | Automated Deployments and environment configuration | http://www.getchef.com/chef/ |
| **Rubocop** | Static Analysis and style checker | https://github.com/bbatsov/rubocop/ |
| **Collaborator** | Code Review | http://smartbear.com/products/software-development/code-review/ |
| **Jmeter** | Performance testing | http://jmeter.apache.org/ |
| **Selenium Webdriver** | User Acceptance Testing | http://www.seleniumhq.org/ |

Table 2. Summary of tools and technologies used by the Unified Self-Help team to implement Continuous Delivery.

## 1.20 Agile Development Practices

We used Scrum for the overall project management, development, and collaboration process. We had used Scrum before and everyone on the team was already trained. With seven engineers, one scrum team was sufficient.

The Product Manager was a natural fit for the Product Owner role. This gave the product team full control over the stories and priorities, and they had the accountability for our expected business outcomes (reduced call volume).

One of the developers on the team played the Scrum Master role. This person has great interpersonal skills, the ability to follow up with people without adding a lot of friction; he was well organized, and had a strong personal disciple.

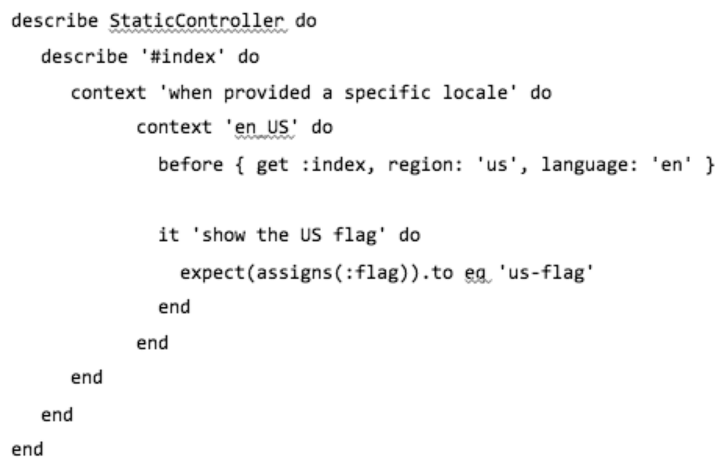
The Scrum team was comprised of developers (both front end and back end), quality engineers, and Development Operations engineers. This multi-disciplined team had the skills to tackle all of the stories in the backlog.

## 1.21 Practices

One of the core differences between the traditional model of software delivery and Continuous Delivery is the automated test coverage, and quality of these tests. Continuous Integration is at the core of Continuous Delivery.

Tests were built in from the start by using Test Driven Development using the RSpec framework to author and execute the tests. The basic practice involves the developer taking the user story and decomposing it into a series of lower level specifications. These are written within the RSpec framework, and become executable tests. Since the code to implement the functionality is not yet created, the tests initially fail. The developer then implements the functionality, and keeps running the tests until they pass.

This practice of creating the test first ensures that the application has high levels of test coverage. The following code example shows both the specification and test for displaying the US flag when the site is using the US English language localization. This specification and test is created before the operational code is created.



```
describe StaticController do
   describe '#index' do
      context 'when provided a specific locale' do
            context 'en_US' do
              before { get :index, region: 'us', language: 'en' }

              it 'show the US flag' do
                expect(assigns(:flag)).to eq 'us-flag'
              end
            end
      end
   end
end
```

Figure 4. Code example of a Behavioral test, which captures the expected behavior and implements a test for that behavior

Keeping the code in a constant releasable state is another benefit of Continuous Delivery. This is accomplished by using the automated test results as gatekeepers for promoting the code to the next level. The following diagram illustrates the four stages of code maturity that we used. These maturity levels correspond to, and are named after, the environments where the code is running.  These levels are Developer, Integration, Pre-Production, and Production.

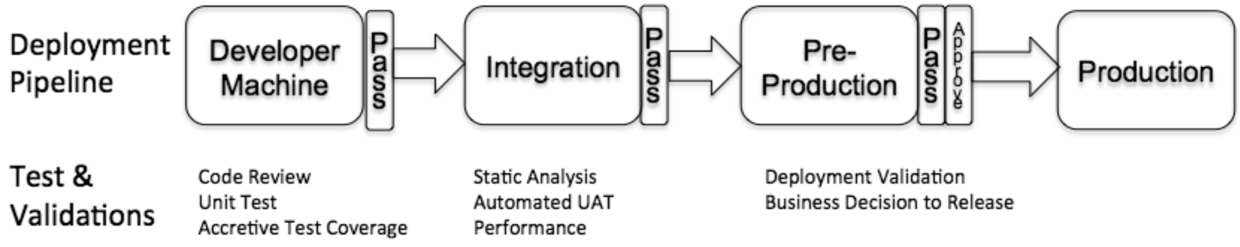


Figure 5. Development workflow showing the deployment pipeline with the tests and validations performed at each stage.

**Developer to Integration**

For any code changes to be promoted from the Developer stage, the code must have been successfully code reviewed, have passed all of the unit tests, and the measured code coverage does not decline. The team is very diligent and quality focused. This ensures that everyone follows these guidelines; they make themselves available for code reviews, and make sure to have good code coverage. Unit tests are reviewed along with the functional code, to help make sure that we have good tests in addition to tests that execute the code. These promotions happen several times per day, when initiated by the developer.

**Integration to Pre-Production**

The Integration stage tests are typically executed in the evening, with results ready the next morning. These can be initiated at any time by the quality engineering team in case of emergencies, but typically the automatic nightly execution is sufficient.

These tests include Static Analysis, verifying code style, potential errors, and bad coding practices. The suite of User Acceptance Tests (UAT) is also executed. These UAT are authored with Tekila *[Bhalla,* *Bhandari, 2010]*, our test automation framework based on Selenium Webdriver. Performance tests are also executed for the key workflows in our application. We used jmeter to create and execute these tests.

With successful Static Analysis, UAT, and performance tests, the code is promoted to the Pre-Production environment. We execute several deployment validation tests and monitor the pre-production environment while waiting for business team to decide on deploying to production.

**Release to Production**

The Scrum master and product owner decide to release a story by reviewing the test results and sometimes reviewing the business objectives behind the story. They give Dev Ops the go-ahead to release to production.

**Post-Production Validation**

We monitor the site availability, performance metrics, and key business metrics after production deployment. If any problem arises, we have the option to fix in place, or to roll back then fix. This decision is based on the severity of the problem and the estimated time to fix the problem. In practice, most of these issues are simple and it is just as easy to fix, rather than roll back.

## 1.22 Oversight/follow-up

Building Continuous Delivery into an application is a lot of work, with a long-term payoff. In business, there are many opportunities and pressures to see results quickly, putting pressure on the team to make shortcuts. We had a few incentives in place to make sure we stayed on track and actually delivered the application, with Continuous Delivery.

Our annual goals were shared through all of the levels of the engineering organization. The VP of Engineering for our division had the goal of building Continuous Delivery capability in the organization. We volunteered our project to be part of that goal. This gave us visibility at his staff meeting, where we gave quarterly progress reports and help when we needed help.

We also included Continuous Delivery as part of our department goals and the leadership team (development and quality managers). Each engineer had as part of their explicit goals the behaviors and outcomes required to make Continuous Delivery a success. These included test driven development, automation test coverage, and the team behaviors listed earlier.

We reviewed the progress of Continuous Delivery at each program review, and the behaviors were included with individual performance reviews.

The DevOps team is vital to the success of Continuous Delivery. We made sure to have dedicated support by converting a developer headcount to the DevOps team, which allowed them to hire someone with experience and dedicate to our project.

# Summary / Playbook

This paper described the experience that our team gained in creating a CD enabled application. The following list gives the “playbook” for using Continuous Delivery for any project.

1. Make sure you have the business need. CD is quite an investment in time and effort. CD is also a hot topic in software development, which makes it easy to try to force CD into an organization. Having a strong business need to deliver enhancements incrementally should be in place before attempting this.
2. Make sure your application is compatible with CD.
    a. SaaS applications work well, since you control the deployments without customer action.
    b. Your application should have the ability to be deployed quickly and automatically.

    c. Environments were failure might have extreme consequences or the process has high levels of accountability might not be compatible with CD.
    d. Your application also needs the ability to rollback in case of errors.
3. Make sure your organization is committed to CD. The process investment is high and has a long-term payoff. The technical and business stakeholders need to be willing to take this investment.
4. Make sure your team has the right mindset. Everyone on the development team needs to create tests, and to build quality in. The whole team needs to hold each other accountable to maintain this high level of quality. Traditional gatekeepers (QA, Release Management, Product Management) are giving some control to the development team and the process.
5. Make sure your processes are Agile. CD requires quick and efficient decision-making which requires quick feedback loops.
6. Make sure you use the right technology. Testability is vital in CD and this includes functional as well as non-functional tests.
7. Start small and build incrementally. CD is very amenable to incremental improvements. In our case, we started with just unit testing and code review for the very early version of the app. Then we added UAT, Static Analysis, and performance testing.

# Conclusions

Continuous Delivery worked well with this application. Since starting this project, I've seen more and more teams try to adopt Continuous Delivery and Continuous Deployment, in part because these practices are hot topics in industry. Having experienced this first hand, I've seen the importance to ensure that the business need is in place, the right people and technology investments are planned for, and the application be suitable for Continuous Delivery before embarking on this goal.

Having the ability to release at any time has been a great benefit for our application. Many times we were able to synchronize with an event happening in the company with our supporting functionality, without having to synchronize schedules. We were also able to move faster with more experimental functionality, knowing that we had the opportunity to fix it quickly, or back out, if warranted.

# References

Duvall, Paul M, "Continuous Delivery: Patterns and Antipaterns in the Software Lifecycle", http://cdn.dzone.com/sites/all/files/refcardz/rc145-010d-continuousdelivery_0.pdf. (accessed July 29, 2014)

Bhalla, Kapil and Bhandari, Nikhil. "Web Test Automation Framework with Open Source Tools powered by Google WebDriver." Proceedings of the Pacfic Northwest Software Quality Conference, 2010

Farley, David and Humble, Jez, 2011, Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, New Jersey: Addison-Wesley

Wikipedia, "Continuous Integration", http://en.wikipedia.org/wiki/Continuous_integration, (accessed August 2nd, 2014)

Wikipedia, "Continuous Delivery", http://en.wikipedia.org/wiki/Continuous_delivery, (accessed August 2nd, 2014)

# Software Quality Strategy Supported by People and Organizations

**Charles R. Matthews**
***The Boeing Company***

charles.r.matthews@boeing.com

## Abstract

It's almost axiomatic that software projects will fail to meet targets for cost, schedule, and quality. The body of literature concentrates primarily on technology insertion, methodology, and metrics. In addition to these dimensions, software quality and organizational culture provide important support to programs, ensuring effective communication, requirements management, organizational change management, and integration of senior management vision and goals with program targets. The phenomenological study focused on decisions made at critical points in software programs, drawing from the observations of 25 participants who had worked in all phases of the software development life cycle. From the responses, key lessons were learned that can help in anticipating or avoiding the pitfalls inherent in common government and industry software development practice.

## Biography

*Charles Matthews is the Manager of the Software Quality Engineering team at The Boeing Company in Mesa, Arizona. His 34-year career has seen activity in all phases of the software life cycle, including work in production environments in Seattle, Washington; Huntsville, Alabama; Washington, DC; and remote desert areas of Arizona. Chuck has worked extensively in support of building Quality Management Systems within Boeing and at suppliers of aircraft parts, ground-based communications, missile defense, and product acceptance systems. Outside of Boeing, Chuck facilitates courses in the software quality body of knowledge and management science.*

*Charles is a doctoral candidate at the University of Phoenix. He holds a Master's degree in Information Systems from City University, Seattle, and a Bachelor of Science degree in Management Science from Northern Illinois University.*

# 1 Introduction

Solving the mystery of quality software is not hard; software quality requires planning, discipline and effective communication. Software failures have been blamed for more organizational problems than any other product (Jones 2008). Failure to deliver quality software results in loss of ability to perform the mission, deliver critical services, and execute important initiatives within the government. Failure of the software in turn contributes to loss of reputation for both government agencies and federal contractors. The consequences of failure include loss of funding and jobs. Yet software provides a primary mechanism for commercial and governmental organizations to deliver products, services, and to meet their goals. Therefore, study of the collective decisions made and the result of those decisions on project personnel is important in learning lessons.

Software provides the potential to enhance effectiveness of systems, organizational capability, and to gain immediate benefits from changes (Turk, France, & Rumpe, 2005). However, both organizational and deliverable system change management is complicated by unintended consequences of decisions that introduce change. In a system with the complexity of an aircraft or an organizational management system, one change may be multiplied by the number of changes to legacy systems or programs, data structures, and schema, even while integrating the need for change to the organizational culture. Such a prospect adds difficulty to creation of new business vision or operational concept (Light, 2008). Federal Government practice often does not include assessment of system impact in an effort to align policy and process, which creates risk in terms of government expenditure.

# 2 Search for best practices

A growing body of research illustrates how software projects fail and how to improve software development and maintenance processes. United States government agencies have not made significant improvement in software development since adoption of the Capability Maturity Model (CMM) in the mid-1990s (Bennatan, 2009; Defense Science Board, 2006). Studies of software projects in 1990 indicated 83.8% of projects were behind schedule, over budget, or cancelled due to failure to deliver required functionality. A 2005 survey indicated 76% of software projects were significantly behind schedule, over budget, and unable to deliver as originally promised. In the nine years since 2005, data indicate similar results. Chronic over budget and behind schedule situations cost taxpayers in government expenditure and in citizens' ability to interact with government agencies (Lipowicz, 2010). The Department of Defense continues to seek improved tools to manage software procurement.

The problem is not new. The Carnegie Mellon Software Engineering Institute (SEI) was established to define best practices for software development, management and delivery of software system products. The SEI has developed and published models and frameworks that are adopted widely by government and industry organizations, agencies, and contractors. In the 1990s and early years of the 21st century, the government flowed down requirements to meet the measurement criteria of the CMM and Capability Maturity Model Integration (CMMI) representations. The CMMI, and complementary models published by the International Organization for Standardization (ISO) and Project Management Institute (PMI) provide guidelines for best practices in project planning, software and system engineering, and business process management. The purpose of each of these models is to provide a path to achieve business goals related to cost, schedule, and quality. Six Sigma, IEEE standards, and balanced scorecards support management system models. However, any process model adopted by the organization also requires cultural alignment and discipline across the organization to adhere to best practices when risks transform to issues.

# 3 Changing environments

The aerospace industry, as with many other industries, faces rapid changes in technology, increasingly intense competition for revenue and information, demand new functionality and modernization, and a faster rate of obsolescence as systems age. This environment puts pressure on procurement organizations across the government and prime contractors who must manage and control increasingly complex supply chains. Plug and play concepts and Agile development appear to increase the probability of project failure (Mosquera, 2008). A recent article in the *Software Practitioner* journal highlighted how fundamental misunderstanding of how Agile complements the traditional software life cycle process, instead of replacing it, contributed to the initial failure of the Affordable Care Act websites ("Was it about Waterfall vs. Agile?" 2014). Experience indicates that leaders and practitioners in an organization must understand the basics of software engineering and software project management when challenges present themselves, or crisis management occurs as leaders depart from early plans in attempts to meet business goals.

The body of available research points to a number of process areas as root cause for failure, such as project and program management, quality management, requirements management, and configuration management. Quantitative assessments have suggested a relationship between failure to adhere to quality software practices within one or more of the process areas identified in the CMMI. Within each process area, a collection of best practices have been developed that contribute to measurement and certification of maturity in a structured assessment (Paulk, Weber, Curtis, & Chrissis, 2006). The integration of best practices represent the most probable path to success for software projects and provide a foundation for a disciplined organizational culture that provides quality software products (Humphrey, 2004). Integration of best practices across organizational disciplines is necessary in addition to software and system engineering. Best practices are also needed in project management, quality, financial management, and operations.

## 3.1 The social organization

Many social reasons for project challenges and failures relate directly to adherence to best practices in organizational and project processes, including issues arising from organizational culture and the relative maturity of practices across projects. Increasingly, software engineers and project managers have engaged in understanding earned value within the software project. Earned value management increases awareness of cause and effect with respect to schedule deviation and how decisions affect suppliers, developers, and customers. Increased knowledge of cost, schedule, and quality metrics and the failure to meet targets leads to frustrated developers in addition to potential loss of revenue and corporate reputation (Bennatan, 2009). Misunderstanding of qualitative factors that balance quantitative factors represented by earned value contribute to failed software projects. Government Accountability Office (GAO) studies and academic researchers cite a growing list of software project failures for loss of confidence in the ability for government contractors and project managers to deliver systems capabilities required for government operational and management system needs. A review of government reports and peer-reviewed literature indicates causes that include lack of adequate training, too much oversight, too little oversight, unclear requirements, aggressive technology insertion, untried methods, obsolescence, and management practices across supply chains.

The GAO (2009) summarized the problem as follows: U.S. Government efforts fail to deliver software products as promised in spite of industry certifications for software development organizations. While government leaders and industry executives establish and exemplify an organization's culture and values, they fail to effectively manage expectations for delivery of systems solutions.

A software development organization exemplifies a techno-social system that combines intellectual activity with technology elements to solve a problem. The human and technology elements of this system are mutually dependent and are in turn highly dependent on organizational management and decision making of the human element, rather than the technical aspects of the system (Bennetan, 2009). A common element of organizations is the need to remain viable through achievement and maintenance of competitive advantage. Therefore, technology and innovation are vital factors in organizational development.

### 3.2 Social structure and organizational goals

Organization theory addressing techno-social theory has matured to account for organizational governance systems and systems that contribute to effective operation of the organization. Rapid advancement of technology and its effect on software development inspired foundational theories that matured into diffusion of innovation theory for information systems (Park & Yoon, 2005). Diffusion of innovation suggested the observable phenomenon whereby technology adoption and acceptance grows at an increasing rate around network architecture, wireless tools, and electronic communication toward a world where all systems and devices are connected.

Observation of software development organizations suggests that introduction of the various standards mentioned before – that is, CMMI, ISO 9001, earned value management, Malcolm Baldridge, Six Sigma, Lean Manufacturing – provide conflict among organizational paradigms, and both customers and senior organizational leadership contribute to this condition. Academic models suggest that competing elements across an organization are basic and stable, and individuals define themselves within the prescribed boundaries of accepted practices consistent with the organization's goals and objectives.

## 4 Study of common themes

A phenomenological study of 25 people who represented functional practice in seven process areas across the software development life cycle provided a view of organizational decisions and project performance. Participants were members of the prime contractors' respective organizations, representing process areas in each phase of the software development life cycle. The goal of the study was to explore organizational culture and decision-making approaches along the project timeline. Interviews focused on points in each program where difficult decisions were required. Five core themes emerged as presented in Table 1. Two supplemental themes emerged through observation of program management practices and root cause analysis.

Table 1. Percentage of projects observed within identified themes.

| Emerging themes in empirical study | Occurrence of theme |
|---|---|
| 1. Quality culture | 91% |
| 2. Internal decisions/deviations causing stressors | 100% |
| 3. External decisions/deviations causing stressors | 89% |
| 4. Delivery schedule as number one stressor | 100% |
| 4a. Multiple subcontractors contributing schedule impact | 86% |
| 5. Communication issues degrading project execution | 100% |

One supplemental theme emerging from the study data focused on the starting point of a program involving both government customers and contractors. Analysts who wrote the request for proposal and the corresponding proposal were seldom the people assigned to the program management office at the program start. Many contractors maintained professional staffs who captured business for the contractor; conversely, project risk seemed reduced in those organizations in which the proposal staff work engaged with the professionals whose job it is to execute the proposal. This approach to implementation prevents the pitfalls that cause risks to emerge by using the combined skills and experience of the proposal and implementation staffs.

A second supplemental theme suggested that U.S. government regulations promote unneeded bureaucracy that promotes inter-organizational conflict and additional overhead. The government organizational culture became misaligned with the software engineering culture in all cases observed. Observation further revealed that when third-party consultants were engaged to oversee a contractor, the third party aligned with the government organization, thus reinforcing the organizational incongruence. The existence and effect of organizational culture emerged as a common theme through the study.

## 4.1 Quality culture

During the project start-up phase, a quality process was agreed upon in all projects studied. Roles and responsibilities were defined, and each participant understood the organizational culture, policies, roles, and responsibilities. Participants were trained in the quality software process, organizational policies, and project standards. All project management participants, and 92% of participants in other functions (requirements, design, quality assurance, and configuration management) asserted that initial experience provide that both government and contractor program managers were interested in process discipline, most often using the CMMI and ISO 9001 based quality management systems. Leaders in the government and contractor organizations agreed to the process architecture, and senior management universally undertook training in processes. Meetings were held to establish reporting channels, quality oversight, and management for contractor organizations and government customers.

Software development leaders, typically software engineers, provided outlier responses. Follow-up interviews revealed most managers perceived that software design and development “just happens.” Since software code is not tangible, enforcement of process discipline was not considered a science: “Code was treated as a black box,” according to one participant. While measures of complexity were used in estimating the project, management did not understand them, and subject matter experts were not asked to provide insight. Therefore, senior management was not able to determine with any certainty why schedules slipped, why more labor power was needed, and why application of more resources did not resolve schedule issues.

## 4.2 Internal decisions causing stressors

A practice observed in some government agencies requires the proposal team leader to become the program manager. When a program faces immediate organizational change at program start, an element of risk is established at the outset of the program that challenges effective teaming. Use of a capture leader to get the business tends to drive aggressive behavior which encourages undercutting accurate cost estimates of the project. The proposal team and the project team represent different skill sets and create organizational subcultures early in the project that must be addressed later.

Such issues relating to misunderstanding of complexity and quality requirements led to the first program decision by the project manager, which was to not report to the customer deviations from plan as risks. Project and program management believed that early design decisions based on complexity analysis and early schedule pressure could be fixed by the software engineers. Internal decisions based on this belief did not require reporting to the government customer, and one project manager suggested, “The messenger bearing bad news is often shot.” This belief creates a culture where risk is not discussed openly.

## 4.3 External decisions causing stressors

When schedule and cost deviations were reported to the government customer, key project personnel were replaced or shuffled within the organization. Participants in the study interpreted these organizational decisions as *failure*, and no other explanation was provided openly so that appropriate corrective action and realignment could be taken internally. All study participants commented on lack of clear communication about personnel changes creating a change of approach in each functional area, which led to diffused collaboration as each function retreated into its own discipline. The perception of schedule as the number one driver to the project was reinforced.

A government practice of hiring third-party consultants when risks emerged reinforced organizational tension and further eroded trust. By contrast, establishing a relationship early in the project between the contractor and the third party provides less tension within the project team and between the contractor and government customer. The best dynamic is one where the contractor seeks to inform the third party assessor.

In 89% of responses, participants reported that government oversight from a third party resulted from reporting process and schedule deviation to the government. In this study, the third party was provided by a Federally Funded Research and Development Center (FFRDC). The evaluations provided by the third party cited other government-funded projects reported by the Government Accountability Office (GAO), Congressional testimony, and government publications to support their findings and recommendations. Recommendations included process and organization changes which were challenged by contractor program management, in turn creating an adversarial relationship between the contractor, the government, and the third party. Project members believed themselves caught in a no-win position and perceived a doomed project. Project participants concluded that senior management and other stakeholders had determined they knew how to deliver the software product better than the project participants. This perception was reinforced each time an organizational change occurred or a program directive was announced.

## 4.4 Delivery schedule causing stressors

The projects within the study planned for an iterative spiral-development model for software development, modified by tailored prototyping. The development model worked well in the beginning, but when projects came under stress for behind-schedule deliveries each project was forced to “deliver now and fix it later.” Study participants described how requirements were redefined to meet the deadline. Each participant admitted that quality was affected by the redefined requirements management practice, which bypassed procedures (shortcutting) and deviated from the planned software life cycle model. The customer’s confidence in the delivered product suffered when heavy emphasis on schedule led to minimizing the timeline for requirements verification, testing, and software qualification. Two participants noted program management direction to talk with the customer outside of the configuration management process to capture information and to generate software code resulting from the conversation.

## 4.5 Communication channels degrading execution

Each project continued to face challenges, and decisions made by the program management office were not communicated clearly or effectively. Early in each project, participants noted open and effective communication, but as organizational changes occurred, schedule pressure mounted, and deviations were reported to the customer, less information was passed between program management and project personnel. In three of the projects studied, organizational decisions were made on a Friday, new management was in place on Monday, with no reason for the turnover provided and no formal communication of the change.

Participants revealed that decisions not to involve personnel who would execute the project was a significant factor in later challenges and risks. While generation of the proposal and statement of work were significant activities, the proposal phase was not considered within the scope of the software development life cycle. Participants who listed the proposal phase as a major problem area suggested that since the government and contractors provide professional acquisition personnel to generate the request for proposal, few people who have experience in managing the mission or capability of the system are involved, and communication is limited from the outset of the project. Critical information was not shared with the project team until they had a *need to know*.

The second supplemental theme suggesting that organizational culture is a significant factor provides support for the fifth major theme as well. Inflexibility between government and contractor organizational interfaces reinforced a “dichotomy between the government and [contractor] skill sets and became unbalanced” as projects encountered increasing cost and schedule pressure. Trust issues emerged when government program leaders attempted to manage technology insertion and requirements were not fully understood or communicated. When the executing team began to slide schedules due to poor requirements and lack of definition, trust eroded until the emphasis for both organizations became to deliver the product at the expense of satisfying initial program goals. Effective knowledge management, processes, and practices were challenged in third-party assessment reports, which commonly recommended that the prime contractor develop new processes, methods, standards, and management processes which were better (and most) understood by the assessors.

Misalignment of organization and culture between contractor teams and government organizations illuminated the difference in quality practices, the requirements of which were not contradictory. However, the specific practices became a distraction for contractors to overcome through communication with the government customer. Late recommendations to alter practices created a negative impact to product delivery in terms of schedule, cost and requirements. In turn, cultural clashes damage the institutional trust between contractor and customer organizations.

# 5 Conclusions

Organizational decision making both reflects and drives organizational values, the culture, and performance of the systems that operate a company. Common misunderstandings created when difficult decisions are made affect perceptions and subsequent actions of project team members. One key conclusion projected that dynamics and practices within the organizational culture are dependent more on social dynamics than the formal process architecture. The social contract to follow defined processes must be reinforced through consistent and effective communication.

Schedule becomes the most common driver for a project as risks emerge and deviations follow. The pressure to deliver becomes the primary criteria for each organizational decision. Introducing new requirements or redefining requirements to give schedule relief creates additional risk as the team attempts to react to shifting priorities. Redefining processes, shortcutting, or waiving planned activities, lowers confidence – and therefore trust – in the project team, both from within the organization and the customer. The study suggested that the senior management directing the proposal team often does not understand the complexity of many software system development efforts and how the organization’s management systems work together to implement the organization’s vision and goals. Yet, the program manager, often a senior manager, attempts to manage the program/project to a predefined baseline not developed or understood by the team executing the project.

Effective communication is the most important factor in leading any software development effort, especially as challenges and risks emerge. Changes in program management, the program management office, or team leadership required communication of situation and context in terms of organizational definition, requirements management, expectations, and delivery. Organizational and program priorities must be communicated, reestablished, and reinforced. When the need for early and clear communication is ignored, the project team is drawn away from requirements management and compliance to process architecture and toward crisis management and delivering the product at all costs. In terms of the CMMI, heroes are created.

Further study is needed to determine the extent to which social structures react to project risk. What behaviors emerge as senior management decisions are made? Anecdotal evidence indicates that many software development organization avoid sharing qualitative data needed to understand organizational culture, and a gap in understanding of organization theory in software and system development organizations exists in formal study. Leaders in government and industry benefit from understanding the evolving social dynamics in software engineering beyond technology considerations. Understanding how people work together in and across organizations can reduce rework and cost and increase return on software development investment.

# References

Bennatan, E. M. 2009. “Project failures: Ignoring the warning signs.” *Advanced Project Solutions.* Retrieved from http://www.advancedprojectsolutions.net/

Defense Science Board. 2006. *Transformation: Process assessment*. Washington, D.C.: Office of the Under Secretary of Defense for Acquisition, Technology, and Logistics. Report 2006-04-DSB_SS.

Government Accountability Office. 2009. *High risk series: An update.* (GAO-09-271). Washington, D.C.: Author.

Jones, C. 2008. *Software quality in 2008: A survey in the state of the art*. Software Productivity Research, LLC.

Light, P. C. 2008. *A government ill executed: The decline of the federal service and how to reverse it.* Cambridge, MA: Fellows of Harvard College.

Lipowicz, A. 2010. “Boeing’s SBInet project under fire again.” *Washington Technology* (January).

Mosquera, M. 2008. “GAO says costs, schedule hamper IRS modernization.” *Government Computing News, 22*(17): 22-25.

Paulk, M. C., Weber, C. V., Curtis, B., & Chrissis, M. B. 2006. *The capability maturity model: Guidelines for improving the software process.* (2nd ed.). Boston: Addison Wesley.

Turk, D. E., France, R. B., & Rumpe, B. 2005. “Assumption underlying agile software development processes.” *Journal of Database Management, 16*(4): 62-87.

Was it about Waterfall vs. Agile? (2014). *Software Practitioner*, *24*(1): 6-7.

# The Fab Experience

How I stopped whining and started to appreciate Process

**Michael Stahl, Ron Moussafi**

michael.stahl@intel.com , ron.moussafi@intel.com

## Abstract

Many software teams struggle to implement and comply with quality processes. Adoption and strict adherence to process is seen as stifling, blocking innovation and redundant.

Contrast this with semiconductor fabrication plants (fabs) where adherence to process is the everyday norm and no fab engineer feels compliance with quality processes is optional.

Studying the top reasons why the fab world follows process so diligently reveals some underlying principles that can be applied to the software development world. Applying these ideas, software companies can improve the ability to implement and adhere to quality processes.

## Biography

*Michael Stahl is a 24-years veteran SW Validation Architect at Intel.*

*In this role, he defines testing strategies and work methodologies for test teams, and sometimes even gets to test something himself - which he enjoys most.*

*Before joining R&D, Michael worked for 10 years in Fab 8. In this paper, Michael draws upon his experience in the fab, in search of improving software quality.*

*Michael routinely conducts training sessions inside Intel, presented papers at a number of international conferences, and teaches a course in SW testing in the Hebrew University.*

*Ron Moussaffi is a 27-years veteran, currently managing a SW/FW Validation group at Intel. In this role, he leads a cross-site Validation team supporting on-chip embedded Firmware and Software products.*

*Prior to his current role, Ron worked for 23 years in Semiconductor factories, in a variety of Systems, Manufacturing and Technology leading roles. In this paper, Ron draws upon his experience in Quality management and improvement in Semiconductor Manufacturing, in search of improving software R&D quality.*

# 1 Introduction

Software development processes are nothing new. “The Mythical Man Month” – a classic milestone in software development thinking - was published in 1975 and is still largely relevant today. There are 40 years’ worth of data, research and experience that shows how adoption of processes such as requirement management, reviews and unit testing help produce high quality software.

And yet, most teams ignore at least some of these learnings. Most projects don’t have proper requirements; most teams do not do thorough unit-test; peer reviews are done on best effort basis.

There are many reasons for this situation: developers who think that structured processes are an obstacle to get real work done; managers who don’t feel comfortable enforcing processes that their team objects to; plain old lack of knowledge about processes and the impact of non-compliance; distortion of methods to avoid parts of the process that seem like a drag and many more.

This paper does not delve deeply into these reasons. We believe they are mostly common knowledge.

Instead, we want to take a look at an engineering community who DOES follow a strict process to the letter and try to understand how and why it works for them. Once we know that, we can look at what can be applied to software development.

The engineering community we refer to are the engineers who work in silicon manufacturing facilities (also known as “fabs”).

Everyone who works in a fab conducts all activities in accordance to clearly written specs. The specs are always up-to-date. Nothing is changed without documentation, review and approval. Everyone – from technician level all the way to the Principal Engineers - follow the process in their daily work.

Why is this so? How is it that whole organizations, hundreds of people, follow strict process, something we, in software, can’t seem to get even a small tight team to do?

We believe that the fab culture holds some of the keys to adoption of proper software development processes. This paper first introduces relevant details about silicon manufacturing and some of the quality management principles used in fabs. It then identifies four areas where the principles and behaviors of the fab world differs significantly from what we have in software development teams. The paper then suggests how these principles can be applied to the software development world in an effort to improve process compliance, which in turn will improve the overall quality of the developed software.

# 2 Silicon Manufacturing Basics

Today’s integrated circuits are nothing short of a miracle. To give you an idea why we think so, let’s look at some numbers:

Intel’s most advanced CPU (Central Processing Unit) contains about 1.4 billion transistors. Each of these transistors is made of structures that are about 20nm in size (1 nm = 1 nanometer = $10^{-9}$ meter). If you lay 4,500 such transistors in a row, the total width will be approximately that of a single human hair.

Integrated circuits are manufactured on silicon wafers – a very thin and highly pure silicon base. Today’s newest fabs process 12” wafers – the size of a vinyl long-play record. About 500 CPUs can be manufactured on a single 12” wafer. During the manufacturing process, a wafer passes through more than 400 manufacturing steps. Each step is a stage in the process where the wafer is physically changed: a new layer is added, material is etched away or sputtered on, films are grown, impurities implanted into the wafer surface.

To get a commercially viable product, the yield at the end of the process must be in the high 90’s percentages (yield = percent of functional integrated circuits on a wafer at the end of the process).

As an exercise, assume that the yield at any step is 99.9%. That is, only 1 integrated circuit out of 1000 is damaged at each step. With 400 steps, the yield at the end of the line will be: $(0.9999)^{400}$ which is 67%! This is an intolerable low yield that will make the whole operation not profitable. The yield in each step needs to be closer to 99.99% (one damaged CPU in 10,000) in order to achieve a yield of 96%. This means that almost nothing should go wrong in the 400 steps - a very tall order.

Achieving such tight control over the manufacturing process is made possible by continuously being VERY careful about every aspect of the process.

In the fab, everything is monitored: temperature, humidity, air-born dust level, incoming material composition, gas flows, liquid concentration, electric power values, air pressure, etc. Each and every quantity is monitored by control charts, and the minute any monitored parameter starts to deviate from the normal, machines are put on hold and someone gets an alert.

# 3 Quality Management in Fabs

All activities in the fab are regulated by specifications and well defined processes. Every person in the fab who has anything to do with the manufacturing process must first read the specs and confirm by signature he/she understands the work procedures. When a change is needed to fix a problem, there is a specification how changes are implemented: what experiments must be done; what data must be presented before a change is allowed. As far as possible, nothing is left for chance.

Things do sometimes go wrong. A machine breaks down in an unexpected way; someone makes a mistake. In fab lingo, these situations are called “Excursions” – a severe deviation from the norm. In such cases, retrospectives and structured problem solving techniques are used to find root-cause for problems. Process fixes are designed that will eliminate the problem, preferably in a way that won’t allow it to happen again.

In the fabs, Quality is the factor that guarantees a product at the end of the line and therefore “Quality is #1 priority” is not a cliché; it’s a way of life. Without tight control over every aspect of the manufacturing process, the yield will be too low or the resulting products won’t last long enough in real life use. If you can’t maintain the highest possible quality level you may as well just close the fab.

To achieve the level of quality needed, fabs maintain a comprehensive quality management process. The quality processes govern the following aspects:

- Metrics & Controls (myriad parameters are measured and tracked: e.g. incoming materials quality, inline machine parameters, on-wafer electrical parameters).
- Excursion identification and containment (short time from the occurrence of an excursion to the identification that it occurred; quick lock-down of impacted materials and fab areas)
- Disposition (how to deal with impacted material)
- Root Cause analysis (why an excursion happened)
- Fix design and implementation (avoid the excursion in the future)

Each of these aspects is in itself a long list of activities and processes. There are also supporting processes related to workforce training programs; change control regulations; Manufacturing process methodology (e.g. Lean). Appendix A gives an idea how comprehensive these lists are.

Everyone in the fab sees quality as an integral part of their responsibilities. As a result, the QA department does not OWN quality – it just facilitates everyone else in maintaining quality in their operations.

The quality processes are also used to achieve continuous improvement of the manufacturing process. Engineers can propose process improvement. The ideas are tried out and tested in accordance with change control processes. If the results are positive, the change is accepted. If the results are inconclusive or negative, the change is rejected. This provides a balance between the wish not to change anything (“if it works, don’t mess with it”) to the benefits that may be gained by a well-designed change.

For someone not familiar with the fabs (such as software developers), the heavy focus on careful change management seems to indicate that the fab environment stifles innovation. The reality is that there is a LOT of innovation in the fab world. New Technologies and Products introduce constant challenge to the factory line. Engineers are continuously improving the manufacturing processes or find ways to reduce costs. Both authors, who worked at fabs during part of their career at Intel, were personally involved in highly innovative projects that streamlined the manufacturing process and saved millions of dollars. Yes, it was done carefully; but it was definitely innovative. We never felt stifled or held back due to the fab processes. Still, you may ask: what has this to do with software development?

# 4 Software Development Quality Processes

Maybe the reason that processes are less dominant in the software development domain, is that there are not that many well defined and proven processes?

The answer is a clear NO. There are many well defined processes. There is ample data to prove the benefit of implementing a variety of processes at different stages of the development process. (Blake et al., 1995), (Intel, 2010) are just two examples.

Processes like inspections, unit-testing, continuous integration, as well as development models like Agile, Pair-programming, waterfall, V-model, Spiral and CMMI (Capability Maturity Matrix Integration) are familiar terms to many developers. As an example of quality management framework for software development, we can take CMMI. CMMI defines five maturity levels:

1. Initial
2. Managed
3. Defined
4. Quantitatively managed
5. Optimizing

A set of software development processes are associated with each level. A CMMI level is achieved when the processes associated with this level are implemented and used by an organization.

While many software teams do not use CMMI as a framework for quality management, one can survey the actual processes used by a team and draw parallels to CMMI processes. Our personal experience tells us the result of such a survey will show that most teams operate somewhere between CMMI level 2 and 3. This experience is also supported by research (Carleton, Anita, 2009); (Elm, Joseph P. et al., 2007).

While seemingly different, there are many parallels between fab and software development processes (see appendix B). This similarity allows us to assess the fab's "CMMI maturity level" by comparing fab processes to CMMI-defined processes. Doing such comparison will reveal that fabs operate consistently at what would be the equivalent of CMMI Level 5.

How come so few software shops are at CMMI-5? Why do software developers find it SO HARD to do what fab engineers think of as "WAY OF LIFE"?

In the next section we will list a number of differences between the fab world and the software development world that we believe are the reason for the different attitudes towards Process.

# 5 Software Development vs. Fabs

The experience of the authors in both fab and software development, puts us in a position to suggest some major differences between the fab and the software development worlds. We believe these differences explain the different attitude towards process.

Additionally, we propose how adopting some of the fab culture and management practices may emphasis the value of development processes and thus increase the chances that teams will implement these processes.

## 5.1 Cost of Error

In the fab, a single mistake can cost millions of dollars. A human error in setting up a machine may cause all the wafers processed by that machine to be defective. Between the time when the error happened and the time it is noticed, a good number of wafer-lots may be processed by the problematic machine, each worth many thousands of dollars. If the error persists for a few days, it may cause enormous losses to the fab. Not only the cost of the raw materials, processing costs and workforce costs are involved, but also the cost of lost sales. The loss is immediate. Within hours or days of making the mistake, the full monetary impact of the mistake is clear.

Everyone in the fab understands the connection between mistakes and the associated costs. People want to be accurate and careful since the impact of not being careful is very tangible.

By comparison, in software development, the cost of error is unknown and hardly felt. What is the cost of a coding bug? What is the cost of a mistake entered at requirements gathering time? We are taught it's 10x more expensive to find a bug in Test rather than at requirements stage, but how much is X? That is a very hard item to quantify. Another factor at play is that fixing a bug is many times just a few hours' work. Not a big deal. Hardly any pain involved. Developers do not see the hidden costs of bugs: The added testing effort; the costs of releasing an emergency patch, There is no big chart showing how these costs accumulate. Indeed one bug is not terribly costly. But it is common to have thousands of bugs in a large software project. These costs add up!

Even when slipping the schedule has clear monetary impact such as delay penalties, many mistakes are made months or years before ship time. It is hard for a developer to link skipping a design review with the risk of late shipment and the associated monetary loss.

In short: Fab engineers have a very strong mental connection between making mistakes and losing money, while many developers do not. Trying to promote process adherence as a method to reduce the cost of errors makes a lot of sense for fab people. For software people, it does not resonate well since there is no clear linkage between bugs and high costs.

We believe that knowing the cost of bugs will cause software developers to be more conscious how their work impacts the bottom line. This in turn will provide a strong incentive to adopt processes that may help reduce this cost. When bug costs are known, they may be associated to the bug-injection point. This in turn will provide the ability to calculate projected ROI for processes that reduce errors from each development stage.

Once we know how to calculate bug costs, the next step is to make these costs visible at program level. For example, when a bug is found in test, a cost chart should show the impact. Visibility creates an incentive to reduce costs; no one wants to be the cause of a jump in the cost chart.

The problem is that it is difficult to create a clear measurement of cost-of-bug. Fixing a SW bug is relatively low effort, however the indirect cost can be substantial. Many hard-to-quantify costs are involved: the cost of reporting, triaging and reproducing the bug; the amount of validation effort the bug-fix triggers; the cost of creating and distributing a new release; the indirect cost of context-switch a developer have to do from their current work; the risk of lost sales. These are just a few of the parameters that add to the cost of a bug.

For programs with clearly associated delay costs, such as delay penalties, one approach may be to calculate the cost of an hour's delay in shipment and associate this cost to the time it takes to fix the bug.

Overall, we believe calculation of bug costs is an uncharted area that calls for further studies.

See (Holzmann, Gerard, 2012) for an example of creating a direct link between mistakes and costs (min. 17:45 in the video).

## 5.2 Accountability and Ownership

As part of the quality management principles of the fab, any misprocess event is investigated for root cause. Whether it was a human error, a spec that was not updated on time or an unexpected machine failure, the location of failure will be isolated. Each area has a clear owner and as the investigation proceeds, the responsible persons will be identified. The involved people know that a large monetary loss, delay in shipments or other negative impacts are associated with their area of responsibility. Not a comfortable position to be in, even if it does not mean punitive action (in most cases there are no serious personal ramifications, unless the problem was caused by gross negligence or violating specs). Since fab culture cultivate the notion of "everyone owns quality", the responsible people feel … responsible.

In the software world, we have many situations where ownership is non-existing or unclear. Documents are written as a one-time-shot and never updated to reflect the final product; architects design a product, move on to the next product before the current one is done, leaving any design problems to be solved by someone else. If there was a major architecture flaw, they may not even hear about it. Quality is owned by the QA. Developers don't feel they own quality – this is why there are testers. Bugs that are found in the field are Test responsibility. In some convoluted way, one may say that developers are almost expected to produce bugs in the code. There is hardly ever an effort to trace bugs to the responsible developers or to initiate root-cause analysis for bugs located in-house.

Ownership and accountability is a major characteristic of the fab work environment and people are proud to be owners of specific processing area.  Ownership and accountability in software development is fluid at best, and there is little connection between cause and effect when failures occur.

Process helps to avoid mistakes or provide a proof you were compliant. Process is therefore seen by fab people as a support tool and the fact you can't do whatever you want is seen as a reasonable price to pay. Software engineers are not commonly held responsible for their mistakes, so have less inclination to take on the burden of adhering to specs.

We believe that software teams need to emulate the fabs in that "quality is everyone's job"; that people are expected to own up to their responsibilities. Quality as everyone's job is already part of agile development methodologies, so it seems we are in the right direction. We need to stop treating bugs as "integral part of the process" and start treating even internally found bugs as Excursions - something that should not have happened. Excursions call for root-cause analysis to find and eliminate the problem's source. One of the side effects of such analysis is finding the responsible area and the responsible individuals.

We propose to measure the Development team's efficiency not only by the extent of new features they create, but also by how much it costs to Test and Integrate them properly. Create quality metrics on per-module basis and make the defect-creating modules stand out. Suddenly, each team would have a professional-prestige stake in eliminating bugs. See (Holzmann, Gerard, 2012) for an example how per-module, public quality metrics help achieve better quality (min. 34:25 in the video).

Root cause analysis of bugs reported by important customers is something that does happen often, due to the visibility and impact of these bugs. However, we suggest to change the focus of such analysis from "why test missed the bug" to "why did the bug happen".

Documentation is another area that will benefit from ownership. While agile process value working software over documentation, it's also a fact of life that good documentation will help you achieve working software. Define the level of documentation you think is beneficial and hold the owners of these documents accountable for keeping them accurate and current.  Un-update documents should be reported as bugs and fixing the documents should be tracked it as part of the project's deliverables.

## 5.3 Culture

In the fab, quality and compliance are values. People are committed to quality and follow the process knowing that this is the only way to have a viable product. No one is allowed to be non-compliant, and there is zero-tolerance for those who violate the specs. Control systems are in place to ensure everyone is trained and updated on procedures and as much as possible lock-out engineers from doing their work (such as maintaining machines) if they did not read the latest spec update. Everyone in the fab follows the rules; there are no exceptions for unique people who are allowed to operate "their way". With that, no one feels a sucker when following the spec. Everyone does it. It's the norm.

In the software world, flexibility is a value. Processes are seen as stifling innovation and blocking progress and therefore bad for getting things done.

The reality that indeed one CAN produce a software product even when not following a process makes managers less confident when they try to impose a process on the team.

Many teams have one or two amazing developers that are so good the managers just let them do whatever they want as long as they crank out code. The fact that some people are exempt from following process has a very strong impact on the other team members, who get the message that following process is for lesser beings.

Even when processes are adopted, teams tend to take well defined and thought-through processes and do only the parts they like. The result are processes that sometime work and sometime not, "proving" the notion that "processes don't work for us".

The strong fab culture causes any new engineer entering the fab to adopt compliance and quality as a main theme in the work place. The opposite is true in software. New college graduates starting a new job see that most of the processes they learned about in university are not implemented or even mocked. The culture calls for flexibility, speed and trying new ideas. Process is seen as adding delay and unnecessary overhead to the overall cost. There is disbelief and ignorance about the positive impact of solid processes. The new engineers adopt this attitude and forget their education.

We believe that the most disruptive behaviors related to process adoption, is exempting some individuals from adhering to the process. If everyone follow the process, no one feels entitled NOT to follow the process; if some people are exempt, everyone treats the process as a “best effort” item. You may be encouraged by this: It’s hard to start mandating processes, but once the culture is changed to be one that embraces process, keeping it up calls for much less energy. It becomes just “this is how we do business here”.

Management should instill a Zero Tolerance to process violations. An example may illustrate this well: A few year ago, Michael led the deployment of Requirements Management tool in his organization. The pushback from the engineering community was strong and anyone who had some kind of half-justified argument allowed himself to avoid the system.

At a certain point, the manager of one team announced that he heard enough about the system’s shortcomings and it was time to move on. He mandated that within two weeks each and every team member will load a requirement document to the tool, regardless of their reservations. That team was the only team who really embraced the tool and realized benefits from its use. Other teams kept procrastinating for quite some time; some of them never used the tool.

See (Holzmann, Gerard, 2012) for an example of imposing “everyone has to do it” attitude to agreed-upon processes (min. 23:40 in the video).As for stifling innovation: processes may actually free time for innovation. For example: when documents are all updated and all reside in an agreed upon location, time is saved for anyone who needs them. Skipping process frequently causes inefficiencies and therefore leaves less time for innovation.

## 5.4 Holistic view of the Process

In the fab, the mission is to run a smooth and efficient production facility. To achieve this, all parts of the line need to work in perfect synchronization. Workers at each step understand how the processing that they do impact the end product. Engineers are aware of the interactions between different stages in the manufacturing line. When something goes wrong, there is a very short feedback cycle to the location where the problem started, so the cause-effect linkage is strengthened. Engineers see the process as a whole and understand their role in this whole.

In a development environment, the mission is R&D. Many products have some technological aspects that are very new and not necessarily fully solved. In such an environment, factory-level efficiency is not expected and tight interaction between the moving pieces is not seen as a goal. It’s the other way around: decoupling is a value and abstraction layers are built to hide details not necessary for integration of modules. As a result, some developers – especially in large projects - don’t have a holistic view of the whole product. Everyone works on their piece of the code, trusting the defined APIs to make things integrate well. On top of it, some of the feedback cycles are very long. It may be six month before a developer learns a mistake he did in the code resulted in a costly delay or a hot-fix.

Seeing the product as the main mission; seeing it as a whole from start to finish, increases the notion of overall responsibility for outgoing quality and is a common sentiment in fabs. It makes every worker, at every step, willing to abide to the process, which helps reduce the risk of negatively impacting the next steps.

Seeing R&D as the main mission, puts the focus on “making it work”. The focus is not on how efficient where you in finding the solution; nor how easy it will be to test or maintain. Many software development processes call for upfront investment to save back-end effort. When downstream efficiency is not a goal, adoption of processes suffer.

There is a view that software development cannot use production-type processes, since every product is unique. Therefore, flexibility is a must and mandating processes make the development team not flexible enough. We think otherwise: Creating software goes through rather generic steps: requirements, architecture, coding… in each project the CONTENT is different, but the PROCESS of creating a product is pretty much the same each time. You want to have reviews, unit test, CI, source control, change control. The process is the level you can standardize without losing the ability to produce a wide variety of software products.

The development process will be streamlined if we start thinking in term of “Software Production” and not just “software development” – simply because the generic supporting processes can be made to conform to production-like controls.

Once you have a stable Software Production framework, you have a stable process. Now you can start measuring and monitoring the impact of one step of the process on the next one, and initiate effective continuous improvement activities – just like it’s done in the fabs.

Another aspect that can be adopted from the fab is the notion of overall owner. A senior process engineer is assigned at the chief technologist for the manufacturing process. This is the person who runs the change control board. This person’s role is to understand the full flow of the production process down to the details. This is the go-to person for questions about cross-steps interactions. This person helps design tests to check a proposed process change. A similar approach is software development will increase the cohesiveness of solutions and avoid the risk of different teams going in different directions.

Another approach, from Toyota, is to keep the same people on a project from start to end:

> "Toyota organizes teams around complete projects from start to finish. For example, the design phase of the interior of the vehicle is the responsibility of one team from the design phase through production. Having the responsibility of participating in the project from beginning to end enriches and empowers the employee."  (Liker, Jeffrey K. 2004, p. 196)

It is our view that it not only empowers the employee. It also creates an incentive to invest upfront in doing things properly, since the benefit of this investment will be ripped by the same people. This notion works well for Agile teams where the boundaries between developers and testers are not rigid and developers have a concrete stake in bug reduction since the entire team will suffer from last minute critical bugs.

## 5.5 Engineers Self Image

In the fab, engineers see themselves as scientists. The image impacts the work style: fab engineers are careful, meticulous, methodical, make decisions backed by data. They can’t afford to be reckless: a single experiment may take 10 weeks to run through the manufacturing line and cost many thousands of dollars. Things must be planned, analyzed and thought through before implementation.

Developers, on the other hand, see themselves as creative beings with unlimited potential to experiment. They are willing to try ideas without totally analyzing them first. Implemented in code, any mistake can be fixed quickly: the next version of the code is just a button click away.

In the fab, experiments take a long time by definition (it takes weeks to months to run a single experiment). The extra time added for following process does not change this timeline in a significant way. It also helps avoid silly mistakes which may invalidate their results. For a developer, just coding something seems much quicker than having it first documented and reviewed. The fact that taking the seemingly longer documentation and review path may eventually make a better solution that works the first time is hard to believe when you KNOW you can hack the code in two days. Worst, if it does not work – two more days to fix it.

This difference in attitude is not something we propose to change, but is an important parameter to remember when attempting to increase the adoption of processes in a software development environment.

# 6 Summary

We described and discussed some fundamental differences between the fab and software development.

While these two worlds are very different, we believe there are aspects of the fab environment that can be adopted by the software development community. If adopted, in full or partially, these aspects will help drive process acceptance and process adherence in the development world, resulting in higher quality of software products, achieved with less thrush and less rework.

# References

Blake Ireland, Ed Wojtaszek, Dan Nash, Ray Dion, Tom Haley, 1995, *Raytheon Electronic Systems Experience in Software Process Improvement*

Carleton, Anita, 2009, *CMMI® Impact,* slide 8. http://resources.sei.cmu.edu/asset_files/presentation/2009_017_001_22751.pdf

Elm, Joseph P. et al., 2007, *A Survey of Systems Engineering Effectiveness – Initial results*, p. 33 http://resources.sei.cmu.edu/asset_files/specialreport/2007_003_001_14801.pdf

Holzmann, Gerard, 2012, *Mars Code*, https://www.usenix.org/conference/hotdep12/workshop-program/presentation/Holzmann . Holzman tells about the quality processes implemented in his team for the Mars Rover. Some notable points are the selection of process rules based on known failures, creating a tight connection between bugs and cost. Also, everyone had to comply with the agreed-upon processes and problematic modules were made publicly visible.

Intel, *Report on process improvement project*, 2010. Implementing requirments management, unit tests, continuous integration, design and code review regime, static analysis, exit criteria for each development step. Some of the results: Only one customer-reported defect (the product used in 100 designs, by more than 70 customers); defects/KLOC: 5x below target; estimated $1.6 million saved by reviews; porting to the new version by customers done in less than 1 week; no need for post released sustaining effort (no hot-fixes needed).

Liker, Jeffrey K, 2004, *The Toyota Way,* McGraw-Hill.

# Appendix A

A very partial list of quality processes implemented in fabs

- Incoming material inspections: Any material entering the manufacturing line is tested for purity and cleanliness
- Correlation tests: Process reference wafers through a machine and compare the results to the expected ones. Continue to use the machine only if it passes the correlation test
- Track trends of hundreds of parameters. Use Statistical Process Control (SPC) methods to identify drifts.
- Setting the Control Limits for the SPC is in itself governed by strict rules and procedures.
- Track Quality results indicators such as yield; number of re-processed wafers; machine Mean-time-between-failure
- Training and certification tracking for each individual
- Change Management Process: a list of procedures define how a change is proposed, reviewed, tested and implemented
- Recipe control: a recipe defines the setting of production machines. A host of interlocks ensure the correct recipe is loaded to a machine to process a certain wafer-lot
- Clean Room behavior rules: Dress procedures; allowed materials; allowed behavior in the cleanroom
- Safety procedures: What to do in case of emergency. The fab considers even a simple water spill as a safety event that needs to be addressed according to predefined procedures
- Material disposition rules: How to quarantine and how to release from quarantine any wafers that did not go through the standard process (due to mistake, machine problem etc.).

# Appendix B

Drawing parallels between software development processes and fab processes

| **SW Development processes** | **Fab processes** |
|---|---|
| Requirements: Define the inputs and outputs of the system. | Process targets: Define the exact values of input parameters (e.g. as gas pressure, chemicals content) and the output targets (e.g. layer thickness; resistivity) for each process step. |
| Architecture specs: Define the various pieces of the software that together achieve a Business goal | Process flow: Define the order in which material moves from one processing step to another, to achieve a complete product |
| Design documents: Define the mechanics of each module that will perform a required data processing | Machine recipe: Define how exactly the machine is set up for processing of wafers |
| Reviews, Change Requests, Design Change Notification (DCN) | Change Control Board: Review, critic, improve and approve suggested process changes |
| Unit tests, Static analysis | In-line tests & monitors: Tests done inline on in-process wafers and on machines, before the final product is completed. |
| Configuration management: A system that ensures developers use the correct version of files. | Recipe control: A system that ensures machines are using the correct settings for processing wafers |
| Build automation & Continuous Integration (CI): A system that ensures fast testing of new code, to ensure smooth flow when new code is added | Factory automation: A set of automation control programs that ensure material is moving into the right machines at the right time, with minimal time lost in waiting for free machines. |
| Test and Test documents | End of Line Tests: Electrical tests to validate the end product is performing its intended function |
| Risk Based Analysis | Test sample plan: Definition of the frequency in which in-line tests are done. Testing too often holds the line and is costly; not enough testing risks losing a lot of material when there is an excursion |
| Root-cause analysis | Root-cause analysis |

# Low-Tech ATDD

**Kevin Klinemeier Davisbase, LLC.**

kklinemeier@gmail.com

## Abstract

"The customer does not like what we made this week" is a bigger problem than "The customer does not like what we made this morning". But the natural inclination of programmers is to program until they are out of one of the following: time, features, or food & water. Much of the Acceptance Test Driven Development (ATDD) literature talks about tools for automation, for example Cucumber, Robot Framework, or RSpec. These tools can be a means to an end, but are not required and may be more of a hindrance than help especially when a team is just starting ATDD.

Instead of tools, focus on planning and task creation. Turn the planning of work items (stories) upside down by asking "where is the risk?" rather than "what needs to be built?" A set of standard starting questions that borrow from the traditional QA skillset of Risk Analysis gets the conversation started. These answers lead the planning of the work, starting from the areas with the most risk. Rather than build the feature in its entirety, programmers build just enough to test, based on this plan. No tools are needed, but sticky notes help.

In our team, identifying these testable subsets of work in advance allowed the testers to receive work every day, shortening the bug's life cycle and reducing the cost of their fix. It also allowed the team as a whole to demonstrate to the business approximately every other day, with similar benefits. When the team then adopted automated testing, this experience helped them identify where it would be most beneficial. This experience also translated to better estimates on the part of the testing work, as the habit of discussing risk and mitigation was carried to those activities.

This low-tech ATDD approach is especially applicable to two kinds of transitions a team may be in. The first are those who are taking the first steps to moving away from a big-handoff approach and toward a continuous testing approach. The second are teams that are attempting to work more closely with their business stakeholders. Generally, this approach is useable for both legacy projects as well as new-development. It is appropriate in both the Scrum and Lean paradigms, making no assumptions about when the planning takes place.

## *Biography*

*Kevin Klinemeier has almost 20 years of software development experience as developer, team lead, software architect, and agile technical consultant. As a consultant, he has used this approach with teams as small as three to as large as twenty. He has worked in industries including telecommunications, global logistics, healthcare, and finance.*

# 1 Introduction - IKIWISI and being Driven

The most exciting aspect of the creation of software is also its most difficult: Every time we set out to write software, we are solving a problem that's never been solved before. Because if there's a solution out there that already works, we will download it, copy it, license it, and use it.

Traditionally, we ask our customer "what do you want?" This may have different levels of formality, from simple descriptions, whiteboard mock-ups, or formal documents. All these paths lead to the same universal experience. When shown the result, our customer says, "Ah, now that I see it..." A common complaint among development team members is, "Our customer doesn't know what they want, but they know what they don't want when they see it."

It is tempting to respond to this uncertainty with even more detailed up-front analysis in an attempt to lock down the specification. However, it is not analysis that will help the customer discover what they really need, it is experience using the real, working software. They are essentially saying, because they are business or problem experts rather than software experts, I don't know what I want, but "I'll Know It When I See It" (IKIWISI)

Often this experience can be expressed as tests. If features are too uncertain to drive our work, then it can be tests. Tests now are the primary focus of planning sessions and status meetings. In our planning sessions we discuss which programming activities are necessary to complete a test, and in our status meetings ask, "Which tests have been completed?"

# 2 User Stories and Acceptance Criteria

Agile approaches to the problem of requirements with user stories and acceptance tests (Mike Cohn 2004 loc 378, 560, 1686). Instead of asking non-experts to describe the end state in detail, it asks them to describe what it is the user needs to do in customer terms. Often this takes the form of a story statement:

> As a <role>
> I want to <cause some effect>
> So that <some value to the user or business>

An example:

> **As an instructor,**
> **I want to post my slides online**
> **So that I do not have to email them to my students.**

Acceptance criteria build on this story statement to describe outcomes that are important to the customer, or potentially unexpected to the implementer. These, too, are written from the customer's point of view but are broader. Continuing the instructor example:

1. Uploading the slides should be a one-step process.
2. Only the students signed up for my class should have access to the slides.
3. Only the instructor should be able to post slides.
4. Only PowerPoint files (.ppt) should be accepted, in order to prevent me from uploading the wrong file.

# 3 The Goal: Feedback at the Right Time

If we cannot predict what we want the software to do, our best course of action is to make the software easy to change. There are volumes on approaches and activities to do this from a software design point of view, such as Object Oriented practices including Interface Segregation and Dependency Injection (Robert C Martin 2009), methodology steps like Continuous Integration (Paul M. Duvall, Steve Matyas, Andrew Glover 2007), and of course the creation of small unit level tests in any language you might choose.

From a quality assurance process point of view, this is accomplished by providing feedback at exactly the right time. That right time is immediately after the programmers have made the change (due to bug or requirement change) but before they have built anything on top of it or forgotten how the work was done.

When we find changes, we want to avoid hearing, “That is an architectural change” or “What idiot wrote this code, and what liar put my name on it?” Instead, we want to hear “I can change that, I just wrote it.” or “We can change that, we have not started on it yet.”

# 4 Plan Differently: *ATDD* starts with *ATDP*

The pressure is on the Quality Assurance organization to demonstrate early and provide that feedback. But small demonstrable pieces of work are not an artifact of a QA plan alone. The *writing* of software must be planned differently. The natural inclination of developers is to build everything that is requested, assuming that “the spec” is correct and that they have not made mistakes.

Therefore, if you want Acceptance Test Driven Development, you must first start with Acceptance Test Driven Planning (ATDP).

ATDP activities occur during the planning phase of agile processes. In Scrum, this is the Sprint Planning ceremony. In Lean, it is likely the planning column on your Kanban board. Also, just as those processes recommend, ATDP is best as a collaboration between all the roles involved, especially developers and testers.

Typically in a task creation session without ATDP developers are in the lead. They break the work down into client, service, data, and jargon layers, and commence discussing which flavor of jargon will be best. This has two outcomes: the tests are an afterthought, and any non-programming testers are left out of the process.

Instead, in an ATDP session, teams identify the tests that they will write first, before any development tasks have been created. There is no need to write the entire test in detail, a title is enough. And as the name implies, the user story approach gives you a starting place: the acceptance criteria.

# 5 Acceptance Criteria to Tests

If the acceptance criteria from the story are directly translatable to small tests, great! Count yourself lucky and move on. Most of the time there is some problem in the transition from criteria to test. Good tests are both informative and small.

## 5.1 Make Tests Informative

Acceptance tests’ primary purposes are to allow the product owner to form opinions about the feature being developed.

This means that the functionality under test should be complete enough to include a result or outcome that is at the customer’s level -- for non-technical customers, this will typically be a user interface. Unless a team’s product owner is comfortable reading XML, that outcome would be unacceptable.

Requiring tests to be *informative* requires the implementation to be done in small, vertical slices. Rather than create all of the data structures necessary for the whole feature set, the team creates only the data tables necessary for this acceptance criteria.
This feels inefficient, and programmers are apt to utter the most dangerous words in programming: “While I’m at it, I may as well…” Time spent on code and artifacts not needed to be informative, is potentially

wasted. If the demonstration causes the product to change, all those unrelated items will likely have to change as well.

**Not Informative: Test that confirms a file is stored to the database**
The product owner has no opinions about the database format. Until that data is put into the context in which it will be used, it does not build any new knowledge. Rewrite the test to include the use of some of the information.

**Better solution: Upload the slides and see a confirmation message**
This test helps the product owner determine if the user's experience is correct. They may learn about the kind of confirmation message and details they would like to present. Perhaps the confirmation should include a link to the download, or the name of the class the slides were posted to. Should the filename appear on this page? Great questions arise from this test. Note that this test is specifically about a successful upload. Failure messages are a different test.

## 5.2 Keep It Small: One to Two Days

Being able to form an opinion does not imply that the work is completely finished and polished. It should still be rough -- not production ready. The time taken to make it ready for production will move us away from "I just wrote it" and into territory where programmers say "I have no memory of this code."

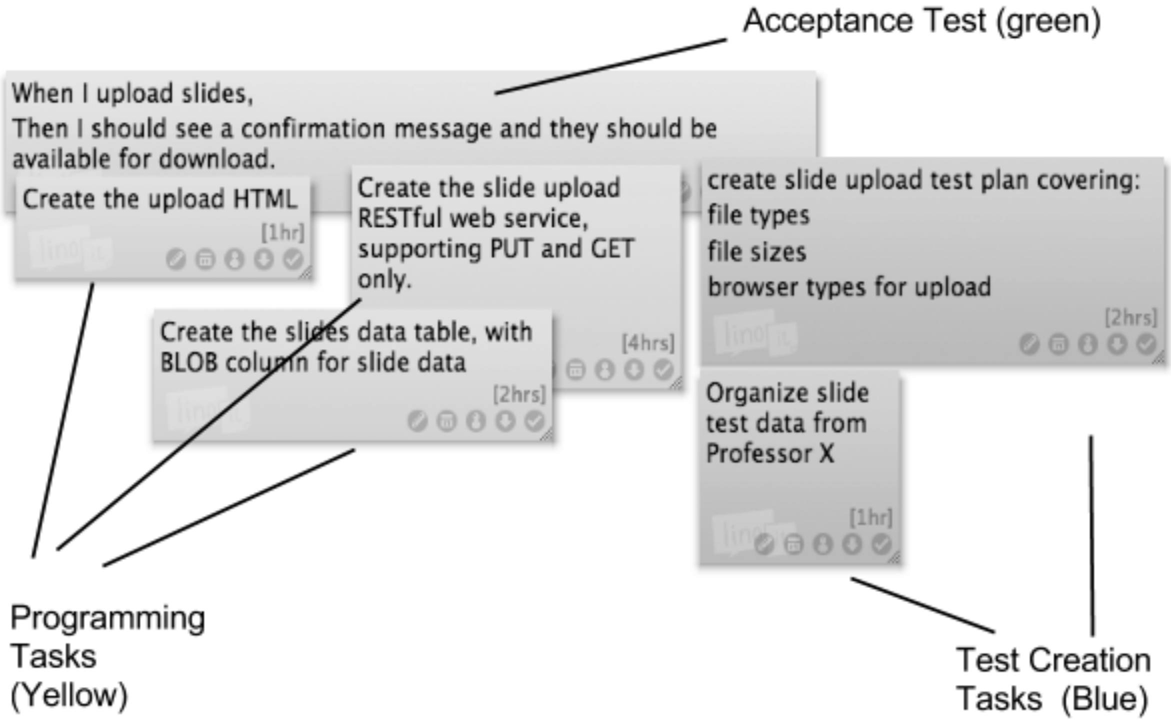
**Not Small:** Instructor can sign up, log in, reset password, and upload slides. There are many features of a login process that are expected in a production-ready product: password resets, multiple attempt lockouts, password rules, etc. If this work is large, such as in a brand new system or complicated enterprise environment, it should be divided into additional tests.

**Better Solution:** Start with one case: "Signed-in instructor can upload slides, students and assistants cannot." Registrations can be simplified, and the work around password resets or lockout durations is moved elsewhere, allowing us to focus on one aspect of the work.

# 6 From Tests to Code

Once you have identified one or more tests, write each one on a card (sticky note, virtual task card -- something visual). Then challenge your development teammates to write only the programming and design tasks that are *absolutely required* to enable that test. Place those tasks visually below the test, marking dependencies physically. Add your testing tasks as well, which can be the creation of detailed test plans, or broader tasks such as exploratory testing.

If you use physical cards, you will end up expressing dependencies very naturally. You physically cannot pick up an underlying task without picking up all the other tasks that need doing first, because the dependent tasks are in the way.



When I upload slides,
Then I should see a confirmation message and they should be available for download.

Create the upload HTML [1hr]

Create the slide upload RESTful web service, supporting PUT and GET only. [4hrs]

Create the slides data table, with BLOB column for slide data [2hrs]

create slide upload test plan covering:
file types
file sizes
browser types for upload [2hrs]

Organize slide test data from Professor X [1hr]

Since each task is estimated in hours, we can calculate how long it will be take to go from start to finish on a single test and evaluate whether we have met the criteria of tests being small. Note: the one to two days goal includes the programming *and* the testing time! In the example above, we have planned a total of ten hours of work, which meets our “small” criteria handily.

If the programming size/effort for your test will take longer than the one to two day guideline, collaborate with your development teammates to break it down into small testable pieces. Here are some options:

- Break the data or interface into separate tests.
  Example: User Setup has 25 fields, start with a test for username, first name, last name, and email.
- Build only the immediate results, implement the delayed results as part of the test for their display.
  Example: In the upload test above, do not store the file -- save that work for the test that displays the stored file instead.
- Break out different outcomes by their results
  Example: build the error-checking as a separate test. Initial implementation for success is to display “error: success not implemented”

# 7 Organization and Prioritization

Great, you have a lot of tests, and work to be done for each one. Where to start? The Product Owner may shrug and say, “It all has to get done.” If we leave it up to the programmers, they will often choose whichever piece has the most interesting coding toys associated with it.

- Likely to change due to uncertainty in our Product Owner or stakeholders.
- Has regulatory or auditing expectations.
- Uses tools or technology we have never used before.
- Integrates with a technology we know to have bugs.
- Integrates with systems we have not integrated with before.
- Likely to have performance problems.

Features with one or more of these attributes are then high candidates for problems during development. For each of these features identify the acceptance test that will test for the related risk. These risk-related acceptance tests should be prioritized first so that we discover any problems as soon as possible. The earlier a problem is discovered, the easier it is to handle.

### 7.1 This is a different role for QA!

This approach changes the role of quality assurance from gatekeeper to communicator -- always searching for how to provide fast feedback. In ATDD, QA leads the team in asking, “How can I show this to someone and find out if it is what we want?”

Many of these testing tasks are not simply for QA to execute, but for QA to facilitate a conversation or demonstration, which in turn leads to more information. Standup and status meetings are good opportunities to ask these pertinent questions:

- Are we working to enable the highest priority test?
- Why are we not working on this task?
- How many demonstrations have we done?
- What percentage of our features have we demonstrated?
- What high priority tests are remaining?

## 8 Making it fun

This approach is compatible with a kind of gamification, or at least a lighthearted approach to execution. Each acceptance test can be treated like achievements in video games. Doing the minimum possible development work to “unlock that achievement” becomes the metaphor for the daily standup.

One example might be that when the Upload test in our example above is accepted by our product owner, we could put a disk-drive sticker on the user story card, or a shield sticker when the security test is accepted.

## 9 Tracking the plan

During the execution of the sprint, testing and development tasks can be used to track progress in the usual way (burndown charts or hours remaining).

In addition, two major metrics can be visible:

- Burndown of acceptance tests remaining
- Number of days since a new test was unlocked

Let’s look at what each item may be communicating, what good and bad looks like, and how to correct our course.

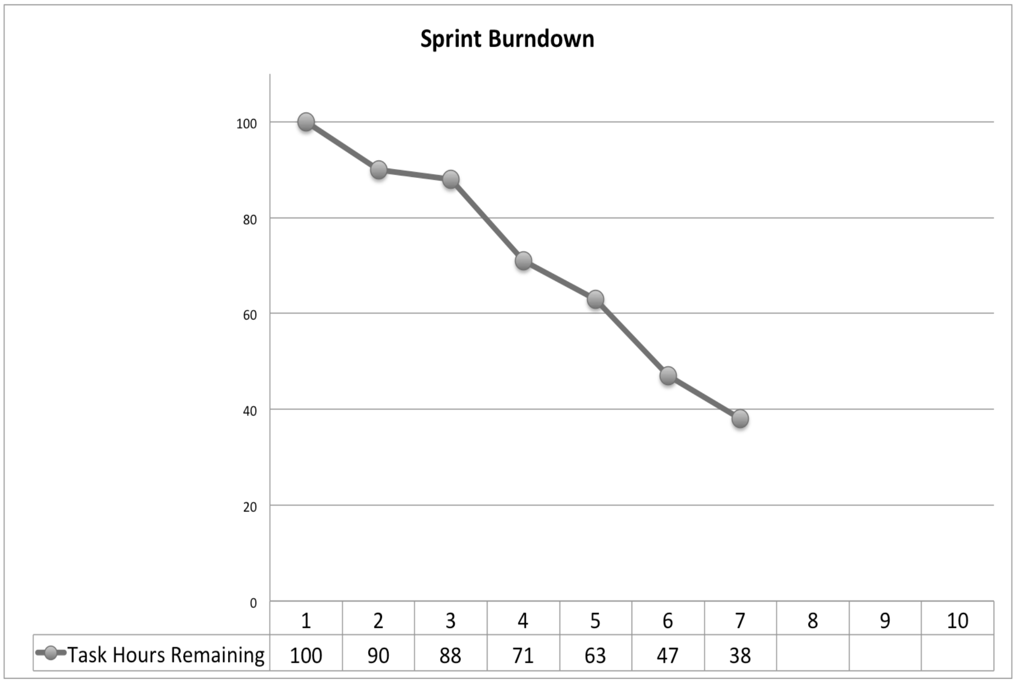
### 9.1 Burndown of acceptance tests remaining

This burndown helps us answer the big question: are we going to finish? Unlike task-level burndowns, simply finishing easy tasks across many different stories does not show as progress. Teams that struggle with story-level burndown charts not being granular enough often find that this measure of sprint completion is very accurate.

This metric, much like a regular burndown, should be updated daily, and reviewed (at least implicitly) as part of the daily standup. Good Scrum Masters and coaches will be sure to point out when our burndown indicates that there is a problem, and translate that into questions for the team during the post-standup “parking lot” session. In this case, the questions are the same as any other burndown projection:

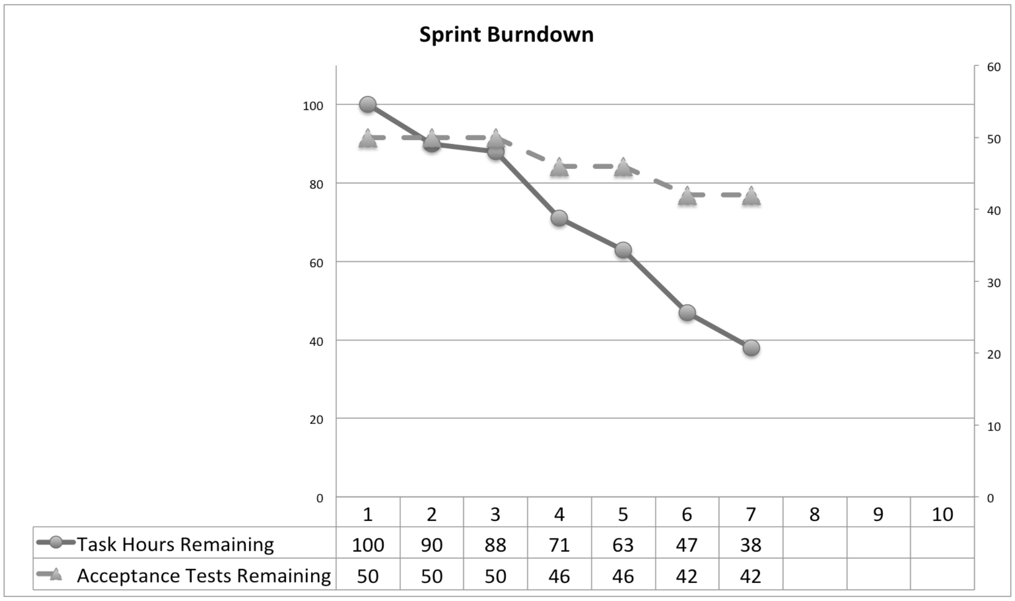
- What are our impediments?
- Can we be sure to finish the most important items?
- Can anything be negotiated out of scope?
- If we do not finish everything, have we reviewed the new scope with our stakeholders?

Let's look at an example.  Consider the burndown example below, which represents a team entering the eighth day of their sprint:



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Task Hours Remaining | 100 | 90 | 88 | 71 | 63 | 47 | 38 | | | |

This team looks pretty good -- on track to complete all the tasks. But completing the tasks is not the same as making customers satisfied. Let's look at an example that also tracks Acceptance Tests Remaining.



| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Task Hours Remaining | 100 | 90 | 88 | 71 | 63 | 47 | 38 | | | |
| Acceptance Tests Remaining | 50 | 50 | 50 | 46 | 46 | 42 | 42 | | | |

This second chart makes it clear that while this team has completed a lot of work, not much of it has been tested. Perhaps they did not build it in testable pieces or there may be an environmental issue that is keeping them from performing their tests. Whatever the cause the second chart shows that there is a lot of testing that has not been done. It will need to be done in a rush right at the end of the sprint, leaving very little time for fixing bugs or adjusting things when their customer inevitably says, “Now that I see it, I don’t like it.”

If we were tracking this all through the sprint, our Scrum Master (or other lean leader) should really begin highlighting this potential impediment starting on day four. This would allow the team to have a conversation about how to adjust the plan in order to execute some of these tests earlier, getting the feedback earlier when changes are easier to make.

### 9.2 Number of Tests In-Progress

This is a measure of those tests that have no dependencies and can be executed, but have not yet moved into the success state. This metric helps teams discover when the bottleneck may be in the testing effort itself. If tests are not completed but new tests are being enabled, this implies that it is taking longer and longer to provide feedback and fixes.

This metric can be used within the sprint as a conversation trigger. Track this metric daily during the sprint, and whenever the team has more than three tests in progress, ask them:

- What are the impediments for finishing the highest-priority test?
- How can others help to test? (Programmers not excluded!)
- Who is working on unlocking yet another test? What can they do instead?

### 9.3 Number of days since new test unlocked

This metric helps teams transition from a mindset of a big handoff at the end by reminding them that testing should start as early as the second day of the iteration.
If it has not started by the third day, ask:

- What are the impediments to finishing the work for the highest priority test?
- How can we get more help on the tasks for that highest priority test?
- What is working and testable right now?
- Did we discover that the test is too big? How can we break it up?

This metric can be used as a conversation trigger as above, but it is also useful in the retrospective to identify habits and behavior patterns.

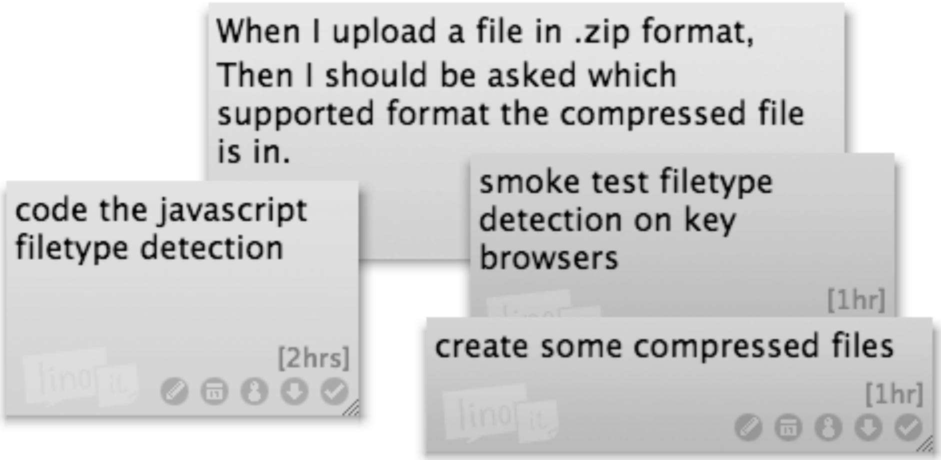
# 10 Changing the plan

Often, the results of a completed acceptance test will be a change to the plan. This change is reflected by changing the acceptance criteria and acceptance tests in the story. That test change is then used by Development to identify what changes are necessary.

Like the initial planning, this is not intended to be a waterfall-style set of handoffs. Teams will find that the acceptance criteria may alter a test so it cannot be completed in a timely fashion. Programmers will propose a modified version of the test. In this way the team optimizes the building of software, the collection of feedback, and the delivery of value.

**Example:**

In our slide sharing user story above, we begin to test the acceptance criteria around uploading only supported file types. We even get to demo the software to a potential user. He uploads a compressed file, in .zip format, and is surprised to see it rejected. The Product Owner asks the team how difficult verifying the contents of the compressed file might be. A conversation about security concerns of expanding files ensues, and the Product Owner accepts a simpler solution that simply asks the user what format (keynote or PowerPoint) compressed files are in. This new acceptance test is added to the board.



When I upload a file in .zip format, Then I should be asked which supported format the compressed file is in.

code the javascript filetype detection

[2hrs]

smoke test filetype detection on key browsers

[1hr]

create some compressed files

[1hr]

The programmers add their coding task (javascript file type detection), and QA adds their detailed testing tasks as well, layering them on top of the main Acceptance Test.

### 10.1 Refactoring

If we change the plan after we have started writing code, then we will necessarily have to change the code. In Waterfall projects, both QA and Business have learned to fear refactoring because changes late in a project are expensive and risky. In ATDD projects, we have addressed the cost and risk by moving the changes from the end of the project to immediately after the code is written. Now our goal is to enable as much refactoring as we can because, when it happens early, it is small in scope and effort.

The practices we have discussed so far support these goals: Write only small pieces of functionality, so that we have little code to change. Execute tests as close in time to code completion as possible, so that we catch faults quickly, and build very little on top of the existing work. This fast feedback approach changes refactoring from fear to necessity.

This can also provide some excellent feedback to teams attempting to take on agile design principles. Want to know if your dependency injection strategy makes code easier to change? Did we segregate the right interfaces? By trying to make changes during the sprint rather than during a maintenance phase, we get fast feedback on the maintainability and extensibility of the design as well.

## 11 What success looks like

One of the most important questions in adopting any kind of process change is, “What does success look like?” An energized, excited team may summarize the myriad benefits of the change into “fixes all problems, and provides running water, Coke, Pepsi, and Spaten in every room.” That is a high bar!

Some post-sprint metrics that may help identify when you are getting the benefits, or guide retrospective conversations on how to change include:

- Number of demos given during a period (higher is better)
- Mean time between starting new tests (lower is better -- we want many small tests).
- Time to first successful acceptance test of a story (lower is better)
- Number of acceptance tests changed during a story (higher is better*)

Counterintuitively, changed tests are generally good. This process captures changes that could only be effectively identified *during* the sprint. When tests change that means the team is applying what they have learned from testing. This metric also exemplifies the need to have open and honest communication.

For teams that adopt this practice as a way to move away from siloed, big-handoff, waterfall development, this is often a hard behavior to change. The team's habitual behavior is often to resist change, or be ashamed of it -- help the team learn to celebrate that change, and the work we did to simplify it. Often, the change is so easy to make that it gets ignored. To keep teams on track and motivated, periodically ask, "how would this change have gone in the old system? When would we have caught that before?"

# 12 The reward for good work is...

As the old saying goes, the reward for good work is more work! Once your team is comfortable using acceptance tests to plan and execute the work, then it is natural to introduce ATDD tools such as Cucumber, Fitnesse, RSpec, etc. Your team will be able to use experience rather than speculation in answering some of these questions that teams ask as they evaluate ATDD tools:

- Are the tests written by non-technical people? (Cucumber does this well)
- Do we have lots of tabular data? (Fitnesse does this well)
- Are our features too different for a standard test template (RSpec has more flexibility)
- Are the tests read by non-technical people? (Trick question – that is what an acceptance test is!)

# 13 Conclusion

It is hard to build software for solutions that have never been solved before, and since the cost of copying data is essentially zero; that is what we are usually asked to do! Respond to the "I Know It When I See It" syndrome by demonstrating working software rather than performing detailed analysis.

Demonstrable software does not come from tools, but from communication. Identifying valuable tests from the start, and developing the software in such a way that those tests are completed early enough to get feedback and make changes will result in higher quality and higher productivity.

Once a team is comfortable using tests to organize their communication within the team and to product owners and stakeholders, ATDD tools can be used to further improve feedback, reduce re-work, share learning, and increase productivity.

# References

Cohn, Mike. 2004. User Stories Applied. Addison-Wesley Professional. Kindle Edition.

Martin, Robert C. 2009. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson Education, Inc.

Paul M. Duvall, Steve Matyas, Andrew Glover. 2007. Continuous Integration: Improving Software Quality and Reducing Risk. Pearson Education, Inc.

Carr, Marvin et al. Taxonomy Based Risk Identification. Technical Report CMU/SEI-93-TR-6, ESC-TR-93-183. June 1993 http://www.sei.cmu.edu/reports/93tr006.pdf

# Quality Data Management

**Christopher W H Davis, Nike Inc.**

em: christopher.w.h.davis@gmail.com; tw: @techchrisdavis

## Abstract

The process of testing code ends up generating a lot of data, but rarely does this data come together in a consistent and insightful way. Normally teams need to focus on making sure immediate deliverables are high quality and they don't have time to figure out metrics by distilling multiple sources. To truly make this data fit into today's agile world the ability to make insightful data the product of a team's work is crucial.

Quality Data Management (QDM) is the art of bringing insights and metrics from data throughout the Quality Management (QM) process back to scrum teams & business owners to show the true picture of quality for the entire Software Development Life Cycle (SDLC). Data points include task management, Kanban/Scrum boards, static code analysis, and automated functional test runs. Making sense of this diverse set of data ends up brining unstructured data analysis back to the QM team.

This session will be to discuss the opportunity of QDM, how to associate these disparate data points, and putting it all together in your automation pipeline.

## Biography

*After over 15 years of experience leading and working on software engineering teams in the finance, travel, and healthcare industry, Chris Davis led the team that delivered the platform behind Nike+, the Nike+ FuelBand, and Nike+ Running. In his current role he is leading the effort to bring world-class QM Automation across all of Nike's consumer digital teams, ensuring quality in and through engineering.*

# 1 Introduction

In today's software delivery models Continuous Integration (CI) is a necessary part of a delivery. Many teams are trying to evolve their CI into Continuous Deployment (CD); which, in a nutshell, is a combination of more advanced building & deployment tools that incorporate integrated automated testing to facilitate faster and more automated delivery. To deliver software faster, the automation of quality checks is critical to be able to keep up with the demands of product owners and consumers in today's rapidly changing technology environments. Consumers expect to see constant upgrades and improvements in software products and without automating much of your delivery it becomes increasingly harder for teams to keep up with demands.

Using today's CI & CD systems throughout the Software Development Lifecycle (SDLC) teams end up generating lots of data that when combined together can give the team great insight into the quality of their processes and products. However because of the distributed and decoupled nature of these systems, as well as the plethora of products that help facilitate all the different phases of the SDLC there isn't a standard for pulling all of this data together or combining it in a meaningful way. This is a significant problem as teams transition to CD, since one of the keys for successful CD is automatically determining the quality of software changes based on the data you have.

Enter Quality Data Management, the ability to manage the data generated through your CI pipeline to determine what the characteristics of code changes are, and if they are good or bad in the context of what you're developing. To understand what we need to do to manage this data the best place to start is by grouping the different systems in the SDLC to figure out what kind of data we can get.

## 1.1 System Components Within a Typical SDLC

The following diagram shows the different phases of the SDLC and what they do:

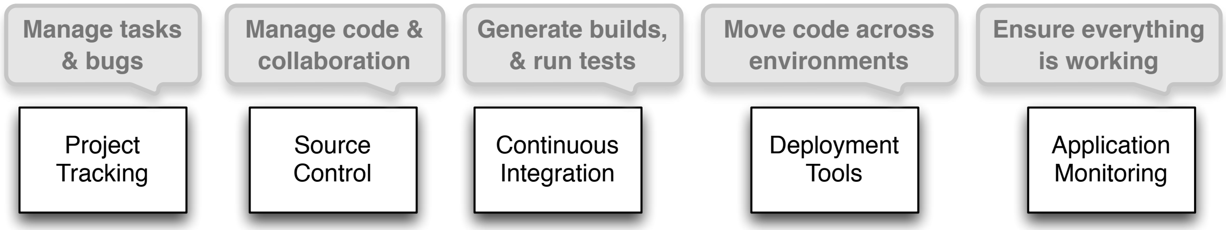

Manage tasks & bugs
Project Tracking
Manage code & collaboration
Source Control
Generate builds, & run tests
Continuous Integration
Move code across environments
Deployment Tools
Ensure everything is working
Application Monitoring

Figure 1: The Different Systems Typically Used Throughout the SDLC

All of these system types have valuable data, that when used together, can help determine the risk of software updates and also objectively determine if a change should be released, or should get sent back to development. Many of the systems in the SDLC can show trends about it's own data but typically can't aggregate all of the data across the complete workflow of software development. To manage all of this data and use it in concert for quality purposes we need a separate system that can aggregate and analyze everything we can get from these source systems.

As with any system you're about to build or buy we need to figure out the key requirements.

## 1.2 Feature Requirements of a QDM System

A system that can give us the level of insight into the software we're analyzing we need to adhere to the following set of requirements.

- Automate data collection from all points in the SDLC
- Allow users to manually tie data together that isn't initially automated
- Store data in a way where it can easily be tied together

- Expose data in a way where it is easily accessed and aggregated for interfaces
- Easily plug in new data sources

By automatically collecting data about the changes you're making to your system, you arm yourself with enough information to accurately point to problems and quantify software quality. Continuous Integration (CI) & Continuous Delivery (CD) systems are automated, so they are perfect candidates from which to automatically collect data. Manually doing this work is cumbersome and costly.

Once you have data you need to analyze it to determine if it's useful or not. Ideally your system can show data in a flexible enough way where users can see trends in graphs and charts to help identify trends. Once we have the ability to see trends we can determine what is good or bad, and ideally use some kind of machine learning to be on the lookout for them later.

Finally systems and technology change. You need to think ahead and assume that any of your systems can get swapped out at any time. From that perspective it should be very easy to plug in new systems or data streams any time there's a new source that could add value to our quality analysis.

# 2 The Data You're Looking For

Some data points to quality in an obvious way, for example, if certain rules are broken or tests fail, you can point to specific things you need to fix to get better quality. However, other things aren't as obvious, for example, code churn or bug counts of non-blocking severities may point to problems with your requirements or development processes rather than the software product itself. In such cases, it's important to look for patterns and collaborate with the team on the larger meaning before jumping to conclusions. Let's take a more detailed look at the systems you need to get data from and what you're looking for from these systems.

## 2.1 Project Tracking Systems

Project tracking systems like JIRA or VersionOne can give you information about what was changed and the amount of churn changes went through in development. This data tends to be relative to the team but can be valuable in assessing the risk of change. For example, in some cases a lot of churn on a particular issue or subsystem can point out poorly understood requirements, changes to requirements, or particular components of your system under development that may require further review or indicate higher risk. Conversely lots of churn could be due to the way the team communicates, and if there is a lot then may be that indicates everyone is participating and there is great understanding. The fact that the same trend can mean something completely different based on the team, makes finding issues from project tracking systems a bit of a gray area. Here, we can go back to the requirement of needing a manual interface with which to identify trends as positive or negative based on how a particular team operates.

## 2.2 Source Control

Source control is where your development team is actively collaborating and changing the product. Since this is where code changes are happening constantly this is a great place to get data on how frequently certain parts of your codebase undergo changes, also known as code churn. On teams using a feature branch workflow where pull requests are required to get changes to consumers you can get data around code churn and code reviews. If a specific part of your codebase is undergoing a lot of churn, this is usually a good indicator that code is going to have a higher risk of malfunctioning than more stable code. Additionally, looking at source control usage as a quality metric can help make the code review process more visible and drive better accountability for individual changes. As churn in project tracking systems pointed to the clarity of requirements, code churn can point to problem spots in specific places in your code that may pose more risk than others.

## 2.3 Static Code Analysis

Using tools like Sonar static code analysis is a simple way to implement basic rule checking into your automated QA process. Static code analysis will check your code against basic rules to make sure you're following best practices and that you're not making any common mistakes like forgetting to handle errors properly or putting variables in the wrong scope. Static analysis can also help you find potential problem spots in your code that may have high cyclomatic complexity or cyclical dependencies. These types of mistakes become more common when a team is put under heavy pressure to deliver. Having a system that can report out rule violations is very useful to ensure that short cuts don't become the norm when deadlines are looming.

Depending on the age of your codebase static code analysis should be used in different ways. For projects that have the opportunity to start from scratch with new code, it's easy to set hard gates and thresholds for the team to follow. For software that's been around for a while it's not so easy; as you likely have many code violations that would take a significant amount of time to track down and fix. In this case, you likely want to analyze new changes to assure that at least code does not get worse, or set thresholds to be able to tackle tech debt as you're delivering new features.

## 2.4 Automated Functional Tests & Production Monitoring

Automated functional tests tell you if your software does what it's supposed to do from the customer's perspective. Realistically these tests should never fail, and if they do a release would likely be blocked unless there are business reasons not to. Whenever you are running automated functional tests your code should be deployed; ideally on a system that replicates your production environment. As these tests run you should be capturing results from the same tools you use to do your production monitoring. You could have 100% of your functional tests pass, but if they generate 1000 errors in your logs with stack traces what the heck are you doing? I have seen this happen when teams swallow errors instead of handling them properly, or when front-end code compensates for a poorly designed or poorly functioning backend by working around its inconsistencies. This is especially common when teams have a large number of functional tests that only test the end result and don't pay attention to what the internal of the system is doing. Ensuring that you're looking at the details of how your system is operating while it's under functional test is a key to getting a quality product out the door.

## 2.5 Combining Data Together

All of these systems on their own tell you lots of good information about the quality of a particular phase of the SDLC, but it's the aggregate of all these systems that truly gives you the full picture of how a team is operating and if the software you're delivering meets the quality requirements of the organization.

Project tracking + Source control = how well your team is communicating & working together.

Static analysis + production monitoring = tell you overall quality of your software.

A team that is functioning at a high level will typically produces better quality, so it's important to look at the combinations of data to get the full picture

# 3 The Quality Data Management (QDM) System

At a high level the concepts we're putting together aren't new, in fact they are a common way to work with large amounts of unstructured data in today's web based world. With the large amount of data that web based systems collect today, management of that data is crucial for the success of these systems. Similarly you can collect large amounts of unstructured data from the various elements of the systems you're using to deliver software and all of those systems can give you insights into the quality of your code and your process. The logical conclusion is to have a system to manage the data can be used in the measurement of that quality; thus a Quality Data Management (QDM) system.

Figure 2 illustrates the types of systems you want to access, and the general flow of a QDM system.

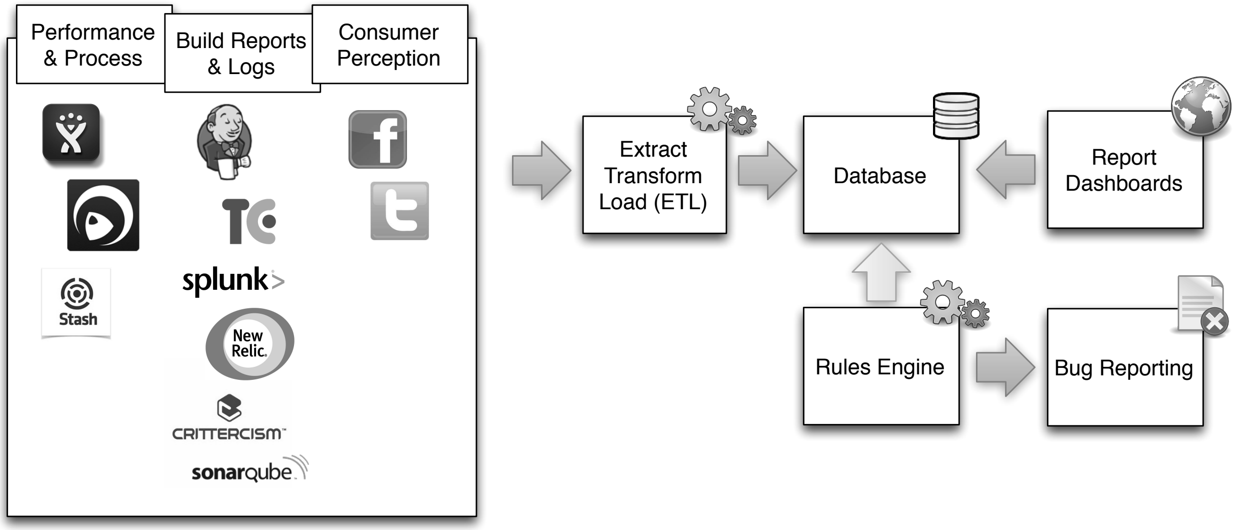


Figure 2 – The High Level System Components

On the left side are the systems you want to collect data from, on the right side is what you're going to do with that data.

## 3.1 Getting Data

Once you've identified all the systems in your SDLC you need to get that data out through some kind of Extract Transform & Load (ETL) process. That will get the data you need a database, since you're combining several different types of data from several different systems a NoSQL document store will be the easiest to work with.

The best course of action is to use tools that your company already supports and knows for the lowest barrier of entry for data analysis and aggregation. However, if you don't have anything you can easily tap into, a simple recommendation for a database would be MongoDB. MongoDB has tools built in to do map reduce functionalities to be able to distill your data easily. It also manages unstructured data very well, is very popular, and even comes preinstalled with some of the more popular versions of Linux.

The easiest way to get data into your database would be to use simple scripts to extract key elements from your target systems and get them into your database. Some popular scripting technologies that make this straightforward are python or groovy. Both have libraries that can be used to get data easily over HTTP and both have libraries to connect to MongoDB fairly seamlessly.

## 3.2 Dashboards & Radiators

Once you have the data you need to display it somehow. Creating dashboards and radiators that are capable of displaying the right data to the right audience is important to validate your quality process. For example, high-level managers likely only care if builds are good enough for production, developers need enough information to find and fix problems with minimal triage, and project managers likely want bug counts and estimated time to fix any issues. The most accessible way to publish the data you need in a flexible way is probably to create a web application that can sit on top of the data you're collecting and display it how you need it.

Putting together a web app is very simple with today's frameworks. If you're using Python or Groovy as mentioned in the last section you're in luck; both have web-scaffolding frameworks (Django for Python and Grails for Groovy) that make displaying the data you're collecting visible over the web relatively easy.

There are also plenty of open source dashboards like Duckboards and Graphite that you can pipe data into. An example of a dashboard that gives you some insight into how your team is working could combine elements of your source control with elements of your product tracking as shown in Figure 3.

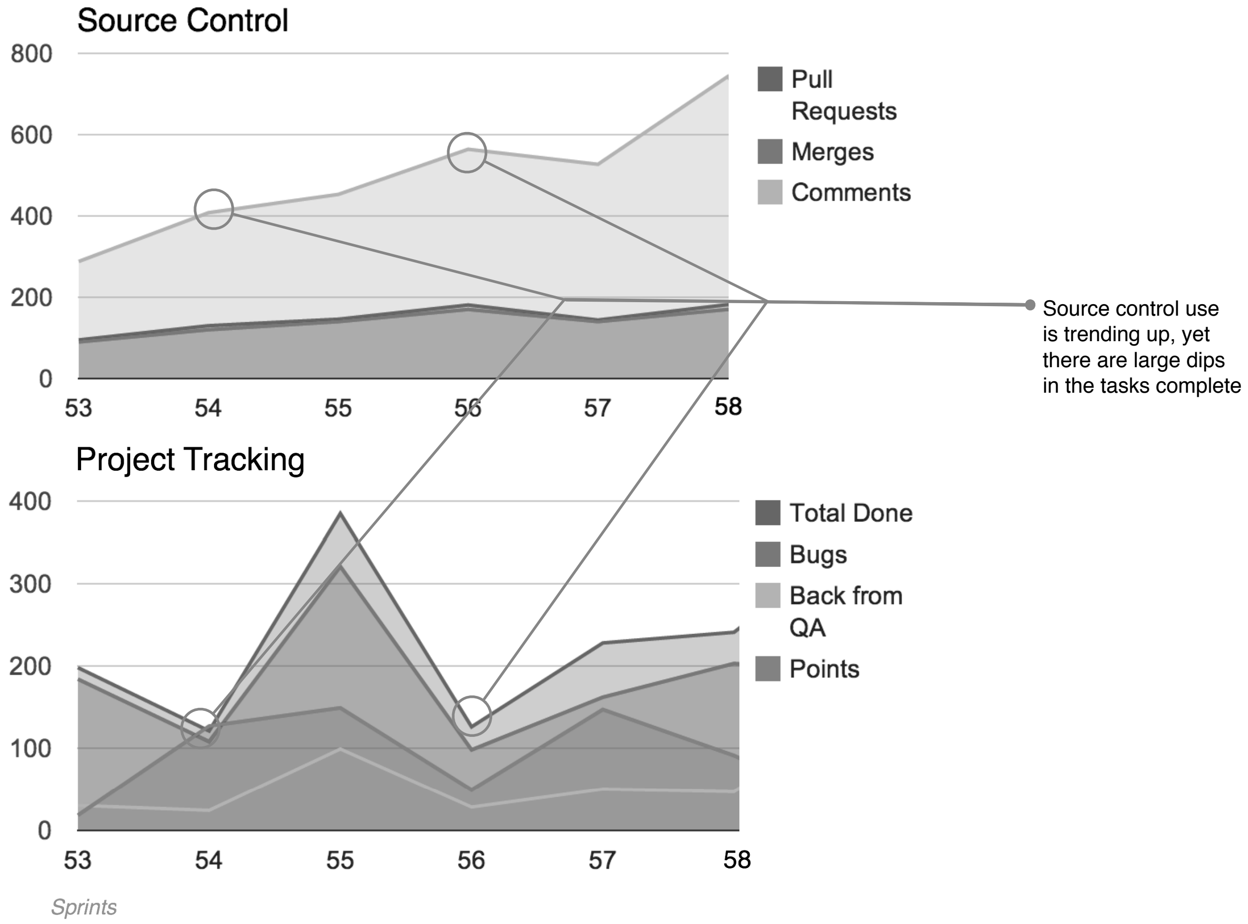


**Figure 3 – Comparison of Source Control and Project Tracking Data**

In this example, we have some key statistics from source control on one graph and key statistics from project tracking on another. The really interesting trend here is the amount of work done doesn't add up to the amount of tasks completed. Notice how the number of tasks is roughly parallel to the number of bugs and not the amount of story points completed. Story Points are supposed to indicate the amount of effort that goes into a sprint, but in this case they don't indicate effort at all. For this team, it seems as if the obvious indicator of effort is source control usage followed by the number of bugs completed. If that's the case then the team needs to re-think how they estimate and plan work. This is a great example of how managing all the data from your project lifecycle can help inform not only quality of your software, but also the quality of the operation of your team.

## 3.3 Data Flow

The QDM system itself needs to be non-intrusive and parallel to the build process, but shouldn't slow it down. Ideally it will get data as code is checked in and build, tests run, and it will act as a gate in the process to stop code from moving where it shouldn't. Figure 4 illustrates the typical grouping of tests in a CI pipeline that will give the system the data it needs to make a very informed decision on whether code should be deployed to the next stage, or if it should get sent back for some rework.

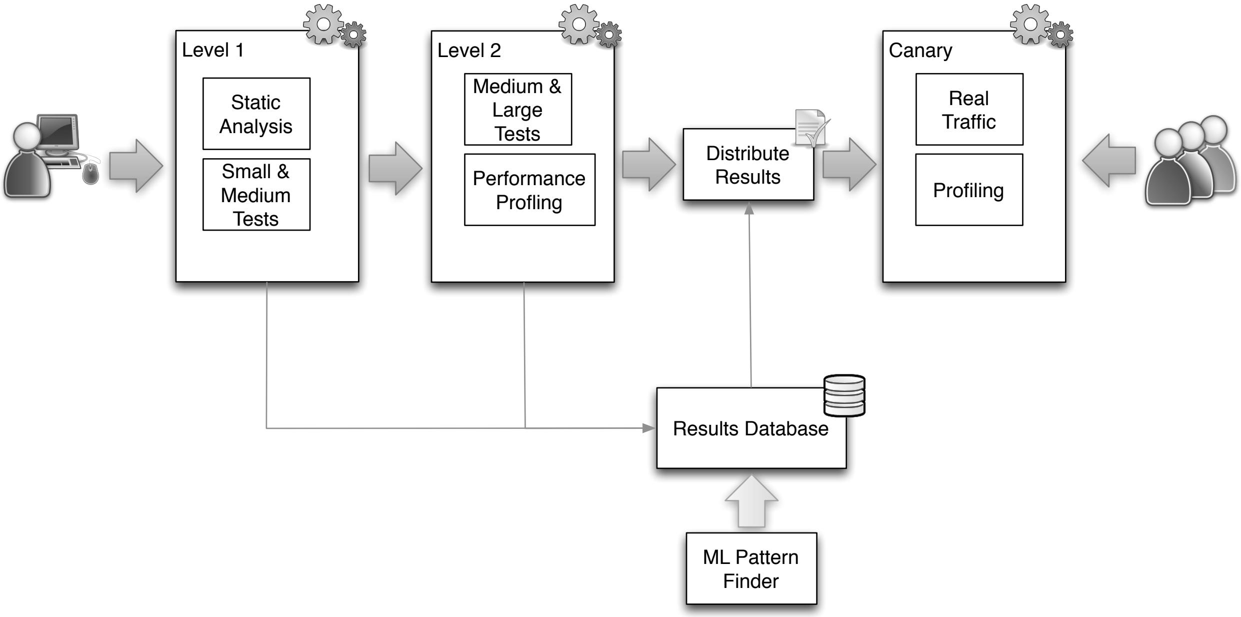


Figure 4– Typical Groupings of Tests in a CI Pipeline

Static analysis will check for code standards, rules, and unit test coverage. This can be used as a hard gate to prevent code without unit tests from slipping through and catch violations before they turn into functional bugs.

The auto-flow is the combination of your functional tests and your production monitoring as mentioned in section 2.4 and shown in Figure 4. In this phase, you can also set hard gates around the data you're getting from your production monitors and the results of your automated functional tests. If you can manage to design your functional tests so that they are also used for performance then that's awesome. If not then you likely need another phase in your build pipeline to get the performance characteristics with enough detail to come up with a quality assessment.

If you have a system where you are actually testing in an environment that has a significant amount of consumers on social media then you can harness that to find out what your consumers are saying about your changes in real time.

The final and most interesting piece of the pipeline is a Machine Learning (ML) component, which can help you identify patterns that you may not be able to identify manually. This is another technology that is becoming more popular and can be integrated with the technologies we've already discussed with minimal effort using several open source libraries available today. Once you start to find patterns that your team determines are important, you can train an ML component in your system by indicating what series of outputs you think are important and the ML component will then try to identify similar patterns in new data.

# 4 Incorporating QDM Into Your Team

The first step is to build a system that can do all this magic for you using modern tools and frameworks that are not hard to deploy. In fact, the easiest problems to solve are technical problems—the hardest problems are behavioral ones. The most important change you need to make on your team to adopt QDM is that everyone must agree to fix problems as soon as the system makes them visible.

Once the system is set up, using these metrics becomes easy. There are a few ways you should incorporate them into your process.

The first key to incorporating QDM is “continuously”.  It’s no accident that the word “continuous” is everywhere today; continuous integration, continuous delivery, continuous testing, continuous choose-your-noun.  The real key to getting better software products out to your consumers as fast as possible is to make everything on your team continuous, including the metrics you use to determine the quality of your software.  This also ends up creating a healthier team, since it makes the QM process extremely transparent and obvious.  By publishing useful dashboards and pushing this data down to the development team as they’re working, you will put the quality criteria in front of the entire team in real time.  The key here is to enforce the rules when things break in real time.

The second is to make the rules you define a part of your delivery process.  When the static code analysis rules you have set up aren’t met – break the build.  When you start to see spikes in the error messages in your logs during a functional test run – fail the build.  Failures need to also be non-negotiable.  Remember the first thing you needed to do was to get agreement from the team that as soon as the system shows something is broken you have to fix it.  With that in mind, if the system gives you false positives then tweak the system.  If the system is giving you good data and you choose to ignore it or minimize it, then you might as well not even bother.

# 5 Conclusion

Using technologies such as map-reduce and machine learning, and harnessing the data that is generated throughout the SDLC you can show objective measures of quality of your software.  By separating the components of your software delivery process you can measure several different aspects, each of which can point to the health of your product and your process, and in most cases can be used as hard gates for your automation process and ultimately can get your team closer to continuous delivery.  Managing the data generated in your quality process can make quality management a more insightful, valuable, transparent, and automated part of your software delivery.

# Building Software That Captures Better Data

**Daniel Casale**

dcasale@gmail.com

**Dan Elbaum**

dan@danelbaum.com

## Abstract

The data that we collect from end-users can be just as valuable as the functionality that we deliver to end-users. But do you think through data requirements as carefully as you think through functional requirements? Given the pressure to deliver quality functionality, it’s easy to forget that data quality is another important dimension of software quality, and that data requirements are a distinct class of requirements worthy of analysis in their own right. With the costs of data collection falling and the power of data analysis tools climbing, getting data requirements right is growing in importance.

So, how do we get data requirements right? How do we identify what data we need to acquire now so that we have the information we need to make good decisions later? How should we store that data in order to ensure that it supports analytics? This paper will describe a few types of data issues and present real-world examples which illustrate their business impact. Our hope is that reading this paper will help you to appreciate how data quality contributes to software quality, and equip you with a set of tools for capturing better data.

## Biography

*Danny Casale is a Database Developer and Data Miner at Portland General Electric in Portland, Oregon. He specializes in information systems and data analytics. Danny has a B.S. in Information Technology from Florida State University and M.S. degrees in Management and Information Systems and Operations Management from the University of Florida.*

*Dan Elbaum works as a Senior IT Business Analyst at Con-Way Enterprise Services where he focuses on mobile applications for truck drivers and freight handlers. He has been specializing in information systems analysis and design since 2007, when he graduated with an M.S. in Management from the University of Florida. He is an IEEE Certified Software Development Professional, and an INCOSE Associate Systems Engineering Professional.*

# 1. Introduction

Requirements elicitation has historically been a process of identifying what services an application should provide rather than what data an application should collect. This bias towards functional requirements made sense in the past when storage capacity and network bandwidth were too limited to collect and store large volumes of data cost-effectively. In that historical context, applications only collected the minimum data necessary to fulfill their functional requirements, and enable user-facing features to work correctly.

In recent years, data requirements have grown in importance. We now have the capability to collect large volumes of data and extract insights in near real-time, using more powerful analytical techniques than were previously available.

The purpose of this paper is to help application analysts and developers identify opportunities to collect useful data which may be valuable to store and analyze. We want to help readers think broadly about decisions that need to be made about data that an application handles.

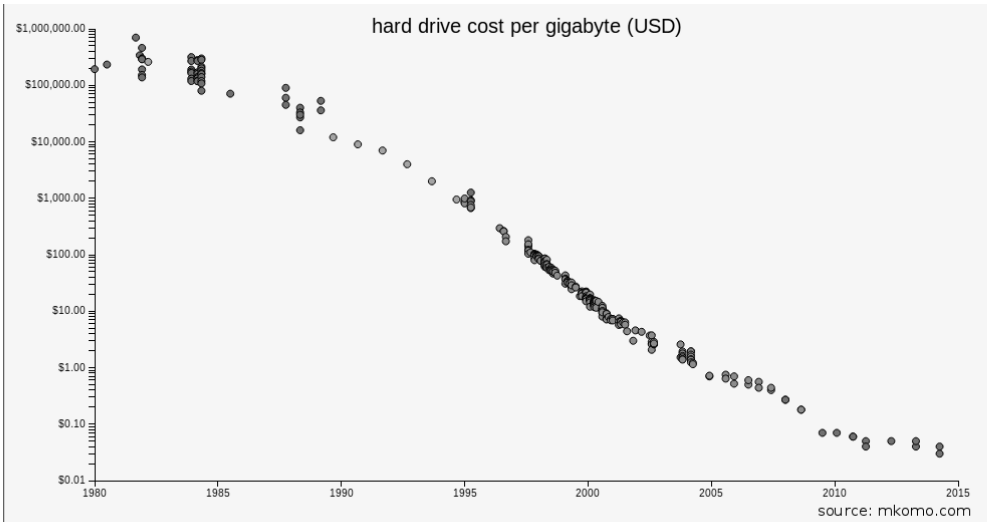
This paper is divided into four sections. The first section reviews recent advances in data storage and analytics which make data more valuable to collect than it was in the past. Section two explains an approach for how to identify what data to collect about each event that occurs within an application. Section three covers how to store the data we've collected. Section four covers how to manage changes to the data after it is already in storage. In each section, we will walk through examples in which we describe a business event, review the possible data that could be captured about it, and discuss the implications of data-related decisions.

# 2. Advances in Data Collection and Analysis Technology

Storage capacity, processing power, network bandwidth, and data analysis tools have all radically improved in performance and plummeted in price over the course of the last decade. These advancements, which we describe in greater detail below, enable the software development community to capture a larger volume of data, and extract more value from it than ever before.

## 2.1 Data Storage and Throughput

The cost of data storage has plummeted in terms of dollars per gigabyte ($/GB) over the last 35 years. Note that the vertical axis in the figure below is on a logarithmic scale.



As of August 2014, Amazon offers storage for 1¢/GB per month (Amazon, 2014).

Data storage technology has improved dramatically because better RAID techniques, hardware and caching systems allow for the I/O and throughput necessary to handle massive-scale applications. The commoditization of 10Gb Ethernet means standard networking equipment has the bandwidth to transfer huge amounts of data. As a result of cheaper storage and processing, database size is no longer the primary limiting factor on system availability and performance in most applications.

### 2.2 Data Analysis and Analytics

The affordability of storage capacity has motivated accelerated development of powerful new tools and techniques for analyzing large datasets. For example, advanced software-enabled analytic techniques like machine learning are gaining popularity, and in many instances partially replacing or automating previously manual statistical analysis techniques. Historically, organizations seeking to predict future events would hire statisticians to apply inferential statistics (analyzing a representative sample and extrapolating to the greater population). This labor-intensive technique required high-quality data, which is rare in most organizations and susceptible to sampling bias. Now, we can supplement traditional statistics with new computational techniques such as machine learning and run analytics on entire populations rather than just representative samples of a population.

Machine learning algorithms have the capability to process billions of rows of data in real-time, continually learn about the anomalies in the data, and recursively improve themselves to account for them. This trend of analyzing entire datasets to overcome data quality issues will continue to rise in popularity as database, storage and computing technology advances.

Taken together, all of these advancements improve the return on investment and time-to-value from data collection. In the next section, we'll introduce some tips for thinking through an application's data requirements so that readers can better acquire the data necessary to harness the power of the new capabilities that we have described in this section.

## 3. Identifying What Data to Collect

So we're building an information system that persists data, how do we determine what data to collect? As a starting point, we should enumerate all events that occur within the application, and then identify every dimension of each event that we could possibly collect data about.

One method for identifying the dimensions of an event is to conduct a "6Ws" analysis:

**W**ho is involved?

**W**hat did they do – To what is it done?

**W**hen did it happen?

**W**here did it take place?

**W**hy did it happen?

Ho**W** did it happen – in what manner?

(Corr, L., & Stagnitto, J., 2012)*

**(Corr and Stagnitto define the 7Ws analysis, this paper only uses 6)*

In the sections that follow below, we will walk the reader through some example situations, and apply the 6Ws analysis to an event in order to show how we explore the issue of what data to collect.

### 3.1 Example: Customer Payments

Companies usually accept customer payments through multiple channels, such as web, phone, mail, or in-person at a physical office. In one problematic instance, we've observed a system that failed to record which channel each payment was made through.

When the company's physical offices were up for renewal, the company was faced with the decision of whether to renew the lease for the existing offices, or rent in different locations. In order to inform this decision, the company would have liked to explore data on which offices were being used by which customers and how.

Unfortunately, the system didn't collect data about which channel payments were made over, much less which physical office they were made at. The problem here was that the data model didn't include an attribute called something to the effect of "PaymentSource" that would serve to identify the Office, or more generally the channel that the payment was made through.

Though this data element was not necessary to fulfill the functional requirement of processing customer payments, the omission of this data element limited the company's ability to make an informed decision about where to rent their community offices.

#### Analyzing the Example

Had a 6Ws analysis been conducted on the customer payment event before the system was built, someone may have recognized that the system failed to collect data to answer the 'Where' question.

Below is a reasonable 6Ws analysis of the customer payments event.

- Who: Which customer submitted payment? Which customer service representative serviced the transaction, if any?
- What: What was the dollar amount of the payment?
- When: When did the customer make the payment?
- Where: Where was the payment made? Was it made in a physical office? If it was made online or through a mobile device, can we collect any geo-location information?
- Why: Why did the customer submit a payment? What services was the payment applied towards? Was it applied to specific charges, or to a general account with multiple charges?
- How: How did the customer pay? Did they pay with credit, debit, cash, or e-check? Did they submit payment using a mobile phone? Did they pay the bill all-at-once, or through installments?

### 3.2 Alerts

Businesses sometimes send "alerts" to notify customers of their upcoming bills, mobile data usage, or electricity consumption. Understanding how alerts influence customer behavior is valuable because it allows companies to continuously improve their customer interactions, and better model and predict customer behavior.

We have seen an implementation of alerts that collected insufficient data for analysis, and precluded continuous improvement of the system. The system at hand stored a user's enrollment status in alerts (subscribed/unsubscribed), but did not maintain a log of all alerts that had been sent. It simply kept a monthly count of how many alerts were sent out to the entire customer population, how many customers subscribed during the month, and how many cancelled their subscription during the month.

This implementation was problematic because a large volume of customers were unsubscribing from alerts, and the business didn't have data to help them decide what approach they could adopt in order to reduce the attrition rate. They had some hunches that the alerts may've been delivered too frequently, or at not the best times, but they didn't have any data to verify those hypotheses.

**Analyzing the Example**

Had an analyst conducted the 6Ws analysis on the alert event before building the system, they may have recognized that the system failed to collect data to answer the ‘when’ question, as well as to collect other data which would’ve helped the company to develop more robust models of customer behavior.

- Who: Which customer received the alert? What time zone do they live in?
- What: Which alert did the customer receive?
- When: At what date and local time did the customer receive the alert?
- Where: Where was the customer when they received the alert?
- Why: What triggered the alert?
- How: Which channel/device did the customer receive the alert on/through? Was the alert delivered via text message, email, or an automated phone call?

# 4 Determining How to Store Data

We will now discuss how to store the identified values of each dimension in a way that makes the data more susceptible to analysis. In this section, we focus on the importance of capturing data at the appropriate level of resolution, and the importance of capturing data in a structured format where possible.

## 4.1 Resolution

Resolution is the granularity of measurement at which a numeric value is stored. Sometimes, categorically new insights can be extracted from granular measurements that are not feasible to obtain from coarse measurements.

### 4.1.1 Example: Electricity Usage Data

Electrical utilities typically record electricity usage by reading customers’ electrical meters and measuring the cumulative number of Kilowatt hours consumed per 15 minute interval. At this resolution, the data does not show how consumption is distributed within each 15 minute interval, nor how electricity usage fluctuates at the sub-kWh resolution. This relatively coarse data resolution was likely adopted before people realized that important insights can be extracted from higher resolution electricity usage data. For example, tools exist that can take as input watt-hour usage data, run analytics, and disaggregate it to determine what types of devices or appliances were likely running, such as heating, cooling, solar, appliances, or malfunctioning equipment. This data can also be used to identify power theft, and forecast load, among other applications.

**Analyzing the Example**

Had we taken the event “Electrical Meter Read” through a 6Ws analysis, we would have identified that we were storing data at an inadequate resolution to fully answer the ‘What’ and “When” questions.

Below is a sample analysis of the event “Electrical Meter Read”.

- Who: Which customer consumed the electricity?
- What: What was measured? (Ideally this would have been watt-hours.)
- When: When was the meter read? For what time interval did we receive usage data? How frequently should we record measurements?
- Where: Where was the meter read from, remotely or on-site? Where is the meter located in terms of both street address and latitude/longitude coordinates?
- Why: Why was the meter read? Was it read at a regularly defined interval, or for another reason?
- How: How was the meter read, by a person physically present on-site or through remote methods?

### 4.1.2 Example: Cross-Channel Analytics

Companies often engage with their customers through a variety of channels such as phone, Interactive Voice Response (IVR), web, storefronts and mobile. The channels which do not require human labor to operate are classified as “self-serve”, and cost substantially less to maintain than non-self-serve channels. The cost of self-serve channels such as web and automated phone systems typically cost on the order of a few cents per customer engagement, whereas phone or storefronts can range in cost from $10-30 per customer engagement (Voxeo, 2013). Thus, there is a clear business case to influence customers to engage the business through self-serve channels like mobile, IVR and the web. So how do businesses achieve this goal? One way is to apply cross-channel analytics. In cross-channel analytics, businesses look at the channel that each customer starts in, and tracks the customer to see if they fall out of the self-service channel at some point, and into a higher-cost channel. Machine learning and other data-driven techniques can then be applied to learn why customers are falling out of the low-cost channels. This information can then be applied to help companies correct those issues and attract customers back into the self-serve channels.

One mistake that can prohibit cross-channel analytics is storing the timing of customer interactions at too coarse of a resolution. When timestamps are captured at day-level resolution rather than second-level resolution, it’s not possible to determine the sequence in which same-day events occurred. So suppose we know that a customer has engaged the company both through phone and web on a single day to find information – the company would have no way of knowing whether the customer began their interaction with a call, and then visited the company’s website, or vice-versa.

**Analyzing the Example**

- Who: Which customer interacted with us?
- What: What did the customer do to interact with us?
- When: When did the customer interact with us? This should be a date/time to enable event-sequencing across channels.
- Where: Where was the customer when they interacted with us? On a particular website screen? Do we want to record the IP address for geospatial analytics?
- Why: What was the customer trying to do? Were they at a particular step in a sequence, like bill payment?
- How: How did the customer interact with us? Did they land on that webpage by directly accessing the page, clicking on a link in an email, or another pathway? What operating system or browser were they using?

## 4.2 Unstructured Data

This section reviews the importance of collecting data in a structured format rather than through free-form text input fields. When collecting data from users, it can be tempting to take the easy path and allow free-text input instead of doing additional work to figure out a set of pre-determined options for users to select from. While free-from text input fields may be less restrictive for users, and easier to technically implement for developers, they generate data that is more difficult to aggregate and analyze in bulk. The better option from an analytics standpoint is to anticipate what type of feedback to expect from users, and then make users select from one of many exclusive choices so that their feedback can be aggregated and analyzed more easily.

### 4.2.1 Example: Marketing, Unsubscribe

Consider the data that companies attempt to collect when people unsubscribe from their emailing lists. All promotional emails are legally required under the CAN-SPAM act to provide a means of unsubscribing (FindLaw, 2014). Often, companies use an unsubscription confirmation page to collect customer feedback and learn how to improve their communications. This feedback can be collected with a free-text form, a list of mutually exclusive reasons, a list of non-mutually exclusive reasons, or any combination of the above. One common error we’ve seen on unsubscribe pages is that companies ask for free-text commentary about the reason people have unsubscribed.

**Analyzing the Example**

It is advantageous to ask customers to provide feedback by selecting from a pre-determined list of reasons that map to actions that the company can take to rectify the problem that the customer identifies. For example, if the customer says that the email is too frequent, the business can adjust the frequency of emails or allow the customer to select the desired frequency of emails. If many customers are giving feedback to the effect that content is not relevant, perhaps the business should consider offering segment-specific messaging or more targeted offerings. By offering a list of issues, businesses can help customers to identify correctable issues. On the other hand, offering a free-text field may help capture issues the business did not think of.

As a note, we recognize that the availability of natural language processing software is increasing, but most companies do not have the software and required skills to perform this type of semi-automated text analysis.

# 5. Handling Changes to Stored Data

Handling changes to already-stored data within an application is another important opportunity for improving the quality of data captured by software. Changes to data include updates, deletes and inserts. There are two main aspects of data changes to deal with: how to archive previous values, and how to capture attributes of the event that changed those values.

With respect to archiving previous values, the safest option from a data retention perspective is to never delete or overwrite any values, but rather archive outdated record or attribute values in a history table. The never-delete, never-overwrite plan is not practical or necessary in most cases, so a middle-ground must be established on a case-by-case basis.

In addition to the archival of previous values, one should also consider recording attributes of the change event itself, such as when the change happened, who made the change, where the change was made from, why the change was made, or what system the change was made through. For example, if a value is updated, consider whether there may be benefit to not only archiving the previous value, but also storing who updated the value, and why the value was updated.

Although data warehousing can solve some of these issues described above, many applications have no data warehouses, and applications are often initially launched before integration into a data warehouse occurs. Additionally, for certain tables that don’t change often and/or are small, keeping historical records may be more efficient than implementing an ETL/ELT process which pulls that data every minute or attaches triggers to the table.

### 5.1 Example: Company Relationships

One scenario where change data can be important is measuring inter-company relationships that span a period of time. For example Businesses often track prospective customers in a database, and those customers are often businesses. Many businesses are interlinked in a family of related business entities, such as a conglomerate consisting of multiple subsidiaries. Each business entity could be acquired, sold, consolidated or closed. How do we handle these events from a data perspective? Should we store only the current relationship status and the date it came into effect, or the start/end date of the current relationship and all prior relationships with their corresponding dates. Such historical records may not be necessary to fulfill the functional requirements of a sales application, but they may be necessary to produce a report of, for example, how customer payments roll-up into parent companies which may have re-structured over time.

**Analyzing the Example**

When a change occurs in a parent-subsidiary relationship between companies, what information might be valuable to capture?

Below is a reasonable 6Ws analysis of this scenario.

- Who: Which companies participated in the relationship?
- What: What type of relationship was created between the companies?
- When: When did the relationship begin and end? Do we need just date or date and time. Do we need to allow for back-dating and future-dating for accuracy reasons?
- Where: Where is the relationship valid or where was the relationship formed? Is the relationship only valid between certain countries or regions?
- Why: Why was the relationship formed?
- How: How was the relationship created? For example, was it through a merger or acquisition?

# 6. Conclusions

Data requirements are an important aspect of software requirements to consider when planning new applications. In thinking about data requirements, we should focus not only on what data must be collected in order to fulfill functional requirements, but also what data might be valuable to analyze in order to extract insights. In thinking through data requirements, we recommend that readers think through each dimension of an event that data could possibly be captured about, how to store that data, and how to capture changes to that data after it is already in storage.

The reader is cautioned, however, that this paper merely outlines a set of tips for thinking through data requirements, and not an exhaustive, holistic approach. There are many other factors worth considering in evaluating how to capture data with software applications.

# References

1. Corr, L., & Stagnitto, J. (2012). *Agile data warehouse design: Collaborative dimensional modeling, from whiteboard to star schema* (Revised/Expanded ed., p. 10). Leeds: Decisionone Press.

2. Komorowski, M. (2014, March 9). A history of storage cost (update). Retrieved August 15, 2014, from http://www.mkomo.com/cost-per-gigabyte-update

3. Amazon (2014). Simple Storage Service. Retrieved August 15, 2014, from http://aws.amazon.com/s3/pricing

4. The CAN-SPAM Act: Trying to Protect Consumers From Unsolicited Commercial E-Mail - FindLaw. (n.d.). Retrieved August 16, 2014, from http://consumer.findlaw.com/online-scams/the-can-spam-act-trying-to-protect-consumers-from-unsolicited.html

5. The Power of Personalization: Optimizing Customer Self-Service for Increased Loyalty and Cost Savings. (2013, June 12). Retrieved August 15, 2014, from http://www.voxeo.com/pdf/Personalization_Whitepaper_June12.pdf

# How Do I Get Started with Mobile Testing?

**Theodore Chan**

TheodoreEChan@gmail.com

## Abstract

With over 60% of U.S. mobile subscribers using smartphones, writing and testing high quality mobile applications that provide user value is critical. Users will uninstall your app if it's not delivering a great user experience especially if it has crashes, bugs, performs poorly, drains the battery excessively or takes too long to start..

This paper is aimed at the beginner mobile tester to assist you in developing a mobile testing strategy; implement a testing process; and iterating to a testing system with a mixture of manual and automated testing for iOS and Android. This paper will guide you as your begin your mobile testing adventure, and provide pointers as you progress through your own testing journey.

## Biography

*Theodore Chan is a Software Development Engineering in Test (SDET) – Senior Quality Engineer at Wellero, Inc., in Portland, Oregon where he focuses on Mobile Testing and Mobile Automation. He is passionate about being an advocate for the user, software quality, process improvements and finding creative ways to improve user experiences.*

*Theodore has a B.S. in Computer Engineering from Rose-Hulman Institute of Technology. He is a Certified ScrumMaster (CSM) and Certified Scrum Product Owner (CSPO).*

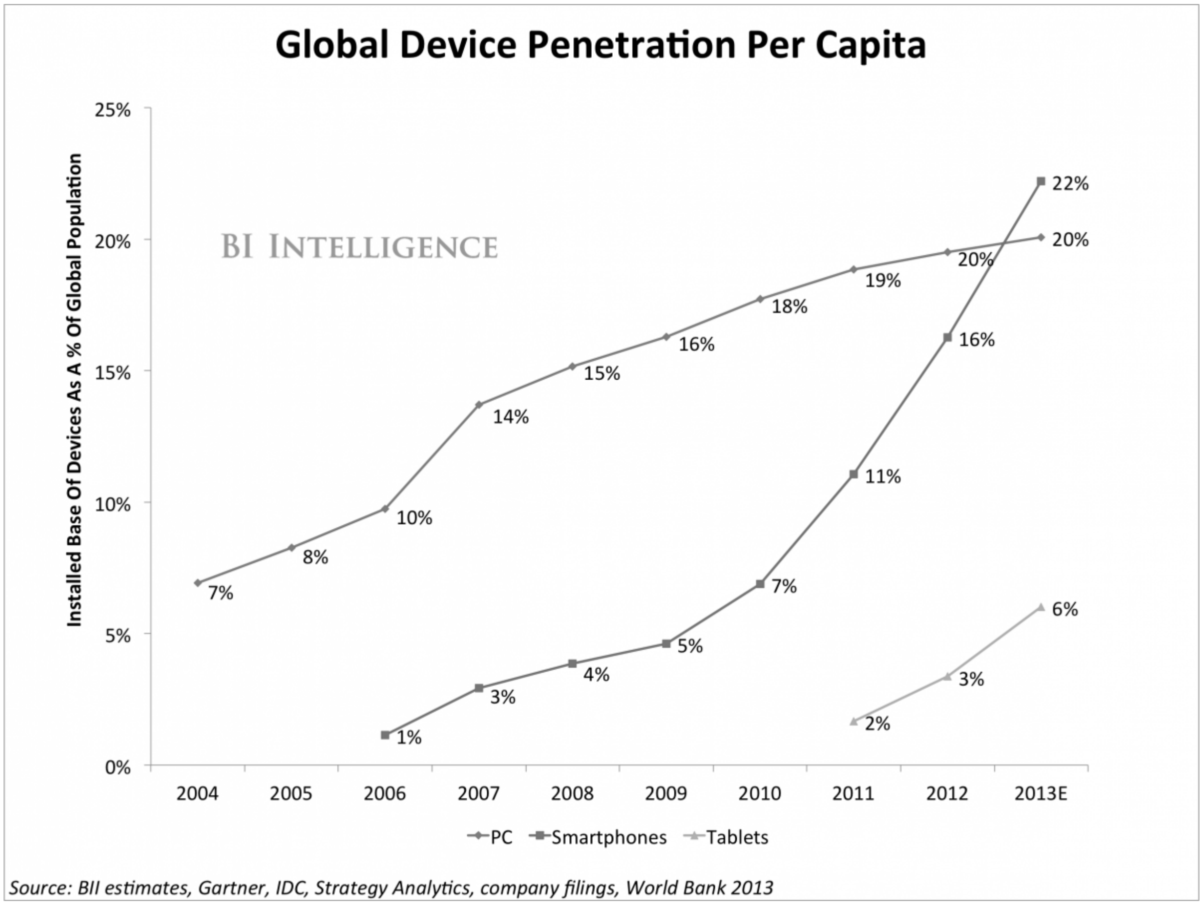
# 1 Introduction

This paper provides the practical tips for beginning with mobile testing and walks through some of the experiences, challenges, pitfalls and successes of mobile application testing. At the end of this paper the testing professional will be able to begin their own mobile testing journey incorporating what they know from traditional testing and applying it to the new and exciting space of mobile testing.

Why should software developers and testers care about mobile?

Mobile Phones, Smartphones, Mobile Apps and Mobile Testing are all hot topics in Software Development. In comScore's Reports January 2014 Subscriber Market Share, 159.8 million people in the United States or 66.8% of the mobile market owned smartphones.

Looking at statistics worldwide, according to a Business Insider's article, "One In Every 5 People In The World Own A Smartphone, One In Every 17 Own A Tablet", by the end of 2013 22% of the global population will own a smartphone. "On average, there will be two smartphones for every nine people on earth, or 1.4 billion smartphones, by the end of 2013." eMarketer projects that by the end of 2014, the global smartphone audience will total 1.75 billion or one in four people of the world population.

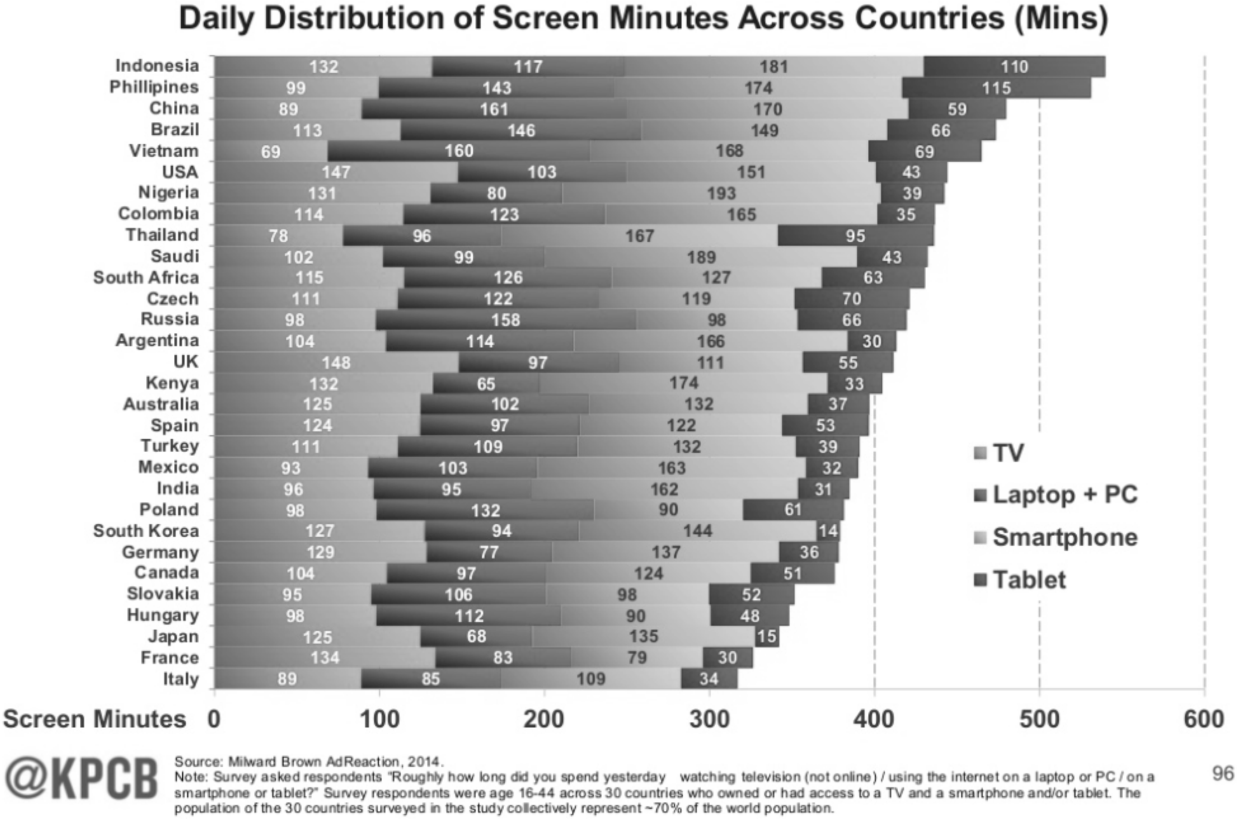


*Source: BII estimates, Gartner, IDC, Strategy Analytics, company filings, World Bank 2013*

http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

Sometime in mid-2012 smartphones surpassed PC ownership worldwide. PC ownership has leveled off, while smartphone and tablets continue exponential growth. For some people, the smartphone may be the only access that they have to the internet.

Daily Distribution of Screen Minutes Across Countries (Mins)



Source: Milward Brown AdReaction, 2014.
Note: Survey asked respondents "Roughly how long did you spend yesterday watching television (not online) / using the internet on a laptop or PC / on a smartphone or tablet?" Survey respondents were age 16-44 across 30 countries who owned or had access to a TV and a smartphone and/or tablet. The population of the 30 countries surveyed in the study collectively represent ~70% of the world population.

http://www.kpcb.com/internet-trends

In Mary Meeker's, "Internet Trends 2014 - Code Conference", Smartphones are a primary screen over TV and Laptop + PC for a high percentage of the countries in world.

Software quality developers and testers are moving to where the users are spending their time and adapting to smartphones and tablets for their applications. They want to meet the needs of their users, adapt their current product lines and build new and innovative solutions for their users.

# 2 Mobile Testing Background

Mobile Testing has many close ties to desktop, traditional web app, and UI testing; however there are many more variables that can throw a wrench into the way we test on mobile platforms that you may not think about when transitioning from web app and UI testing to Mobile Testing.

The goal as a mobile applications tester is to ensure that the software that you and your team are releasing provides value to the end user. Also, the tester wants to minimize or eliminate bugs, crashes, poor performance, poor speed, battery drains, "slow" behavior or long startup times, and complex or confusing UI and UX.

In this paper we will discuss the following:

- Web App Testing vs. Mobile App Testing
- Mobile operating systems
- Differences between native apps, hybrid apps and web apps
- App store submissions
- Testing considerations
- Frequency of use and app value

- Emulation vs. Physical devices
- Hardware testing including
    - Geofencing and Location testing
    - Push notifications
    - Data connections
    - Device level interrupts
    - Multiple apps running
- Beginning Mobile Test Automation

# 3 Web App Testing vs. Mobile App Testing

A Mobile App Tester needs to keep a frame of reference of the user, the context and intent of the user, such as

- When they are using their device
- Where the user is while using their device
- How much time they have
- How focused can they be

The time that a user spends in front of their mobile device can be greater than the amount of time they spend in front of their desktop computers as an aggregate. However, a user typically uses their mobile device in short spurts to accomplish very specific tasks. This would include acquiring relevant information in the shortest amount of time that is relevant to the task they have at hand.

An example of this would be a user checking the weather for the day when they first wake up to see if they need an umbrella, a jacket, or to put on shorts.

Another example would be a user making a reservation at a restaurant with a Restaurant Reservation Application, such as Open Table, sending a message to their friends about the reservation, with WhatsApp, and navigating to the restaurants with a real time update of their location that they share as they drive to the restaurant to meet their friends, with Waze.

Mobile Devices have many built-in inputs, sensors, and outputs. This enables some very creative uses of the mobile devices, but also adds complexity and variables to testing.

### 3.1.1 Web App Testing vs. Mobile App Testing

| | **Web App Testing** | **Mobile App Testing** | |
|---|---|---|---|
| Categories | | **Android** | **iOS** |
| **OS** | Windows, Mac, Linux, Unix, others | Custom overlays on top of AOSP Samsung, LG, Sony, Motorola etc. | iOS |
| **Users Updates to OS** | | Varies by Manufacturer and Carrier | typically 3 months to latest version |
| **Device** | Typically x86, x64 processors | Varies by Manufacturer | Apple iPhone, iPad, iPad Mini, iPod Touch |
| **Inputs** | Keyboard, Mouse, webcam, microphone, fingerprint scanner | touchscreen, soft buttons – (home, back, menu), soft keyboard, front facing camera, back facing camera | touchscreen, soft buttons, soft keyboard, front facing camera, back facing camera |
| **Outputs** | screen, speaker | screen, speakerphone, headphone jack, haptic feedback | screen, speakerphone, headphone jack, haptic feedback |
| **Buttons** | | power button, volume up/down | power button, volume up/down, home button |
| **Sensors** | | light sensor, Accelerometer, Gyroscope, Pedometer, Compass, Hall, Fingerprint ID, Gesture, Barometer, Step detector, Step counter | Three-axis gyro, Accelerometer, Proximity sensor, Ambient light sensor, Fingerprint identity sensor, Home/Touch ID sensor, Backside illumination sensor |
| **Location** | from network/ip address | GPS, A-GPS, Glonass | Assisted GPS and GLONASS, Digital compass, Wi-Fi, Cellular |
| **Connectivity** | Wi-Fi or LAN, Bluetooth | Wi-Fi, Cellular, Bluetooth | Wi-Fi, Cellular, Bluetooth |
| **Network Consistency** | consistent | can be intermittent | can be intermittent |
| **Screen Size** | varies | varies | varies |
| **Screen Resolution** | Varies | Varies | Varies |
| **Battery** | | varies 10 to 20 hours | varies 10 to 20 hours |

Table 1.0 - Web App Testing vs Mobile App Testing Comparison Chart

In the Web App Testing vs. Mobile App Comparison Chart (Table 1.0), mobile devices have sensors such as a GPS, Accelerometer, Barometers and light sensors that a traditional laptop or PC might not include natively. For Android, Google categorizes sensors into three Types of Sensors according to Google's Android Developer overview "Sensors Overview" there are the following types of sensors:

### 3.1.2 Motion sensors

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

### 3.1.3 Environmental sensors

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

#### 3.1.4 Position sensors

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Keeping the types of sensors in mind helps to separate out the testing the tester will need to run, and may influence the decisions on testing with physical devices and/or simulators and emulators that will be discussed later in this paper.

# 4 Mobile Operating Systems

Android and iOS are the predominant operating systems with Android controlling 51.7% and Apple controlling 41.6% of the market share for a total of 93.3% of the US Smartphone subscribers according to “comScore Reports January 2014 US Smartphone Subscriber Market Share”.

Each year Apple (at WWDC) and Google (at Google I/O) have developer conferences where they announce and release changes to the Operating System including adding new API’s, libraries, and/or changes to the SDK. Android and iOS are updated to the general marketplace within a few months of the developer conferences every year.

The changes may be slight updates and tweaks or they may be major updates to everything from graphics, layout, and design concepts to adding access to additional sensors or hardware capabilities.

Major UI changes and requirements to the operating system can result in a major overhaul needed for the development team and their application in order to conform and follow new design guidelines and styling rules. Also, this might mean that API’s that the application is using have changed or become deprecated. Other effects are, existing paradigms of the OS such as navigation within the application may change as well.

For Mobile Testing, one thing is constant, CHANGE! The operating system and different API’s will change and as a mobile application tester knows, they will need to be able to adjust and adapt to new UI designs, different API calls, and possible changes in workflow.

#### 4.1.1 Personal Experience – Transitioning from iOS 6 to 7 and Gingerbread to Jelly Bean

*I have worked through testing of mobile applications from the transition of iOS 5 to iOS 6, and iOS 6 to 7. With Android, there were some changes to the UI and API’s that developers were accessing transitioning from Android Gingerbread to Ice Cream Sandwich; and from Ice Cream Sandwich to Jelly Bean. After each Operating System Update the application that I was working on had to be updated and retested for the older version of the operating system as well as the latest and greatest version of the updated operating system.*

Being cognizant of changes in the UI, UX, how the native applications operate, user expectations and norms is important to testing applications that adhere to design guidelines that make mobile applications intuitive for users.

A software quality tester is the advocate for the applications’ end user. The tester needs to understand what the end user’s expectations of

- How the mobile application should function
- How they will use the application
- What they are trying to accomplish with the mobile application
- Assist the user solving their problems or achieving goals

# 5 Native Apps, Hybrid Apps, Web Apps and Responsive Design

Developers and testers need to take into account if the applications that they will be testing are native apps, hybrid apps or web views of a responsive designed web site. The article “Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options” has describes the differences between Native apps, hybrid apps and web apps as follows:

Native apps are apps that are installed on the device, typically installed through an app store such as Google Play Store or Apple’s App Store; can access all of the devices inputs, sensors and outputs, and has full access to the OS’s API’s. Native apps perform the best and typically confirm to the commonalities of that operating system.

Web Apps are apps that use standard web technologies – typically JavaScript and HTML5. Developers can develop sophisticated apps with HTML5 and JavaScript. However, there are some limitations as the web apps cannot access some of the native device functionality although that is beginning to change as more mobile centric protocols are added to the web stack.

Hybrid apps are apps that embed HTML5 apps inside a thin native container.

| | **Native** | **HTML5** | **Hybrid** |
|---|---|---|---|
| **App Features** | | | |
| Graphics | Native APIs | HTML, Canvas, SVG | HTML, Canvas, SVG |
| Performance | Fast | Slow | Slow |
| Native look and feel | Native | Emulated | Emulated |
| Distribution | Appstore | Web | Appstore |
| **Device Access** | | | |
| Camera | Yes | No | Yes |
| Notifications | Yes | No | Yes |
| Contacts, calendar | Yes | No | Yes |
| Offline storage | Secure file storage | Shared SQL | Secure file system, shared SQL |
| Geolocation | Yes | Yes | Yes |
| **Gestures** | | | |
| Swipe | Yes | Yes | Yes |
| Pinch, spread | Yes | No | Yes |
| **Connectivity** | Online and offline | Mostly online | Online and offline |
| **Development skills** | ObjectiveC, Java | HTML5, CSS, Javascript | HTML5, CSS, Javascript |

https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

The rest of this paper will focus primarily on testing native mobile applications, and some of the key concepts will be applicable to the mobile software quality tester even if they are testing web apps and hybrid apps.

# 6 App Store Submissions

The majority of applications are distributed on the application stores:

App Store for iOS

Play Store for Android

There are several smaller app stores such as Amazon App Store, Samsung App Store or app stores / curated apps that are 'approved' by the cellular networks such as Verizon's Android App Store, or AT&T's Android App Store.

Application Submission

For apps on the Apple App Store, they must be submitted to apple and it can take up to two weeks for an app to be processed and approved before they will appear on the app store. If the developer is releasing an update to the application, the app needs to be re-submitted each time there is an update and uploaded to apple again. Once an application is approved, the developer can hold off on releasing it to the public or set it to publish immediately once it is approved by Apple.

Apps submitted to the Google Play store do not require any wait times. However, Google runs through a battery of automated tests to verify no malicious code or inappropriate API's or permissions are used. Once it is approved, the app can be live on the Google Play Store, or a date can be set for "release" in the future.

Some considerations about testing apps after they have been accepted by the app store:

- Verify that the application can be downloaded from the app store
- Verify that the application be upgraded from a previous version of the app store
- Verify that the application that is on the app store allows registering a new user
- Verify that the application allows a user to login

### 6.1.1 Others' Experience – Checking Production Builds on the App Store

*It's good to verify the above in production to ensure that there were no changes to the code when switching from qa / development branches to putting the application into production. There have been issues with people pointing to a qa or dev endpoint rather than a production end point when publishing their application. Data is added to a development database rather than saved to the production database. Another possibility is a last minute change creeps in after QA has completed testing and the application crashes when trying to register.*

Developing deployment processes, testing and verifying that everything goes to production (in the case of a mobile app, the app store) as expected is critical because otherwise all of the testing and development work will not reach the end user and will not provide value to the business.

# 7 Testing Considerations

## 7.1 Android

Android allows a lot of customization and flexibility. That is one of the beauties of having an operating system that is open source, although a large majority of the code is developed and led by Google. However, the openness also makes it much more challenging for developers and software quality assurance testers.

### 7.1.1.1 Android Overlays

Each manufacturer has a different overlay to the standard open version of Android Open Source Project or AOSP that they may add additional API calls to their specific hardware or implement a couple of things differently.

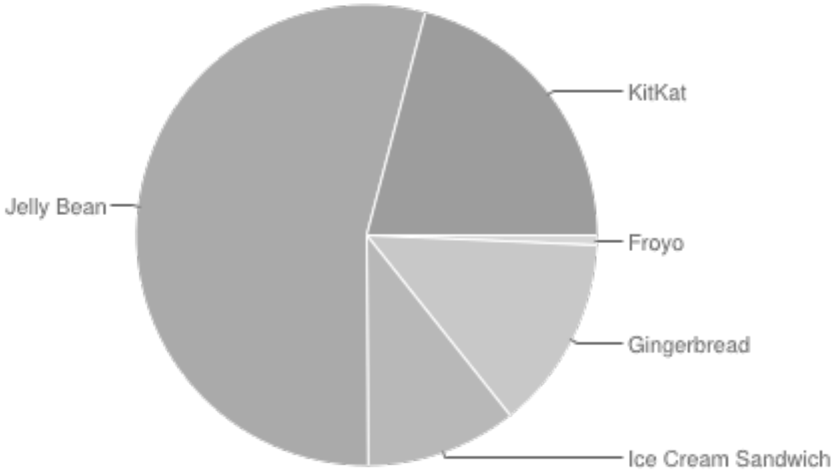
Different types of UI's compared, see PhoneArena.com's post "HTC Sense 6 vs Samsung TouchWiz vs Sony Xperia: UI comparison", to see the graphical differences between HTC, Samsung and Sony's Android Overlay.

| **Manufacturer** | **Launcher/Overlay** |
|---|---|
| Samsung | TouchWiz |
| HTC | Sense |
| Sony | Xperia |
| LG | Optimus UI |

Despite all the talk about how fragmented Android is, it is becoming less of an issue because:

1) Much of the standard API's are being incorporated into the Google Play Services which every device that is running the Google Play store utilizes. See Ron Amadeo, "Balky carriers and slow OEMs step aside: Google is defragging Android", in Ars Technica for more details

2) The Android Dashboard shows that devices using 4.1.x Jelly Bean and above totals 75.1% as of August 20, 2014 (According to the Google Dashboard which is updated every 7 days). If Ice Cream Sandwich version 4.03-4.04 is included in that number, over 85.7% of the android users. There have been some minor changes between Ice Cream Sandwich to Jelly Bean and Kit Kat.



http://developer.android.com/about/dashboards/index.html

For Android there are still a small portion of users 13.6% that are still using Gingerbread, but the majority of users are using Ice Cream Sandwich or above. To help the developer determine what level of API level to utilize, they may want to consider:

Does the application need the service calls that are made available in the latest API level?

Some considerations with testing mobile apps:

- What level of API is the app targeted towards
- Should tablets be considered as a target platform

Physical Device Considerations:

- Older devices that have slower processors, or lower specifications including RAM and on-board lesser amounts of memory
- Smaller sized screens – this will allow you to test your layout and UI
- Larger sized screens – with the advent and popularity of phablets, the tester may want to make sure that the layout can scale, doesn't look awkward on larger sized screens including tablets
- Different manufacturers and their various UI overlays
- Test older operating systems (SDK's on the Simulators) on older devices
- Test the latest operating system with the latest devices
- Physical Keyboards
- Other Built in Accessories specific to the device hardware

Testing devices with physical keyboards, built in accessories like the S-Pen for Samsung note device, or infrared blasters. If your application is highly dependent on text input, it is a good idea to include testing with devices that have physical keyboards. Some enterprise companies have switched from using BlackBerry devices with physical keyboards to android devices with physical keyboards.

### 7.1.1.2 Personal Experience - Android Operating System Testing Jelly Bean vs. Kit Kat

*For the mobile app that I work on, the current minimum target API level is Gingerbread (Android 2.3.3 or API level 10); testing with a Kit Kat device our team was able to locate a defect with how data was handled in an array from a JSON response from the Web Server. The development team had used a Nexus device that was running Android Jelly Bean and did not see any issues. However, testing with a Nexus 5, running Android Kit Kat, the response returned was different with Android Kit Kat. The Android developer was able to iterate through the array of returned data from the web service call and match the relevant response rather than relying on the call to return the items in a certain order which worked fine on the developer's Android Jelly Bean device.*

### 7.1.1.3 Latest Devices

If it is available, and within the company budget, test on the most popular devices for the year, i.e. best sellers and flagship devices because these will be the devices that the majority of the audience will be utilizing the app with most frequently. Most of the time these devices will not have issues with the processing power or memory; the issues will typically be around intricacies in how the overlays, and customized launchers handle the interactions of the application and some device specific hardware that may not be common to all devices. Some examples of newer sensors:

- Heart rate monitor
- Fingerprint ID
- Environmental Sensors
  - Thermometer
  - Relative Humidity
  - Pressure Sensor

### 7.1.1.4 Lower End Devices

It is a good idea to obtain a few older devices for compatibility testing. Testers can acquire devices on the lower end of the spectrum with the minimum API level that the application supports.

Make sure that the devices have a variety of:

- Screen sizes including smaller screens
- Lower RAM specifications
- Slower processors
- Physical keyboards

#### 7.1.1.5 Personal Experience – Variety of Devices

*Early on in testing our application, we obtained a Pantech Pocket. It looks almost square shaped, runs Android Gingerbread 2.3.4, has a resolution of 600 x 800 pixels, has a 1 GHz Qualcomm Snapdragon S2 processor and 512 MB of RAM. It was a good test device, because it had lower specifications, had an oddly shaped screen, and made by a smaller manufacturer.*

#### 7.1.1.6 Alternatives to Building a Device Lab

If the tester's company does not have ability to purchase, rent, or lease some of the latest devices, there are testing services that allow the tester to push their application to a vendors' set of test devices such as AppThwack https://appthwack.com/ or Test Droid http://testdroid.com/ and run tests on their set of hardware.

There are also services that allow running tests on a battery of pre-configured emulated devices such as:

Sauce Labs - https://saucelabs.com/mobile

Perfecto Mobile - http://www.perfectomobile.com/

SOASTA - http://www.soasta.com/

## 7.2 iOS

The variability for iPhones is a little lower compared to Android. However, there are some things that the tester should keep in mind as they begin testing iPhone, iPad and iOS Apps.

Test with larger and smaller screen sizes. This includes testing with iPad's and iPad Mini's to see how your application scales with the 2x button.

### 7.2.1 Personal Experience – iPhone Screen Size

*In testing our application, there were some defects where certain parts of the screen looked fine for the iPhone 5, and 5s; but the keyboard covered a field, or a section at the bottom of the screen that was not visible because of the smaller screen size for the iPhone 4 and 4s. Another example, on certain screens of the application there were a couple of places where the user would need to scroll vertically on an iPhone 4s in order to view the content at the bottom, however the view was not scrollable. Everything was visible testing with an iPhone 5 and 5s.*

If your application supports a minimum level of the OS, test with older versions of the Operating System as well as the latest version to ensure that there weren't any defects related to updates or changes in the OS.

### 7.2.2 Personal Experience – iOS 7 Updates

*In one of the updates to a version of iOS 7, there was a defect from Apple that had to do with the list views that caused a line with pricing information not to be displayed. We found the defect and added a patch to our mobile app. A couple months later Apple updated their defect, and their update broke our patch, so we had to update our code to take out the fix that was added. Staying on top of updates and patches is important for testers.*

iPhone users update their phones when a new release is distributed fairly quickly because it comes directly from Apple. According to Kukil Bora, "Apple iOS 7 Two Months After Release: Adoption Rates Cross 70% Mark, Surpassing Those Of Predecessor's" in IB Times and Chitika's "UPDATE: iOS Version Distribution Study - 2012 through 2013, 3 months after iOS 7 was released 70% of users updated their devices to the latest OS. Some older devices did not meet the minimum requires specifications, and were not able to be updated.

# 8 Application Value and Frequency of Use

Mobile App Testers want to ensure that the user has a good user experience, and will come back to the application the next time they need to interact with or solve the problems that the application assists them in resolving. If the user has a bad experience, they will uninstall, delete and even give the app a poor rating. Continuing to provide value to the user is critical to a good user experience.

- What is the frequency of use?
  - Used Frequently
  - Used Occasionally
- What is the value of the application?
  - Acquire relevant information
  - Provide access to important content
  - Assists in tracking or monitoring some personal data?
- Does it simplify the user's life?
  - Reminders
  - Directions

Some apps are used on a daily or multiple times a day such as social media, news, weather or other timely information apps

Other apps are used a couple of times a month such as banking apps to check on the status of your account or make sure that a payment has been processed.

The apps that are used infrequently need to provide a value to users. Most of the time an app only stays on the smartphone as long as it is providing value. If it stops providing value or crashes the user will delete it. Which apps do testers have that were installed, used once or twice to explore and then immediately deleted, because they were buggy, or did not fulfill the desired need?

If a tester takes a look at the apps on their phone:

- Which ones do they use frequently
- Which ones do they use occasionally
- Which ones only a couple of times a year

The tester wants to ensure that the app is valuable to the end user whether it is used frequently or occasionally. The app must provide some value to the user such as relevant content, entertainment, ability to communicate or share.

- Is your app used a couple times a year, but still has value because it simplifies the life of the user?

### 8.1.1 Personal Experience – Simplifying Life of Users and High Value

*The application that I work on is estimated to be used about 3.4 times a year for the average user. It has a high value because it simplifies the users' life, and assists them in keeping track of their personal health information. The app allows users to easily interact with their personal data, have better conversations with their healthcare providers and can act as a reminder to ask additional questions on the next visit. The app provides enough value that they do not delete the application even though it is not used on a daily basis.*

- Is the UI intuitive enough for the user to come back to, if they use the app infrequently?
- Does it follow the design principles, UI and UX prescribed by Apple or Google?

Mobile app testers want to ensure that the app is easy for users to figure out how to use without complex instructions or tutorials. The application should conform to common design practices, be intuitive, and simple to understand even if your apps are not simple underneath the covers.

# 9 Emulation vs. Physical Device

There is a lot of discussion about using real physical devices and emulation. Physical devices are what the consumers will be using and as a tester, you should try to simulate the end user experience as much as possible. However, the devices are designed to be consumer grade level devices, and are not necessarily designed to be run 24 hours a day, 7 days a week as a test device. The devices will eventually fail and you may discover flakiness due to hardware issues, radio issues, heat etc. especially if the device is older, or has been used heavily for testing. Therefore, it is important to have a good mixture of emulation / simulators and physical device testing.

In Thomas Knych's portion of the GTAC 2013: Breaking the Matrix – Android Testing At Scale presentation, he discusses testing automation using physical devices vs. emulating devices. See the section starting at https://www.youtube.com/watch?v=uHoB0KzQGRg#t=812

It's worth watching if the tester is unsure about the pros and cons of emulation/simulation vs. physical devices. The basic conclusion is to try to do as much testing on a simulated device as possible, because the tester can spin up a variety of screen sizes, API levels, and memory configurations at a lower cost than on physical devices. Physical device testing should primarily be used for testing the areas that simulators cannot do, or don't do well.

### **9.1** Personal Experience – Simulator

*The Android developer on our team did not have access to a device running Android Kit Kat, as he had a Nexus with Android Jelly Bean. One of the defects that QA found only occurred with a Nexus 5 running Android Kit Kat. The Android developer was able to reproduce the Kit Kat defect, debug and implement a fix using a simulator.*

A physical device may be necessary if your application is explicitly using a sensor or input that cannot be easily simulated. Many applications use location services tied to GPS and Wi-Fi, camera, or microphones. Some of the latest devices include accelerometer, gyroscope, temperature, pressure, magnetic field, light, step counter and relative humidity and some of these are not be easily simulated. See information about inputs in "Sensor" article of Android Developer Hardware Reference.

If the app uses GPS, you can use mock locations as long as you use a simulator that has Google play services enabled.

**HAXM suggestion**

If you are worried that simulators are slow, you can enable and use Intel's hardware image to speed up the simulator. It works for both Windows and Mac machines. It's called Intel Hardware Accelerated Execution Manager or HAXM for short.

https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager

for windows machines, you may need to boot into the bios and enable virtualization. However, once that's done you can be speeding along with a simulator.

In the GTAC 2013: Breaking the Matrix – Android Testing at Scale, they had a demonstration where tests were run on a HAXM enabled simulator that ran faster than a physical device (Nexus 4).
https://www.youtube.com/watch?v=uHoB0KzQGRg#t=1434

# 10 Hardware Testing

Some considerations for testing with hardware include location services such as GPS and Wi-Fi, push notifications, data connections and device level interrupts.

## 10.1 Geofencing and Location Testing

iOS seems to have pretty good location services and is fairly consistent with the location tests, due to aGPS and Wi-Fi being on by default for users. With Android, there is quite a bit of variance on how well the GPS operates from device to device. The Wi-Fi assistance and variability in the hardware / chips that are used based on a manufacturer's. Some android devices can take a couple of minutes to lock onto a location, but if you are indoors it may not lock onto a location with GPS even after 10 to 15 minutes.

### **10.1.1** Personal Experience – Locations Testing

*For the application that I test, we ran some tests with GPS inside the office building, and I was not able to get any GPS lock even after 10 minutes. I tested placing multiple devices next to the window and walking around the office building, and still was not able to get a GPS lock. However, within one to two minutes of walking outside I was able to obtain a GPS lock. If your app uses location services, you and your developers need to be how accurate location services are and how long it will take and how quickly the* *device will lock onto a position. Also, there are power and latency considerations as well. Both iOS and Android have coarse and fine ratings for their location services.*

## 10.2 Push Notifications

### 10.2.1 Push Notifications on Android Simulators

For Android, push notifications can be received on a simulator, but the simulator will need to have google play services enabled. In the Android SDK Manager they will say Google APIs (x86 System Image) or (ARM System Image).

For more information see the following on stackoverflow.com “Push notifications don't work”, and “Android emulator not receiving push notifications”.

### 10.2.2 Push Notifications on iOS Simulators

For iOS, push notifications cannot be sent to simulators, so the tester will need to use a physical device. An iPhone, iPod Touch or iPad on Wi-Fi will work. The device does not necessarily have to have a cellular connection and will still be able to receive push notifications.

For more information see the following on stackoverflow.com “can we check push notification in simulator? [duplicate]”, and on stackexchange.com ““Is it possible to send push notifications to the iPhone Simulator?”.

## 10.3 Data Connections

As a Mobile Tester, it is important to consider the intents of the user and most likely where they will be using your application. If your application relies on a data connection, you will need to take into account how the application behaves when

1. The cellular connection is intermittent
2. The user switches from cellular to Wi-Fi
3. The user switches from Wi-Fi to cellular
4. How other applications utilizing data could affect the data performance of your application

Network Consistency can be a consideration that new mobile app testers don’t think about too much. With a Web App, typically the user expects the website to have excellent uptime, and just work. With cellular connections and switching between cellular and Wi-Fi, the network connectivity needs to be taken into account for your testing.

## 10.4 Device Level Interrupts

Mobile testers need to take into consideration device level interrupts such as:

1. Phone calls coming in and how the app handles going into the background
    a. Resuming after the user has completed the telephone call
2. Text messages coming in and how the app handles going into the background
    a. Resuming after the user has completed reading and replying to a text message
3. Low battery
    a. Does the app decrease the frequency it tries to access the data network if the battery status is low
    b. Does the app shut down or go into the background
4. Notifications

    a. How does the application handle notifications that occur in the background and the user accesses the notifications

## 10.5 Multiple apps running

How does the app handle multiple applications being open?

Does the app gracefully handle going into the background?

If there are many applications opened up and the memory a majority of the available memory is used up, does the app minimize the amount of memory?

If the user runs some of the commonly used system level and third party applications such as maps, calendar, weather, news, does the app perform properly?

If the app interacts with other apps, are they handled properly such as:

1. Sharing links or articles
2. Launching a web browser
3. Launching the email application with some of the information pre-configured for to address and subject line
4. Opening phone numbers in the dialer
5. Launching a map application to navigate to a particular location
6. Voice search using microphone input
7. Image capture using camera or front facing camera
8. Capturing video

If there is music or sound effects for your application, how does it interact with system level sounds?

How does the app interact with applications that use sound such as music, maps, voice search?

# 11 Beginning Mobile Test Automation

As a Mobile App Tester, trying to determine what to manually test and what to automate is a balancing act. When you embark on your mobile app testing journey, you will most likely have to do a lot manual exploratory testing. As you progress and the testing process matures you will gain an understanding of the core functionality of the application under test.

The mobile tester can start automating the basics and build up a set of tests that can cover all the areas that would qualify for a smoke test to provide feedback to developers in a quick and timely manner.

Before the Mobile App Tester begins to automate any tests, it’s

1. Take a look at some of the options for mobile test automation
2. Determine the requirements and desired outcomes of automated mobile testing
3. Compare the pros and cons of the various tools to the needs and desires of the development team, product owner, and business value
4. Assess the familiarity of the test writers to programming and the various programming languages that may be involved with writing test automation.
    a. Will QA people with experience be writing and maintaining the tests?
    b. Will developers be writing some of the automated tests?
    c. Will any non-programmers or non-technical people need to look at tests or interpret test results?
5. Decide on a test framework and test strategy

6. Try a proof of concept for the selected testing framework
7. Evaluate or re-evaluate if the framework will fulfill the requirements
    a. If it fulfills the requirements, continue to write additional tests to cover a smoke test of the mobile app
    b. If it does not fulfill the requirements, find the next testing framework down the line and repeat steps 5-7 until the tester finds a testing framework that fulfills the requirements
8. Periodically re-visit the testing frameworks, and updates to see if the tools have improved.

One of the simplest tests to start writing is logging into your application. The next one may be registering a new user. After that, look to automate some of the most common things that a user would need to perform within the app; this can include CRUD (Create Read Update Delete functionality) for various aspects of the application. After that has been completed, start adding additional features and hooks to build out the testing framework specifically for the mobile application.

Currently, there is no one size fits all solution for automated mobile app testing. Each team seems to come up with their own solutions that seem to work best for their situation.

Some of the Testing frameworks that

**Android**

- Robotium
- Espresso
- UiAutomation

**iOS**

- UI Automator
- KIF
- Frank
- iOS Driver

**Both Android and iOS**

- Appium - http://appium.io/
- Calabash - http://calaba.sh/
- Monkey Talk - https://www.cloudmonkeymobile.com/monkeytalk

### 11.1.1 Personal Experience – Beginning Mobile Android UI Automation

*For our Android mobile automation QA initially investigated Appium. We tried setting up Appium on Windows and ran into difficulties starting and running Appium on windows. Then, the development team tried Robotium and was able to get it started in a day or two. We built up a testing framework with tests that covered the typical smoke test and core areas of our mobile app. However, there were some issues with consistencies and intermittent failures.*

*We attempted to incorporate Jenkins and CI on a virtual machine and ran into major issues with timeouts and test failures. The tests would run without issues locally, but on the VM the tests would time out even after increasing global time outs significantly. After GTAC 2013, there was information about Android Espresso that was written by Google employee, Valera Zakharovand. The development team tried a proof of concept and transitioned to using the Espresso Framework a couple of months after the release of Android Espresso. QA and the development team have been adding tests to the automation suite and working on incorporating CI with Jenkins.*

You can find out a little more about Espresso:

https://code.google.com/p/android-test-kit/

In writing tests and helpers for mobile test automation, some different styles of the framework may include:

- Page object model, or in the case of the app a screen object model
- Flow or path dependent based on activities of the app

Some advantages of the screen object model, is that all elements on a screen will be accessible for the screen and can be easily reused. One disadvantage is that some of the helper methods can have very little functionality, and it can get complicated to navigate between different menus or screens if the tester needs to chain together a sequence of multiple screens.

The Flow or path methodology works well if there are many different steps in the application that is being tested and there are many screens that would need to be accessed. It is a good idea to try to encapsulate methods that can be re-used into helper files. One of the disadvantages of flow testing is that if there is a slight variance in the test a new flow or path of the test will need to be initiated.

### 11.1.2 Personal Experience – Screen Object Model and Flow Testing

*When the QA and development team started with the Robotium framework, we borrowed a lot from our experience with Selenium and decided to go with a Page Object Model / Screen Object Model. It seemed to work well during the first couple of iterations of the mobile application. However, as more functionality of the app was added it seemed to get heavier and a bit cumbersome.*

*When the development team and QA decided to transition to Espresso, we decided to attempt to write tests in a flow methodology with each test containing the full flow of the test. IN the beginning, there wasn't as much encapsulation of functionality as we would have liked.*

*Eventually the development team and QA concluded on doing a combination of Screen Object Model and Flow testing for the mobile application depending on the complexity of the particular test. Often times, there would need to be 4 or 5 things that would need to be setup before the functionality that was under test could actually be tested. The common setup steps ended up being a part of the flow testing and the new functionality would have more of a screen object model paradigm applied to it.*

Mobile App Test Automation is a changing process and new tools and better methodologies continue to appear as the mobile app test automation space continues to evolve and mature.

# 12 Conclusion

Mobile App Testers can see that there are a variety of considerations to take into account when creating and implementing the mobile testing strategy. If the mobile app tester is coming from an n-tier application architecture or web app testing environment, the web services and back end still function the same. The main difference is the presentation tier is on the smartphone rather than on a webpage of a desktop or laptop computer. Many of the same skills and techniques can be re-purposed for mobile app, and web app testing on mobile devices.

This paper discussed WebApp Testing vs. Mobile App Testing; the dominant Mobile Operating Systems of iOS and Android; differences between native apps, hybrid apps, and web apps; app store submissions; mobile app testing considerations; frequency of use and app value; Emulation vs. Physical devices; Hardware testing; and Beginning Mobile Test Automation.

After reviewing this paper the mobile tester should have a better idea of all of the considerations and things that they might not initially had before they sit down and begin testing. They can review the questions about testing strategy, devices, emulation vs. physical devices and map out a testing strategy that will fit into the constraints of the business, budget, and time availability. Test and try out many different apps and become comfortable with exploratory and manual testing.

Mobile app testing can be a challenging arena due to the many changes of hardware, operating system, UI changes, UX changes, market expectations, and the constant upgrade cycle of users. It is also rewarding to see the application being used and providing value for users and knowing that the mobile tester contributed to an intuitive and enjoyable experience for the end user.

This is definitely an exciting area to be doing software quality assurance and testing, at the same time the tools and platform that the mobile tester is using today and will use tomorrow is constantly changing. Many of the problems and tools that are still being developed today will help solve the issues that mobile testers are encountering today and in the future. Just as web app testing took time to mature over the last several decades, mobile testing will evolve and the tools and best methodologies will be firmed up. Testers will see tools mature and industry accepted tools similar to what we have with web service and web app testing.

# 13 References

"comScore Reports January 2014 US Smartphone Subscriber Market Share", comScore.com, last modified March 7, 2014, accessed June 16, 2014, https://www.comscore.com/Insights/Press-Releases/2014/3/comScore-Reports-January-2014-US-Smartphone-Subscriber-Market-Share

"One In Every 5 People In The World Own A Smartphone, One In Every 17 Own A Tablet [CHART]" Smartphone And Tablet Penetration - Business Insider", BusinessInsider.com, last modified December 15, 2013, accessed June 16, 2014, http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

"Smartphone Users Worldwide Will Total 1.75 Billion in 2014", eMarketer.com, last modified January 16, 2014, accessed August 18, 2014, http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

Mary Meeker, "Internet Trends 2014 - Code Conference" (slides presented at 2014 Internet Trends by Kleiner Perkins Caufield Byers), http://www.kpcb.com/internet-trends

"Sensors Overview" Android Developer Sensors Topics, accessed June 16, 2014, https://developer.android.com/guide/topics/sensors/sensors_overview.html

"Mobile: Native Apps, Web Apps, and Hybrid Apps", Raluca Budiu, modified September 14, 2013, accessed June 16, 2014, http://www.nngroup.com/articles/mobile-native-apps/

"Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options", salesforce.com Developer, accessed June 16, 2014, https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options

Chris P., "HTC Sense 6 vs Samsung TouchWiz vs Sony Xperia: UI comparison", PhoneArena.com, modified March 25, 2014, accessed June 16, 2014, http://www.phonearena.com/news/HTC-Sense-6-vs-Samsung-TouchWiz-vs-Sony-Xperia-UI-comparison_id54269

Ron Amadeo, "Balky carriers and slow OEMs step aside: Google is defragging Android", modified Sept 2 2013, accessed June 16, 2014, http://arstechnica.com/gadgets/2013/09/balky-carriers-and-slow-oems-step-aside-google-is-defragging-android/

Kukil Bora, "Apple iOS 7 Two Months After Release: Adoption Rates Cross 70% Mark, Surpassing Those Of Predecessor's", modified December 4, 2013, accessed June 16, 2014, http://www.ibtimes.com/apple-ios-7-two-months-after-release-adoption-rates-cross-70-mark-surpassing-those-predecessors

“UPDATE: iOS Version Distribution Study - 2012 through 2013”, Chitika.com, modified December 12, 2013, accessed June 16, 2014, http://chitika.com/insights/2013/ios-distribution-update

Thomas Knych, Stefan Ramsauer, & Valera Zakharov, “GTAC 2013: Breaking the Matrix – Android Testing At Scale”, Google Test Automation Conference, modified April 29, 2013, accessed June 16, 2014, https://www.youtube.com/watch?v=uHoB0KzQGRg

“Sensor”, Android Developer Hardware Reference, accessed June 16, 2014, http://developer.android.com/reference/android/hardware/Sensor.html

Christopher Lawless, “Android emulator not receiving push notifications”, stackoverflow.com, modified December 11, 2013, accessed June 16, 2014, http://stackoverflow.com/questions/20521600/android-emulator-not-receiving-push-notifications

user1242617, “Push notifications don't work”, stackoverflow.com, modified May 23, 2014, accessed June 16, 2014, http://stackoverflow.com/questions/16713321/push-notifications-dont-work

Rahul Vyas, “can we check push notification in simulator? [duplicate]”,  stackoverflow.com, modified November 28, 2009, accessed June 16, 2014, http://stackoverflow.com/questions/1811900/can-we-check-push-notification-in-simulator

Simon, “Is it possible to send push notifications to the iPhone Simulator?”, stackexchange.com, modified March 5, 2012, accessed June 16, 2014, http://apple.stackexchange.com/questions/42654/is-it-possible-to-send-push-notifications-to-the-iphone-simulator

# Non-Functional Risk Assessment Framework to Increase Predictability of Non-Functional Defects

**Vijayanand Chelliahdhas**

**HCL America, Inc.**

Vijayanand.c@hcl.com

## Abstract

Typically organizations tend to adapt a one-size-fits-all approach to validate a system for its non-functional (NF) requirements. Not surprisingly, a significant number of non-functional issues surface post go-live. Such unpredictability in the non-functional quality of a software implementation can be largely attributed to the lack of a focused approach to non-functional testing targeting the critical technical and non-functional risks in the system under test (SUT).

This challenge can be mitigated by conducting a Non-Functional Risk Assessment exercise to develop a comprehensive Risk Catalogue. As a result, the NF test strategy and scenarios would be directly mapped to all the technical and NF risks in the system.

The “NF Risk Assessment Framework” described in this paper enables to precisely determine the key vulnerable areas in the system and not just the application, addresses coverage across multiple non-functional domains of the SUT and not just performance, ensures the tests are designed to simulate production condition as close as possible and helps to prioritize test activities based on the criticality assigned.

## Biography

*Vijayanand (Vijay) Chelliahdhas is a Solutions Architect at HCL America, Inc. currently responsible for providing solutions around non-functional testing including Performance Testing and Engineering, Security Testing, Mobility Performance, Testing on the Cloud and Service Virtualization across the US geography.*

*For more than a decade, Vijay has been involved in evaluating performance for a variety of systems including OLTP, batch programs, middleware and package implementations across multiple technologies. Vijay has extensive experience in devising performance engineering strategies for multi-year multi-release IT programs, setting up performance test centers of excellence and providing consulting services to optimize the ways of working in the performance testing and engineering lifecycle. Additionally, Vijay also specializes in conducting maturity assessment for performance testing organizations and risk assessment for multi component distributed systems.*

# 1 Introduction

The standard approach to non-functional testing, which in most cases is limited to performance testing only, is to identify a subset of use cases, which are anticipated to be executed frequently in production, and conduct a set of non-functional tests. Such tests could include Load Tests, Stress Tests, Endurance Tests et al using any renowned commercial off the shelf (COTS) tool and then publish the test results.

Where the test results indicate that certain transactions do not meet the stipulated Service Level Agreements (SLAs) (if they were defined in the first place) then either an exception is sought to go-live as per schedule or execute a re-test post tuning. Such tuning attempts are predominantly limited to modifying certain configuration values either at the application server or database server layer or both to alleviate the symptoms of performance issues.

This superficial approach to non-functional testing tends to leave gaping holes in the system and exposes it to serious performance and non-functional quality issues when the “rubber hits the tarmac” in production.

Given that there could be various reasons why the non-functional issues were not detected in a pre-production environment, one of the most critical elements is the ability to simulate the anticipated risks in a production like pre-production environment. To accomplish the same, the system under test should be systematically studied and all the potential risks thoroughly understood in the context of historical production non-functional issues and futuristic workload and deployment changes.

This white paper explains how to undertake a Non-Functional Risk Assessment exercise to ensure the non-functional test strategy and scenarios have a fool-proof coverage of all the technical and non-functional risks to maximize predictability of non-functional quality attributes of the system.

# 2 Objective and Outcome of Non-Functional Risk Assessment

The driver for conducting a Non-Functional Risk Assessment is to technically assess the System Under Test (SUT) for potential non-functional risks and thereby develop a Non-Functional Risk Catalogue, which would be the basis for developing the non-functional test strategy and the test scenarios.

The Non-Functional Risk Catalogue is a collection of detailed risks and their impact to the non-functional quality attributes of the SUT. This catalogue is derived from a Risk Matrix which is essentially an intersection of the various potential “Threats” and “Focus Areas” for the SUT.

The following sections detail the method to study a software system, assess it from the various non-functional attributes perspective and arrive at a detailed risk catalogue.

# 3 Non-Functional Risk Assessment Framework

The proven framework for non-functional risk assessment comprises of the following 3 steps:

- System Appreciation and Technical Assessment
- Develop Risk Matrix
- Create Risk Catalogue

This framework was implemented to assess the non-functional risks of a Trading Product developed by a major UK based banking software provider. The following sections of the document will leverage the above said actual implementation experience for illustration purposes.

## 3.1 System Appreciation and Technical Assessment

This is the fundamental step in the framework, the output of which is critical in building the Risk Matrix subsequently. This is further broken down into 3 steps as follows:

- Study application architecture and design
- Historical non-functional incidents analysis
- Understand future deployment and workload characteristics

The architecture of the SUT is thoroughly studied from the logical and physical point of views, the technology landscape, including studying the various components of the architecture, the core business engine, how the complex algorithms were implemented, the synchronous and asynchronous communications, CPU or Memory or I/O intensive operations, what were the integrated components and how the data flows in integrated scenarios.

If the SUT has already been in production, then the information from the historical incidents reported in production is invaluable information to understand the maturity and weak areas of the system. In our example, at least 12 months of data from the Incident Management System was extracted for a thorough entry by entry analysis. Besides documenting various record keeping fields for each Incident shortlisted, the following data were also gathered, analyzed and documented. A sample entry is shown below:

| | |
|---|---|
| Incident Description | Trunk End of Day (EoD) has a reproducible error where extract reports are requested in conjunction with re-use reports. The remaining steps are still unexecuted but the EoD job has completed. |
| Technical Analysis | When a job/step is bypassed, the next job/step should take over and execute. And EoD status should reflect this appropriately. However in actuality, due to bypass of a step, the EoD loses status integrity (suspected to be at least 2 causes - weblogic to database (DB) connection starvation, DB lock contention) |
| Non-Functional? (Y/N) | Y |
| NF Domain | Reliability |
| Class of Issue | Connection Pool Starvation, Database lock Contention |
| Sub-Class | Bypass a step in EoD (particularly a step in Extract Report) |
| Potential way of Detecting the issue | Schedule an EoD run with Extract Reports in conjunction with Reuse Reports. Bypass a step manually in the middle of the run (particularly a Step in Extract Report) Track the status of the next job, current job and overall EoD |
| Applicable NF Test | EoD Batch Test (peak daily volume) |

Out of a total of 3000 incidents, about 212 non-functional incidents were identified for in-depth analysis. Each of those shortlisted entries were studied and documented in the above format.
After the historical incidents analysis, several discussions were conducted with two program teams who were involved in the two biggest implementations of the software product spanning over 36 months. The objective was to understand the technology stack, the infrastructure capacity details such as the number of nodes, VMs, data centers, latency between the architecture components, the latency between the end users and the datacenters, growth in business volume and therefore the projected workload, the growth in the number of end users, the rate of growth of the database size, the plan and algorithm for data archival and purging.

This comprised all the activities involved in System Appreciation and Technical Assessment, which helped gather all the requisite data to arrive at the Risk Matrix.

## 3.2 Develop Risk Matrix

The Risk Matrix is essentially precipitated from the information analyzed in the above step. The objective of the risk matrix is to map the potential “Non-Functional Threats” to the “Focus Areas” of the SUT and to highlight the degree of impact of each Threat – Focus Area mapping.

A “Threat” is a technical attribute or event that can impact the non-functional quality of the SUT. Examples of threats classified in the illustration include: processing overlaps, concurrency, JVM sharing, multi-geo access, VM crash, incorrect error handling etc.

A “Focus Area” is a component or set of functionalities in the SUT that is critical to the non-functional quality of the SUT. Examples of focus areas classified in the illustration include: OLTP scenarios, EoD processing, interfaces processing, global app, infrastructure utilization etc.

In our particular illustration, 14 non-functional domains were initially identified which was later crystallized to 7 at the end of the System Appreciation and Technical Assessment phase. The 7 Non-Functional domains included Scalability, Reliability, Performance, Resilience & Recoverability, Capacity, Interoperability and Compatibility.

There were a total of 31 Threats identified across the 7 Non-Functional domains and these were mapped to 7 Focus Areas. This resulted in a mapping of 217 non-functional risks in total, which was then classified to 5 different types as indicated below.

Following is a snippet of the Risk Matrix, displaying a sub-set of Threats mapped to the Focus Areas.

**NFT Risk Matrix**

| Non-Functional Focus Areas >> | | Online Transaction Processing (Transactional) | EOD Processing | Adhoc Reports | Interfaces Processing | Global App Behaviour | Zone Behaviour | Infrastructure Utilization |
|---|---|---|---|---|---|---|---|---|
| **NF Domain** | **Threat** | **TI Processing** | | | | **TI Systems** | | **Infrastructure** |
| **Scalability** | Processing Overlaps | ▓ | H | ░ | ▓ | ▓ | H | ░ |
| | Concurrency | H | ▓ | ▒ | ░ | H | ▓ | ░ |
| | Integration Complexity | ░ | ░ | | H | | ▓ | ▒ |
| | Horizontal Scalability (lack of) | ▓ | ░ | | ░ | ░ | ▓ | |
| **Reliability** | Processing Overlaps | ░ | ░ | ▒ | ▒ | ░ | H | |
| | Stress Conditions | ░ | H | ░ | ▓ | ▓ | H | ░ |
| | Concurrency | H | ░ | ▒ | ▓ | ░ | ░ | |
| | Prolonged Usage | ▓ | ░ | ░ | ░ | H | H | ░ |

Figure: Non-Functional Risk Matrix

| Rating | Description |
|---|---|
| H | High Impact + High Probability of Occurrence |
| ▓ | High Impact |
| ░ | Medium Impact |
| ▒ | Low Impact |
| | Negligible Impact |

Figure: Legend for the Non-Functional Risk Matrix

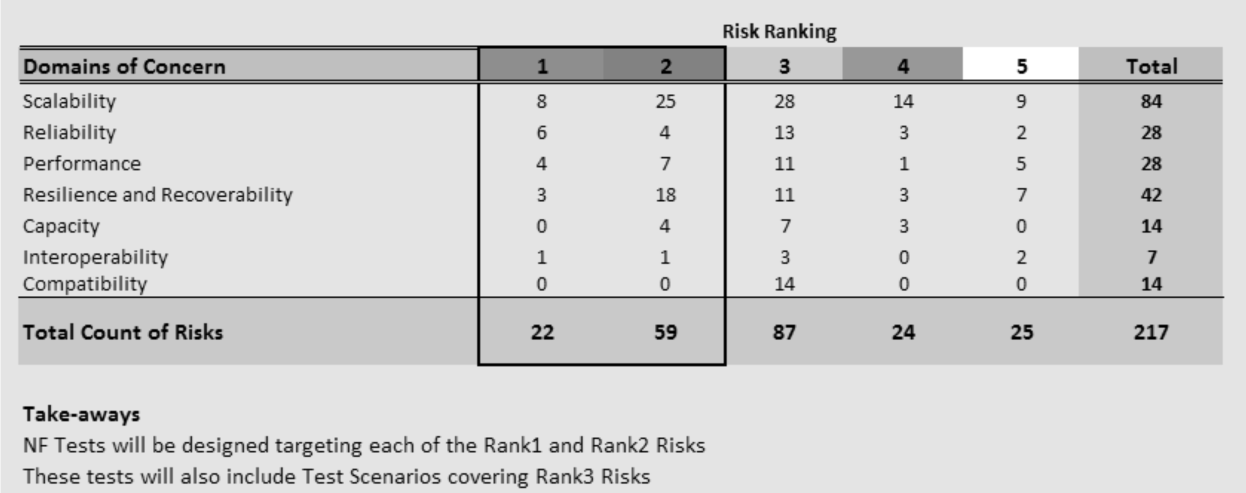
## 3.3 Create Risk Catalogue

Once the Risk Matrix is developed, the next step of creating the Risk Catalogue is essentially a task of detailing the risks in accordance with each Threat – Focus Area mapping. In our particular illustration, out of the total 217 risks, about 81 risks which fell under the first 2 categories (brown & red) were documented in detail in the form of a Risk Catalogue.

For each Risk Catalogue entry, besides the risk and impact, the sub-threats and parameters to measure were also gathered and documented which could be directly converted to Test Scenarios. Following are 2 sample entries from the Risk Catalogue.

| **#** | 1 | 2 |
|---|---|---|
| **NF Domain** | Scalability | Performance |
| **Threat** | Processing Overlap | Multi Geo Access |
| **Sub-Threats** | Intra Zone Processing | - |
| **Risk** | Two or more Multi Bank Entities (MBE) within a Zone could be performing different operations at the same time, leveraging the same application/OS/database resources and processing the same data set or accessing from the same data source (table/schema/database) | User sites are spread across the globe, however all user access have to pass through the Global Single Sign On (SSO). There will be only one primary instance of Global App in one location and all users will be routed through this single Global app. |
| **Impact** | There will be intermittent delays in online transaction processing (OLTP) or delays in Message transmission into the Transport Client | User accesses from multiple geographies to the global App and the response therefore will potentially be slow, influenced by the bandwidth congestion over the wide area network (WAN) between the user sites and the global app |
| **Parameters to Measure** | OLTP Response Time | Global Dashboard Response Time |
| **Focus Area** | Zone Behavior | Global App (SSO, Dashboard) |

Eventually the Non-Functional Risks were grouped by the NF Domain and summarized as part of this exercise conducted for the Banking Product.



| | Risk Ranking | | | | | |
|---|---|---|---|---|---|---|
| **Domains of Concern** | **1** | **2** | **3** | **4** | **5** | **Total** |
| Scalability | 8 | 25 | 28 | 14 | 9 | **84** |
| Reliability | 6 | 4 | 13 | 3 | 2 | **28** |
| Performance | 4 | 7 | 11 | 1 | 5 | **28** |
| Resilience and Recoverability | 3 | 18 | 11 | 3 | 7 | **42** |
| Capacity | 0 | 4 | 7 | 3 | 0 | **14** |
| Interoperability | 1 | 1 | 3 | 0 | 2 | **7** |
| Compatibility | 0 | 0 | 14 | 0 | 0 | **14** |
| **Total Count of Risks** | **22** | **59** | **87** | **24** | **25** | **217** |

**Take-aways**

NF Tests will be designed targeting each of the Rank1 and Rank2 Risks

These tests will also include Test Scenarios covering Rank3 Risks

Figure: Non-Functional and Technical Risks Summary

The Non-Functional Risks in the catalogue were then mapped to the identified NF tests to ensure traceability and coverage of the potential risks.

### 3.4 Advantages & Benefits

The following benefits could be potentially reaped, by conducting a comprehensive non-functional risk assessment exercise:

- Gain a precise understanding of the vulnerable areas and use cases (risks) of the SUT
- Develop an exhaustive repository of non-functional test cases/scenarios
- Ability to design tests focused on simulating the specific technical and non-functional risks
- Ensure maximum possible coverage and traceability of the NF risks in the SUT
- Predictability into all probable outcomes in production in the event of a technical failure or an unexpected workload situation or projected business growth

## 4 Conclusion

The Risk Assessment exercise illustrated here was conducted by 2 non-functional consultants over a 6 week period at the client site. Given the effort it takes to conduct such a comprehensive exercise, it may not be a feasible solution for short-term projects or low complex applications or highly matured systems. However, for software product vendors and applications with large-scale usage with unanticipated workload patterns and business growth scenarios, the qualitative and the quantitative benefits of conducting a Non-Functional Risk Assessment exercise clearly outweighs this modest effort and investment.

This was a consulting exercise where the primary objective was to conduct the Risk Assessment and develop the Risk Catalogue. Subsequent to which, the existing delivery team on the customer side had the responsibility to conduct the test execution. We did not have access to any additional data points or supporting facts post production to substantiate any further benefits due to this exercise.

## 5 Glossary

| # | Term | Description |
|---|---|---|
| 1 | NFT | Non-functional Testing |
| 2 | Load Test | A test conducted by simulating a concurrent workload on the SUT to determine whether it can process the expected workload within the stipulated processing time |
| 3 | Stress Test | A test conducted by simulating an incremental concurrent workload on the SUT to determine whether the system performance begins to degrade in proportion to the increase in workload or the maximum resource utilization is achieved |
| 4 | Endurance Test | A test conducted by simulating a prolonged concurrent workload on the SUT to determine whether system performance sustains or degrades on continuous usage over a period of time |

| | | |
|---|---|---|
| 5 | COTS | Commercial Off The Shelf |
| 6 | SUT | System Under Test |
| 7 | SLA | Service Level Agreement |
| 8 | EoD | End of Day |
| 9 | JVM | Java Virtual Machine |
| 10 | VM | Virtual Machine |
| 11 | OLTP | Online Transaction Processing |
| 12 | SSO | Single Sign On |
| 13 | MBE | Multi Bank Entity |
| 15 | WAN | Wide Area Network |
| 16 | Scalability | Indicates the ability of the SUT to sustain system behavior when the scale of operations (number of instances, amount of workload, integration complexity, capacity and geographical spread) increases exponentially/linearly in a larger scale deployment |
| 17 | Performance | Indicates the ability of the SUT to meet stipulated performance requirements (at the minimum, Throughput and Response Time) when the system is subjected to production like workload |
| 18 | Reliability | Indicates the ability of the SUT to function as intended for a prolonged duration of time under average, stress and unanticipated business workload conditions with minimal failures |
| 19 | Resilience & Recoverability | Indicates the ability of the SUT to handle faults gracefully and systematically. In the event of a hardware or software failure, the system should have the ability to repair and resume business as usual with minimal end user impact |
| 20 | Capacity | Indicates the amount of hardware capacity (in terms of CPU, Memory, Heap, Disk et al) required to run the system for business as usual and for futuristic workload requirements |

| | | |
|---|---|---|
| 21 | Compatibility | Indicates the ability of the SUT to operate as intended on various flavors of the computing environment, comprising of multiple technology platforms and databases |
| 22 | Interoperability | Indicates the ability of the system to interact and exchange information seamlessly with other systems across different protocols and transmission formats in heavy workload, high stress conditions |

# Testing @ Tango

**Amit Mathur**

amit@tango.me

## Abstract

Tango is a free mobile messaging service with 200+ million registered members available on iPhone, iPad, Android phones and tablets. Tango's large user base, broad platform support, frequent releases, and organizational design pose challenges to achieving quality. In this paper I discuss how we are addressing these challenges from our organizational and testing culture, as well as how we utilize test and deployment automation.

We have a culture at Tango of pushing the limits of what is possible and the same applies to our testing mindset. We are constantly adopting tools and techniques that hit at the core of problems with minimum bureaucracy so that we can continue on a release cadence of every 2 weeks.

This paper touches upon how we have approached testing as well as other ways in which we have attempted to address the challenge of shipping with high quality. Some efforts were more successful than others and I will share some lessons learned.

## Biography

*Amit has over 15 years of experience in software quality assurance as a tester, test architect, test manager, and Director of Technology at Motorola, Veritas, Symantec, Microsoft, and now Tango. He has presented in numerous testing conferences including CAST and Star West.*

*Amit holds a B.S. in Computer Science from the University of Illinois at Urbana Champaign*

# 1 Introduction

Mobile communications is hot and Tango is at the center of the mobile social communications race. I would describe the race as more of a sprint than a marathon. This has led us at Tango to converge towards a 2-week release cycle.

As if releasing every 2 weeks isn't challenging enough, we also try hard to give vertical feature teams a lot of autonomy to get their features out which poses interesting challenges when we attempt to integrate all the pieces together.

# 2 Organizational Design

From a testing perspective our over-arching strategy is to have developers test the code that they write and to provide them with the necessary tools and motivation to do a thorough job.

We are organized into multiple *vertical* teams that own the key resources that they need to deliver on features. These resources in include Android Developers, iOS Developers, Server Engineers, and Test Engineers.

There are some *horizontal* teams, whose job it is to support the vertical teams in getting their features out with high quality. The horizontal team are EVIL(core server infrastructure), Server QA (integration test for servers), and Client QA (client end to end testing).

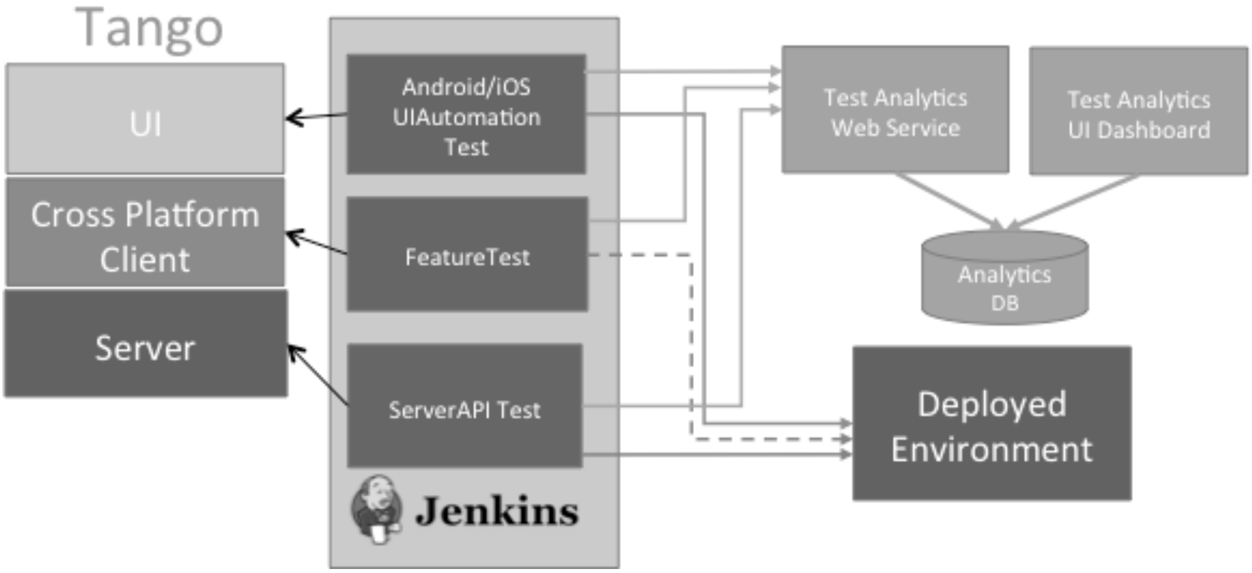
# 3 Wine Helps Us Achieve Quality

Wine is a big deal at Tango (due to in no small part our VP of Engineering and Chief Wine Geek Gary Chevsky[1]). In addition to having wine tastings, we have found ways to utilize wine to help us achieve quality using the construct of a 'Wine Bet'.

A Wine Bet is placed between a vertical team leader and the VP Engineering wherein a specified dollar value of wine is bet against the number of high priority bugs that are found by the test team after a feature branch is merged to trunk. If greater than the number of agreed upon bugs are found by QA then the vertical team leader loses the bet. The idea is to incentivize teams to only merge to trunk when they are ready. Usually the vertical team lead has to buy a bottle of wine.

# 4 Automation

## 4.1 Overview



At a high level, we have three test frameworks with different goals. The UI automation frameworks are designed to test the UI layer and below (all layers) for end-to-end coverage. The FeatureTest framework is designed to cover the client cross platform layer to server layer. The ServerAPI tests are designed to specifically target servers.

All frameworks are designed so that test cases can be annotated and results sent to a central analytics server to provide a high level view of all tests and the ability to drill down to specific test failures to reduce risk associated with product and test issues.

## 4.2 UI Automation

Our original UI automation strategy was to leverage Android and iOS UI automation to cover our new client UI during a release. The problem is that our UI changes very frequently (daily) and our automation ends up continuously broken. We decided that instead of spending tons of time during the release fixing the functional tests we would focus our efforts on two fronts:

1. Performance/Reliability – The performance and reliability of our clients are critical as they have a direct impact on the user perception of quality and their engagement. We realized that we simply could not do performance and reliability without automation since it requires running in loops and taking time measurements. Towards the end of the release, we run performance automation for core scenarios as part of the final sign off to ensure there is no performance regression for new clients.
2. Server Regression Validation – Although we get solid targeted server coverage via ServerAPI tests and client/server coverage via the FeatureTests that can be targeted towards server environments, we still like to ensure that when we release new server code that our existing production clients do not break. This is accomplished by continuously running the current production client release as monitors against our server environments to ensure that core users scenarios are working end to end. This does not suffer from the problem of client UI constantly changing since we are keeping clients fixed and changing the servers.

### 4.3 Feature Test Framework

The Feature Test Framework is a hermetic[2] system (self-contained server system with no network connection) utilized by developers to author end to end tests that can be run on their development machine. The system consists of a test client that is a light wrapper on top of the cross platform client layer that contains the bulk of the client code that gets released with the exception of the user interface layer.

Developers are required to author feature tests to ensure that they have end-to-end coverage. The key benefit of feature tests is that developers can understand the impact of changes throughout the ecosystem on their components. For example if they have a dependency on a downstream component that changed, they may witness their feature tests breaking as a result and quickly get the issue resolved with the relevant party.

Finally, a subset of feature tests (indicated via the dashed line above) can be run against deployed environments. They serve as tests for environment stability and some of them are turned into production server monitors. They serve as good monitors since they simulate user behaviors very well as they contain the key parts of the clients that users actually use.

### 4.4 ServerAPI Tests

ServerAPI tests are test cases that specifically target direct server interactions. As our services are RESTFul (Representational State Transfer), the tests perform http requests and verify the correct http responses are received.

The idea behind having targeted server tests is that they can be developed and executed without the client being present. In particular, it allows us to target servers directly that only interact with other servers.

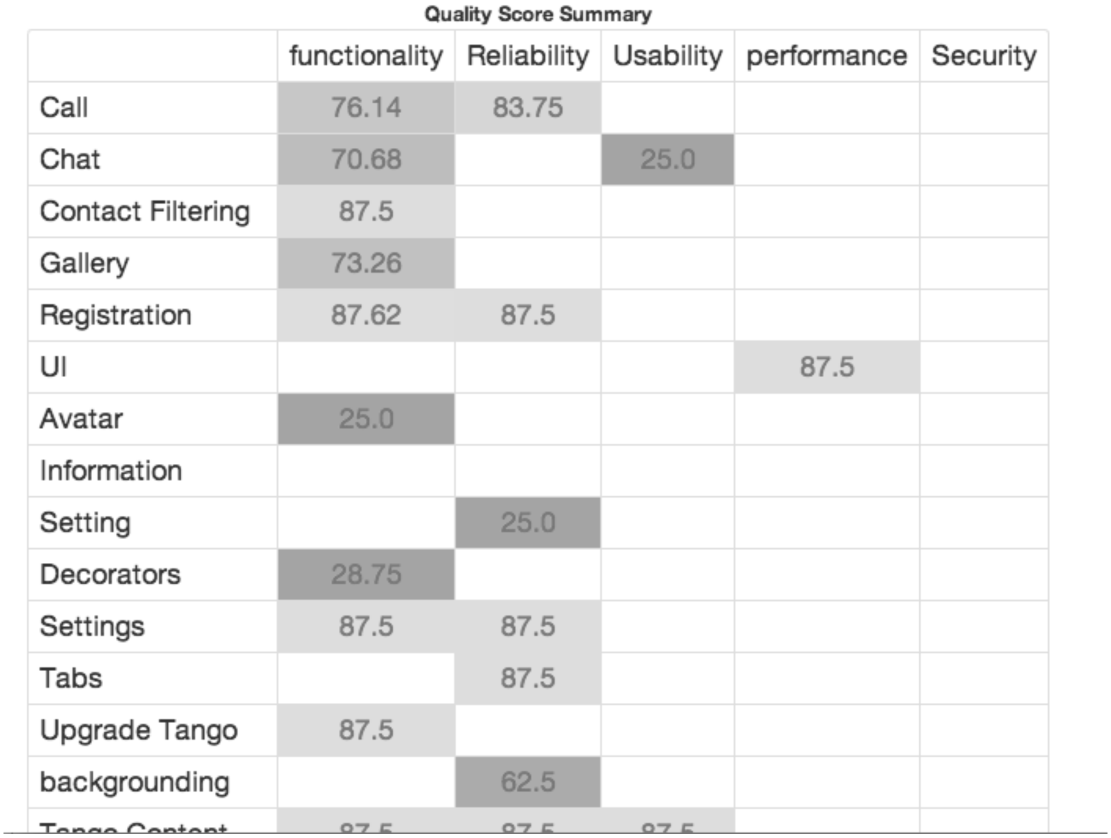
## 5 Fault Injection with Mosh

The Tango is a beautiful and elegant dance. When things go right in production it is equally as beautiful. However, we find that things often do go wrong, and the environment looks more like a mosh pit at a heavy metal concert. As our server side load continues to grow, we constantly hit reliability issues that escape to production.

We created 'mosh' for fault injection system to inject bad stuff (network, cpu, memory, io issues) into our servers in our test environments before they happen on their own in production. For network faults (delays and dropped packets) we utilize *netem*[3]*,* which is integrated into the linux kernel. For CPU, Memory, and IO faults we use a simple tool called *stress*[4]*,* which allows us to target these failure modes directly.

We have found that this approach helps us target reliability bugs that only manifest on systems where resources are scarce as well as bugs related to the impact of a dependent component being down or slow. In particular, dealing with timeout settings for dependent components. We also use this to see how the component recovers once faults are introduced.

# 6 Test Cases



Quality Score Summary

| | functionality | Reliability | Usability | performance | Security |
|---|---|---|---|---|---|
| Call | 76.14 | 83.75 | | | |
| Chat | 70.68 | | 25.0 | | |
| Contact Filtering | 87.5 | | | | |
| Gallery | 73.26 | | | | |
| Registration | 87.62 | 87.5 | | | |
| UI | | | | 87.5 | |
| Avatar | 25.0 | | | | |
| Information | | | | | |
| Setting | | 25.0 | | | |
| Decorators | 28.75 | | | | |
| Settings | 87.5 | 87.5 | | | |
| Tabs | | 87.5 | | | |
| Upgrade Tango | 87.5 | | | | |
| backgrounding | | 62.5 | | | |
| Tango Content | 87.5 | 87.5 | 87.5 | | |

We like to keep our test plans as light weight as possible and provide some visualization of areas of coverage. As such we have adopted the ACC[5] (Attribute/Capability/Component) methodology:

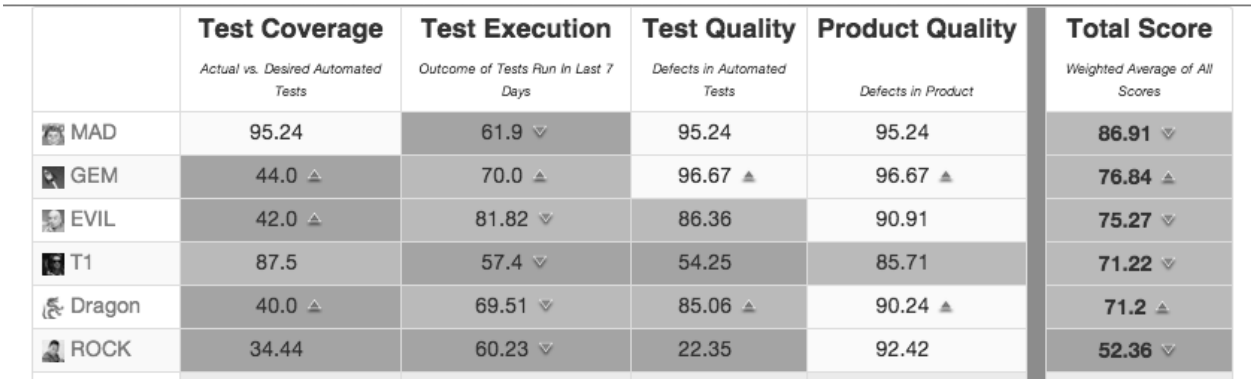
**Capabilities**: These are the "verbs" of the system which contain a feature and attribute mapping. (e.g. text messages should be received within 2 seconds of being sent 99% of the time)
**Attribute**: The "adjectives" of the system (Functionality, Performance, Reliability, Usability, Security*)*
**Feature**: A feature is a "noun" of the system (e.g. video, text, avatar, etc)**.**
**Components**:These are Tango components that implement the capabilities. (e.g. facilitator, authTokenServer, etc). Components can have dependencies on other components.

```
from tea_common import description, components, tags, capability

class TestXmppBasic(unittest.TestCase):

    @implemented(value=True)
    @description(name="Test Tango Registration by sending Tango reg IQ stanza")
    @components(names=["facilitator"])
    @tags(names=["ServerAPITests","Smoke"])
    @capability(name="Users should be allowed to register", feature="registration",
attribute="functionality")
    def test_registration(self):
        print "In test_registration"
```

All automated test cases can be annotated with the above information so that we can visualize risk areas in real time. The key benefit is that instead of seeing a list of failures, we have a better way to know which specific areas are at risk so that we can make quicker informed decisions on how to proceed with the release.

# 7 Visualizations

## 7.1 Team Testing Dashboard



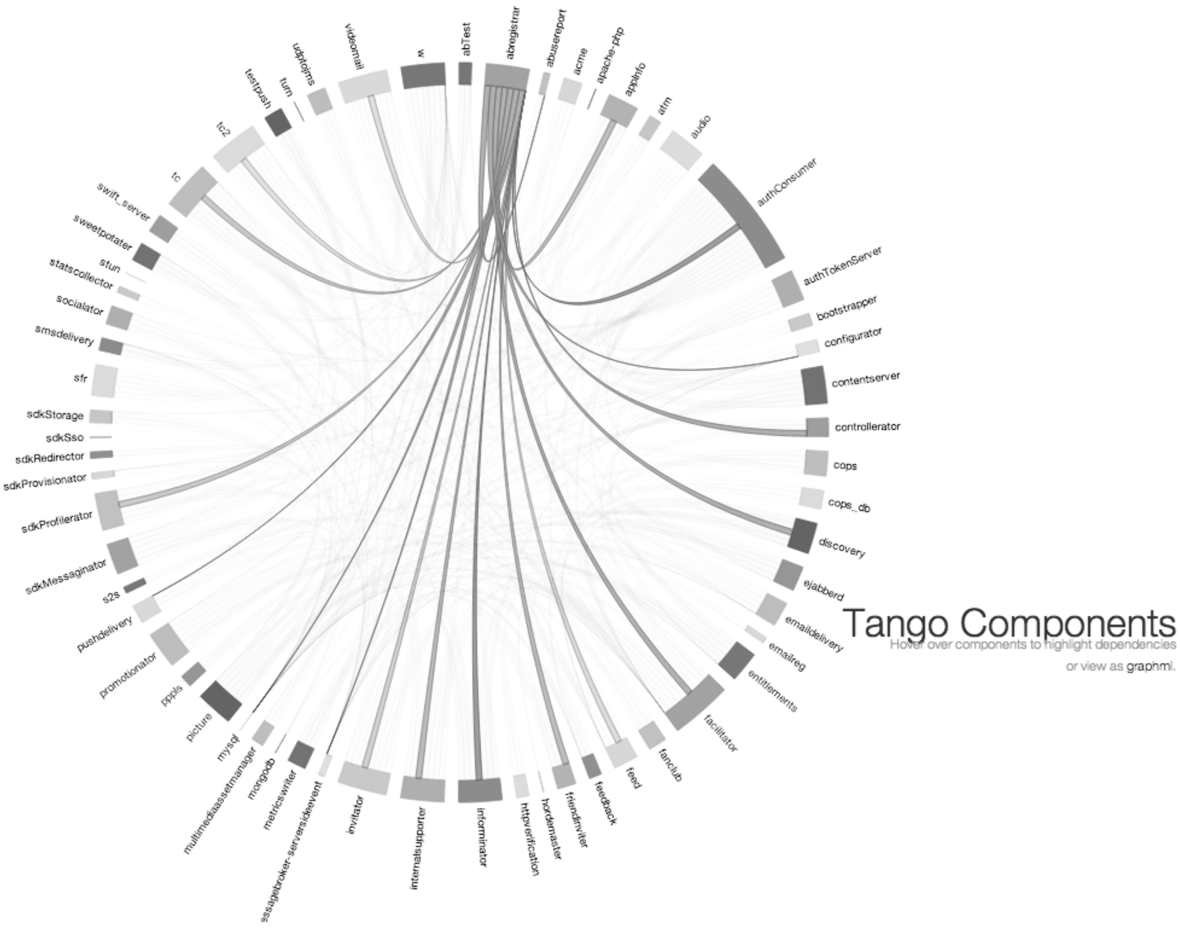
| | Test Coverage<br>*Actual vs. Desired Automated Tests* | Test Execution<br>*Outcome of Tests Run In Last 7 Days* | Test Quality<br>*Defects in Automated Tests* | Product Quality<br>*Defects in Product* | Total Score<br>*Weighted Average of All Scores* |
|---|---|---|---|---|---|
| MAD | 95.24 | 61.9 ▽ | 95.24 | 95.24 | **86.91** ▽ |
| GEM | 44.0 ▲ | 70.0 ▲ | 96.67 ▲ | 96.67 ▲ | **76.84** ▲ |
| EVIL | 42.0 ▲ | 81.82 ▽ | 86.36 | 90.91 | **75.27** ▽ |
| T1 | 87.5 | 57.4 ▽ | 54.25 | 85.71 | **71.22** ▽ |
| Dragon | 40.0 ▲ | 69.51 ▽ | 85.06 ▲ | 90.24 ▲ | **71.2** ▲ |
| ROCK | 34.44 | 60.23 ▽ | 22.35 | 92.42 | **52.36** ▽ |

We want the vertical teams to 'own' the quality of what they built so we built a Test Testing Leaderboard which helps the entire company visualize where each team is at with their test automation (coverage, what was being run, product issues found, and test issues).

The idea is to gamify testing so that teams feel motivated to execute and keep their test automation clean. This encourages a bit of healthy competition between teams. All test automation is run continuously via our Jenkins continuous integration system and fed in real time to this dashboard.

Once nice element of the dashboard is that it gets updated in real time and provides a stock ticker like up/down symbol providing continuous feedback loop to test teams.

## 7.2 Component Dependency Visualization

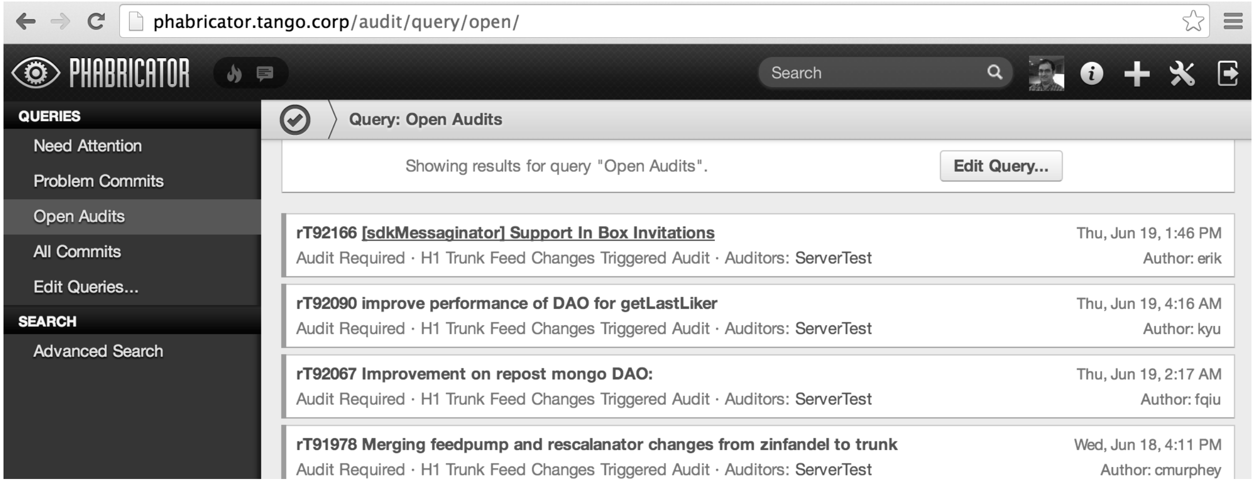


We have over 150 server components that support the backend systems for Tango. It is hard to know when a server component changes what possible impact that could have on the entire system. We built an automated visualization system to help us deal with dependency detection and visualization.

In the example above, if you hover your mouse above a component it will depict all inbound and outbound dependencies on that component. The way that dependencies are detected is based on scanning component configuration information (property files).

This tool has been helpful in both test enumeration and design reviews as it provides a ground truth understanding of dependencies and not what people remember or what exists in outdated design document on the wiki.

# 8 Change Awareness and Impact Analysis



Showing results for query "Open Audits".

rT92166 [sdkMessaginator] Support In Box Invitations — Thu, Jun 19, 1:46 PM
Audit Required · H1 Trunk Feed Changes Triggered Audit · Auditors: ServerTest — Author: erik

rT92090 improve performance of DAO for getLastLiker — Thu, Jun 19, 4:16 AM
Audit Required · H1 Trunk Feed Changes Triggered Audit · Auditors: ServerTest — Author: kyu

rT92067 Improvement on repost mongo DAO: — Thu, Jun 19, 2:17 AM
Audit Required · H1 Trunk Feed Changes Triggered Audit · Auditors: ServerTest — Author: fqiu

rT91978 Merging feedpump and rescalanator changes from zinfandel to trunk — Wed, Jun 18, 4:11 PM
Audit Required · H1 Trunk Feed Changes Triggered Audit · Auditors: ServerTest — Author: cmurphey

Given the quick release cycle and largely autonomous feature teams it can be challenging to keep track of what is important when lots of things are changing all the time. To help with this we adopted Phabricator[6] to enable everyone to subscribe to changes in areas of the code that matter to them.

The QA teams use this to subscribe to changes in components that they own or depend on. This allows conversations to happen earlier than they would normally and hence increase the likelihood to the component being certified on time.

# 9 Unfortunate Escapes

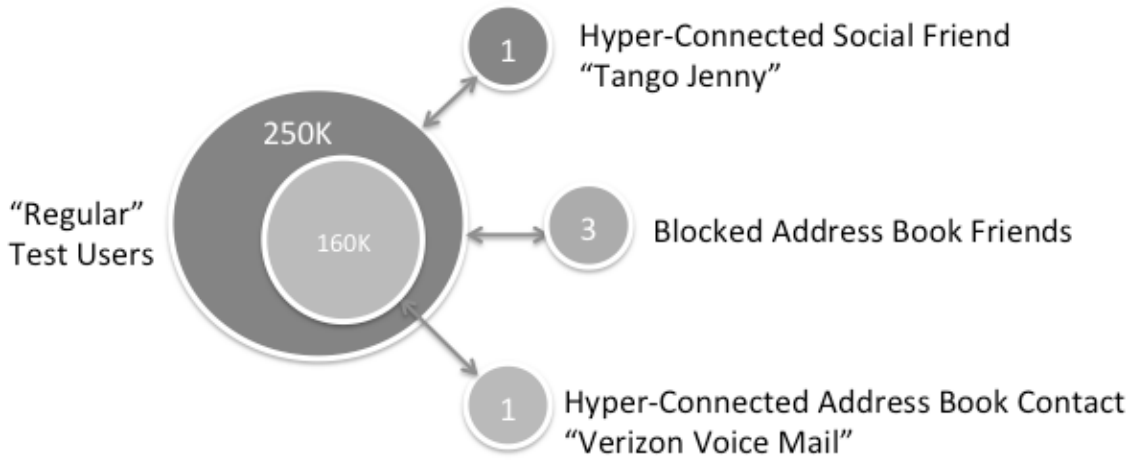
With the engineering systems we have in place we try to catch all bugs before they escape to production but unfortunately we are not perfect. Here are some issues that evaded our systems and wreaked havoc in production.

## 9.1 Verizon Voice Mail

The way Tango works is that users register with a phone number. If that phone number is already registered by someone else then we provide a mechanism via SMS to verify that number. We also provide the ability to publish the contact list on your phone to our servers so that we can tell you who else is on Tango. This works well in most scenarios because (1) most people use their own phone number to register and (2) most people only have dozens of contacts on their phone.

One day neither of these assumptions held as someone decided to register the Verizon Voice Mail address as their Tango phone number. We allowed that since nobody else had claimed that number. The issue that this caused was the entire system came screeching to a halt as our servers were busy attempting to perform reverse lookup on an entry with millions of users since by default many users have the voice mail number in their contact list.

As a result of this we added new kinds of tests to our system that model this hyper-connectivity. We use this to ensure that our backend systems are designed to deal with highly connected people.


1
Hyper-Connected Social Friend
"Tango Jenny"
250K
160K
"Regular"
Test Users
3
Blocked Address Book Friends
1
Hyper-Connected Address Book Contact
"Verizon Voice Mail"

### 9.2 DoS Attack (a.k.a we found the enemy and it was… us)

Having hundreds of millions of mobile clients hitting our backend services requires constant vigilance to ensure that as load patterns change we are able to adapt to the changing behavior. It is common to see spikes in services due to new features that we roll out.

Recently we released a client that had a bug which hammered our backend services almost causing an outage. The issue was related to feedback logs that are sent from the client to our servers to allow us to know about issues. Unfortunately, in this case the feedback log was an issue in itself.

One step that we have added to our certification process is to sniff the packets from a test client to our backend servers to see if there are any large changes in the number of requests. Unfortunately this methodology is not robust since it isn't always possible to trigger the error condition in the client at the time that the packet sniffing is in effect. The most helpful way we have found to address this risk is to slowly release new clients to production and closely monitor the increase in load.

# 10 Conclusion

Based on these (and other) test investments we have been able to scale up the company dramatically while steadily improving the quality of our client software and servers. We still have a lot more work to do to ensure that nasty bugs don't escape to production but are happy that we are fully focused on continuous improvement.

# References

1. Iron Chevsky Wine Blog http://www.chevsky.com/

2. Hermetic Servers, Chaitali Narla & Diego Salas, http://googletesting.blogspot.com/2012/10/hermetic-servers.html

3. Stress Tool http://people.seas.harvard.edu/~apw/stress/

4. Network Emulation http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

5. ACC Explained https://code.google.com/p/test-analytics/wiki/AccExplained

6. Phabricator, an opensource, software engineering platform http://phabricator.com

# Developing Heuristic Based Mobile Apps

**Pooja Sinha, Samrat Dutta, Sundaresan Krishnaswami**

sinhapooja2004@gmail.com, samrat.dutta501@gmail.com, sundaresank@gmail.com

## Abstract

Phones have evolved exponentially in every aspect in the last few years, from a traditional landline to cordless to mobile to smartphone, it has improved in terms of capability, storage, features, components, but above all smartness.

The question arises are smartphones really smart by themselves? Can they solve their own problems? Can they monitor, analyze, repair or archive things without human intervention?

A smartphone is no longer just a calling device, but is used today for almost everything, and this is made possible by, first, the phone itself – with its high-performance yet optimized hardware, and its Operating System – which has opened an avenue for smartphone applications to make the best use of the phone's features while greatly enhancing user experience.

They are several layers of data available on a smartphone that can help both device manufacturers and application developers to leverage the inherent smartness for a myriad of applications such as security, usability or performance.

Our paper is a collaborative effort to analyze such heuristics already applied or researched in the field and present it as a quick reference for product architects to apply it in identity protection, threat protection, optimize resource usage on a smartphone, which otherwise applications may take it for granted.

## Biography

*Pooja Sinha is a Software Engineer at IBM Software Group, currently working for MaaS360 device management products in Bangalore, India. She is a domain expert in cloud based applications, who has extensive experience in testing large-scale database transactions and analytics. She has also lead the persona driven mobile device management solution for MaaS360 and continues to drive the 3rd party integration points for MaaS360Pooja has a Master's degree in Information technology from IIIT-Bangalore.*

*Samrat Dutta is a Software Engineer at IBM Software Group's, storage division. He is a software technology enthusiast and interested in opensource projects and has submitted multiple works for Pattern Language of Processing conference, Apache Tomcat along with several whitepapers for IBM's storage portfolio, mainly in the fields of Cloud, Mobile and Software Quality..*

*Samrat has a Master's degree in Information technology from IIIT-Bangalore.*

*Sundaresan Krishnaswami is a Staff Software Engineer at IBM Software currently leading the technical testing effort for Android ecosystem at MaaS360. He has over 10+ years of experience in the field of software testing, test automation and mobile application testing.*

# 1 Introduction

Every individual is unique; they think differently, their perspectives differ, and they react differently to the same situation.

This holds true for smartphones, too. Users customize their smartphones based on their priorities or to suit better usability.

For example, a child will use his/her phone mostly for educational, messaging or gaming applications, the kid may also overclock the phone’s CPU for gaming. A professional on the other hand may not tweak the hardware while he/she will interested in using productivity applications such as emails, creating documents, customizing notifications or simply browse the Internet. A developer may use the phone for developing or testing applications with minimum applications running the background while several developer options enabled on the device.

Let us illustrate this using an habitual example: After waking up in the morning, almost everyone has the habit of checking their phones, some may access email application and spend some time checking their social networking statuses, plan their day using a to-do app or key-in their notes for the day. The behavior in usage may change depending on the day and time, but tasks become habitual over a period of time.

## 1.1 Making use of usage patterns

The major concern over buying a smart phone is security. Several questions arise when the phone falls into the hands of an unintended user.

Europe’s Union Agency for Network and Information security cite’s Data Leakage resulting from loss of device or theft as the major cause of concern (Top Ten Smartphone Risks, 2010). The study is still relevant and one of the reasons several players are getting into the enterprise Mobile Device Management space, to reduce corporate security risks.

## 1.2 Smartphones are usability tools

With advancement in smartphone electronics, it is easy to make applications that are context aware

Let us take for example an email application. What if the smartphone is left stationary at a location every night and accessed only in the morning.

With location and accessibility data available on the smartphone, the email application can be made aware to stop syncing emails during the night and automatically sync during a time period in the morning.

## 1.3 Smartphones are performance intensive

One of the significant drawbacks of smartphone, albeit the intensive performance it can offer, is the battery life it offers. All activities consume battery power, making a smartphone usable only while its battery lasts.

What if the smartphone could control its apps’ dormancy or what if the apps could actually make use of the data available through the user behavior to modify its performance?

Keeping the above points in mind, we came up with this question -

What if the smartphone with a given set of data could do the following?

a. Block an unintended user from using the phone
b. Leverage the usage pattern to optimize application performance
c. Enhance usability by displaying relevant notifications

To achieve this, we came up with a solution that forms the following framework:

1. Gathering data
2. Data collection methods
3. Creating the activity map
4. Application

# 2 Gathering data

Data can gathered from logs, settings, events, triggers, alarms, back ground and foreground activity timeouts among others can be helpful in creating the activity map using heuristics.

While we would like to generalize as many data points as possible for any given operating system, we may tend to gravitate towards Android given its openness to the application developer.

Table of reference for user interaction points

| **Interaction points** | **Common data points** |
|---|---|
| Swipe | • Frequency of use<br>• Maximum usage in a given time period<br>• Minimum usage in a give time period<br>• Applications sensing or using a particular behavior (example, use of the accelerometer or location services)<br>• Applications sharing data<br>• User behavior after using an application or a service<br>• Profiling steep deviations from regular usage |
| Single tap | |
| Multi-touch | |
| Headphone use | |
| Location,change in location | |
| Sharing | |
| Connectivity | |
| Typing, typing speed | |
| Volume | |
| Voice Quality (Prof. Emily Mower Provost) | |
| Call | |
| SMS | |
| Network usage | |
| Accelerometer (Weiss, 2013) | |
| Navigation | |
| Phone on/off status | |
| Screen lock | |
| Charging status | |
| Accessibility | |
| Brightness | |
| Accessing notifications | |
| App preference or settings | |
| Rooting or Jailbreak status | |
| Stylus use | |

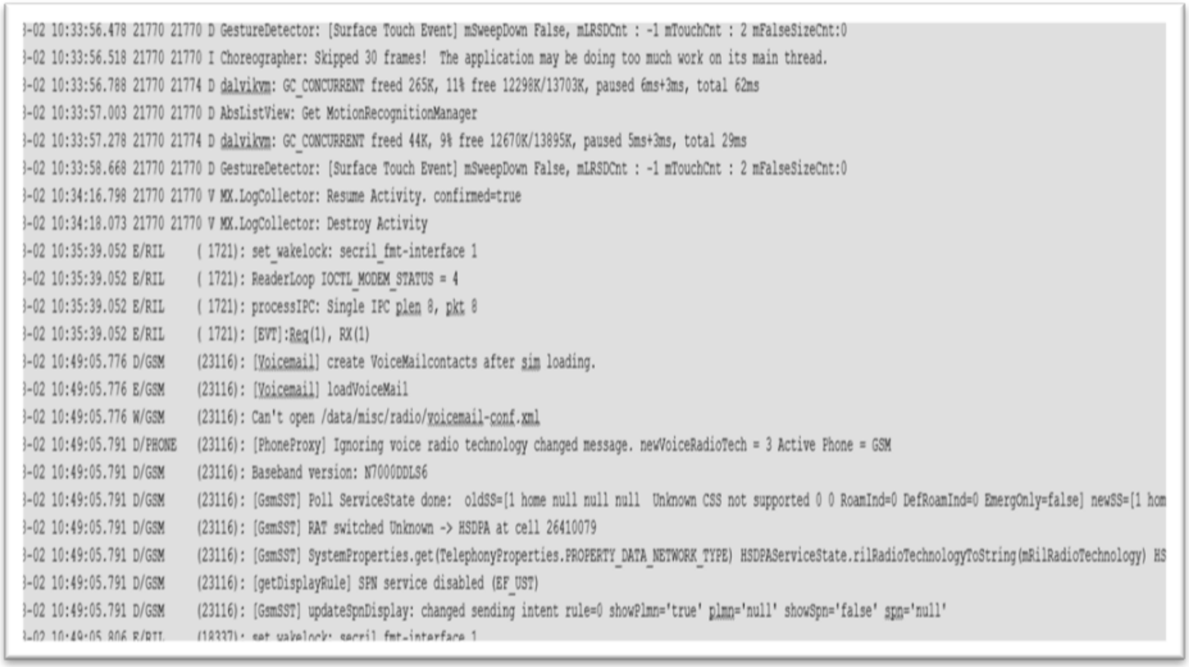
# 3 Data collection methods

There are several sources for gathering this data, the data collection methods can vary between operating systems.We would like to quote a few examples in the table below.

| **Data collection methods** | **Specfic examples** |
|---|---|
| Logs | Analyzing usage pattern from logs is discussed at length on page 5 |
| Application broadcasts | In this paper by Erika Chin et al (Erika Chin), they discuss about how securing inter-application communication on Android. This example could be used in ways application interaction can be captured via Application Broadcasts and analyzed. |
| Accelerometer / Gyroscope sensor data | In this paper by Brunato et al (M. Brunato), how machine leaning can be used to detect early Parkinson's disease, the authors have used accelerometer to study patient behavior. We understand this can be used in understand any user behavior too) |
| GPS coordinates or Location detection using Geo-IP | Service providers such as Trusteer lookup location or IP change on a device to understand user behavior[1] to prevent fraud while using Banking application. |
| Touch sensitivity | Longer unusual user presses can be captured and analyzed for patterns. This stackexchange discussion (Authors) explains how this can be achieved on any operating system. |
| Compass co-ordinates | In this paper by Nirupam Roy et al (Nirupam Roy), the authors discuss about analyzing user's everyday walking direction based on smartphone's compass co-ordinates. This can be used in building a heuristic to track unusual activity on days the smartphone is carried in a different direction. |
| Network switching pattern | In this paper by Kevin S Kung et al, (Exploring universal patterns in human home/work commuting from mobile phone) the authors have successfully used big data analytics on cell phone data to understand user commutes on a given day. |

[1] *http://www.trusteer.com/blog/real-time-credential-theft-your-risk-engine-won%E2%80%99t-catch-this-one*

### 3.1 Analyzing the usage pattern from logs



```
3-02 10:33:56.478 21770 21770 D GestureDetector: [Surface Touch Event] mSweepDown False, mLRSDCnt : -1 mTouchCnt : 2 mFalseSizeCnt:0
3-02 10:33:56.518 21770 21770 I Choreographer: Skipped 30 frames!  The application may be doing too much work on its main thread.
3-02 10:33:56.788 21770 21774 D dalvikvm: GC_CONCURRENT freed 265K, 11% free 12298K/13703K, paused 6ms+3ms, total 62ms
3-02 10:33:57.003 21770 21770 D AbsListView: Get MotionRecognitionManager
3-02 10:33:57.278 21770 21774 D dalvikvm: GC_CONCURRENT freed 44K, 9% free 12670K/13895K, paused 5ms+3ms, total 29ms
3-02 10:33:58.668 21770 21770 D GestureDetector: [Surface Touch Event] mSweepDown False, mLRSDCnt : -1 mTouchCnt : 2 mFalseSizeCnt:0
3-02 10:34:16.798 21770 21770 V MX.LogCollector: Resume Activity. confirmed=true
3-02 10:34:18.073 21770 21770 V MX.LogCollector: Destroy Activity
3-02 10:35:39.052 E/RIL     ( 1721): set_wakelock: secril_fmt-interface 1
3-02 10:35:39.052 E/RIL     ( 1721): ReaderLoop IOCTL_MODEM_STATUS = 4
3-02 10:35:39.052 E/RIL     ( 1721): processIPC: Single IPC plen 8, pkt 8
3-02 10:35:39.052 E/RIL     ( 1721): [EVT]:Req(1), RX(1)
3-02 10:49:05.776 D/GSM     (23116): [Voicemail] create VoiceMailcontacts after sim loading.
3-02 10:49:05.776 E/GSM     (23116): [Voicemail] loadVoiceMail
3-02 10:49:05.776 W/GSM     (23116): Can't open /data/misc/radio/voicemail-conf.xml
3-02 10:49:05.791 D/PHONE   (23116): [PhoneProxy] Ignoring voice radio technology changed message. newVoiceRadioTech = 3 Active Phone = GSM
3-02 10:49:05.791 D/GSM     (23116): Baseband version: N7000DDLS6
3-02 10:49:05.791 D/GSM     (23116): [GsmSST] Poll ServiceState done:  oldSS=[1 home null null null  Unknown CSS not supported 0 0 RoamInd=0 DefRoamInd=0 EmergOnly=false] newSS=[1 hom
3-02 10:49:05.791 D/GSM     (23116): [GsmSST] RAT switched Unknown -> HSDPA at cell 26410079
3-02 10:49:05.791 D/GSM     (23116): [GsmSST] SystemProperties.get(TelephonyProperties.PROPERTY_DATA_NETWORK_TYPE) HSDPAServiceState.rilRadioTechnologyToString(mRilRadioTechnology) HS
3-02 10:49:05.791 D/GSM     (23116): [getDisplayRule] SPN service disabled (EF_UST)
3-02 10:49:05.791 D/GSM     (23116): [GsmSST] updateSpnDisplay: changed sending intent rule=0 showPlmn='true' plmn='null' showSpn='false' spn='null'
3-02 10:49:05.806 E/RIL     (18337): set_wakelock: secril_fmt-interface 1
```

**Figure 1**

Line 1 in Figure 1 log explains the user gesture activity *mSweepDown* which is set to False, based on this data the 'List view' performs up or down motion on the device. This can give us the user's gesture pattern.

# 4 Creating the activity map

There are several data interaction points that a smartphone's operating system provides that allow us to create an activity map.

The motivation behind activity map:

1. Activity map created using decision trees are proven methods for machine learning
2. It is simple to understand and implement

The smartphone can heuristically derive the usage pattern and create an activity map. This map can be a decision tree.

To optimize creating the decision tree, based on whether this is implemented on the smartphone itself or by an app, it is best that the process should be only executed when the device is idle.

All data logged in the decision tree will have some value; these values are based on the usage behavior of the user.

For example, when a user buys a new phone with this heuristic applied, every action and sequence of operation the user does get stored in the decision tree and the initial weighted value. This will be a decision tree node.

## 4.1 Creating and updating the Decision Tree

Again on the next day if the user performs a sequence of operation that may or may not be matched the previous day's data. Now with further usage on upcoming days if the data matches with its initial nodes, the weight for each node will increase.

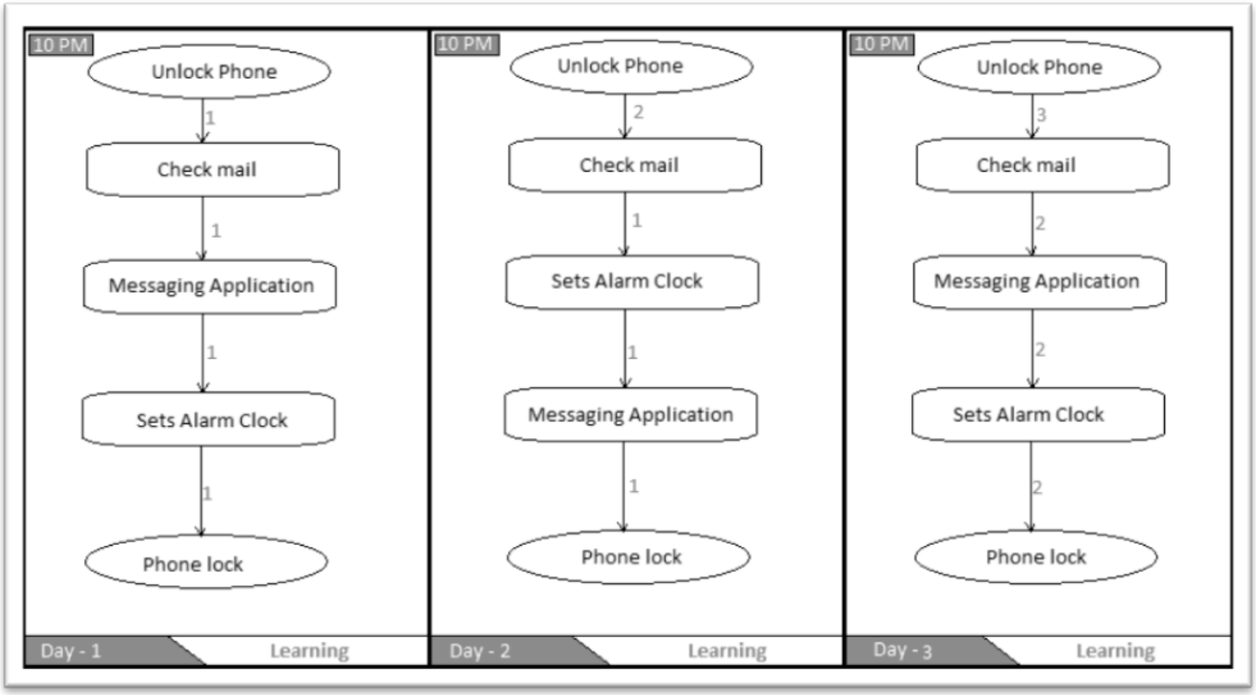
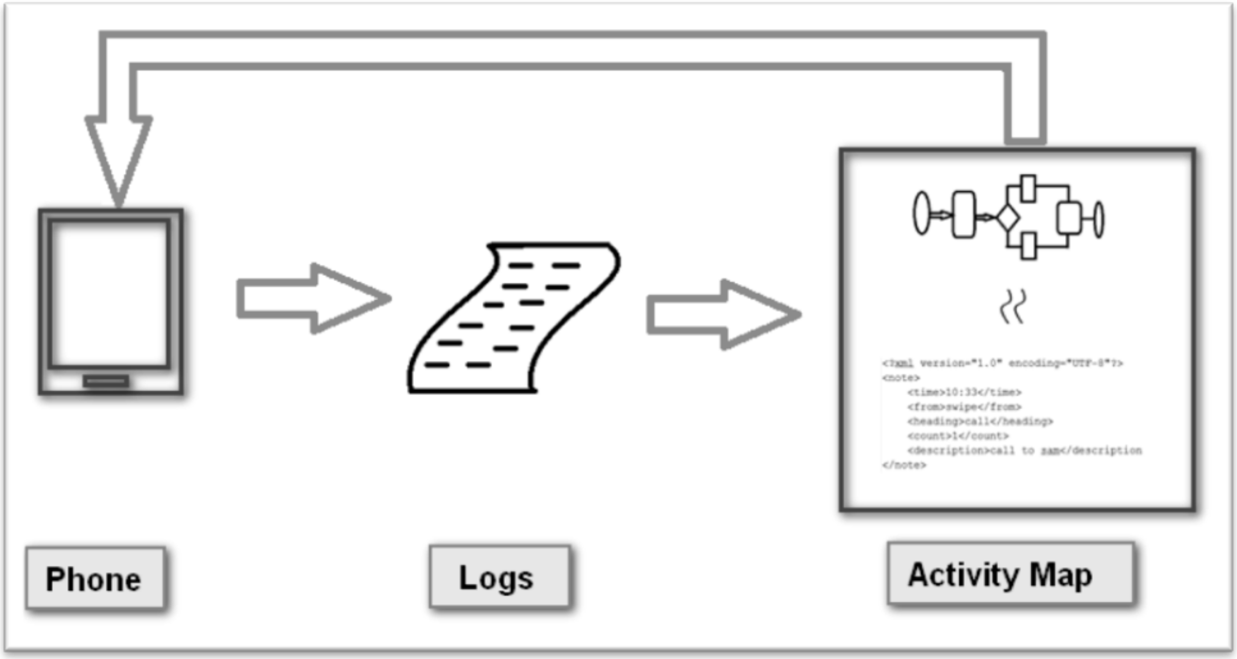


**Figure 2**

Figure 2 explains that the operating system will derive from the decision the node that has the highest value for a given behavior. Any deviation from this node would mean that it may not be the intended user. Based on a set of conditions the operating system or the application can take action.

## 4.2 Building the heurisitics

Let us try to put the ‘Building the heuristics’ aka, building artificial intelligence in a perspective with Figure 3 given below:



```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <time>10:33</time>
    <from>swipe</from>
    <heading>call</heading>
    <count>1</count>
    <description>call to sam</description
</note>
```

**Figure 3**

# 5 Application

## 5.1 Sample application ideas to apply the Heuristics

### 5.1.1 Security

- An enterprise application that manages sensitive data, which can also be prone to attacks, can actually derive the usage patterns to block access to the application if required

- The operating system can gather this data, learn from it and enhance security on the device. A user may be prompted or blocked from device use if the usage pattern deviates from a set of regular patterns.

### 5.1.2 Performance

- An email application can sync user data once based on the usage pattern which was discussed in our example for smartphone as a usability tool.

### 5.1.3 Usability

- A task management application can learn from user behavior, patterns and automatically start suggesting tasks based on the usage

- Data generated from the application from the usage pattern can be used for various analyses that can help in business development. For example, if the e-commerce industry wanted to target the

20-30 age group, and they want to know: what kind of content they are browsing, the most used phone models, and how much time a user spends on an app on a regular basis. This can be derived using the pattern derived from heuristics, based on these pattern e-commerce industry can venture new business ideas.

# 6 CONCLUSION

Security, usability and performance are the hot keywords in the mobile industry. With the advent of technology, everything now has become smart enough to fulfill the user's desire.

From a security standpoint the authors believe that a set of ‘What if’ questions can be answered by building and applying heuristics

Considering the limitations of a smartphone, it is imperative that an application has to be intelligent enough to understand and adapt to a user’s whim to make the app more usable.

The same limitations may apply for performance as well. We strongly believe that logging the patterns discussed in our paper, deriving heuristics from it, and applying it wherever we can, can only make smartphones smarter.

## References

Authors, M. (n.d.). Retrieved Jul 1, 2014, from http://stackoverflow.com/questions/1127682/is-it-possible-to-determine-the-finger-pressure-on-the-screen

Erika Chin, A. P. (n.d.). *Analyzing Inter-Application Communication in Android*. Retrieved Jul 1, 2014, from https://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf

*Exploring universal patterns in human home/work commuting from mobile phone.* (n.d.). Retrieved Jul 1, 2014, from Arvix.org: http://arxiv.org/ftp/arxiv/papers/1311/1311.2911.pdf

M. Brunato, R. B. (n.d.). *KDNuggets.* Retrieved 1 2014, Jul, from kaggle2.blob.core.windows.net: https://kaggle2.blob.core.windows.net/prospector-files/1117/958625cf-3514-4e64-b0e7-13ebd3cf9791/kaggle.pdf

Nirupam Roy, H. W. (n.d.). Retrieved Jul 1, 2014, from http://synrg.csl.illinois.edu/papers/walkcompass.pdf

Prof. Emily Mower Provost, P. S. (n.d.). *Listening to Bipolar Disorder: Smartphone App Detects Mood Swings via Voice Analysis*. Retrieved Jul 2014, 1, from umich.edu: http://www.eecs.umich.edu/eecs/about/articles/2014/app_for_mood_swings.html

*Top Ten Smartphone Risks.* (2010, Dec 10). Retrieved Jul 1, 2014, from European Union Agency for Network and Information Security: https://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1/top-ten-risks

Weiss, G. M. (2013, Jan 4). *Your Smartphone Knows You Better Than You Know Yourself.* Retrieved Jul 1, 2014, from Inside Science: http://www.insidescience.org/content/your-smartphone-knows-you-better-you-know-yourself/904

# To Build an Agile Company, Do Not Begin with Agile Development

**Michael Belding and Phyllis Thompson**

mbelding@shiftwise.net and pthompson@shiftwise.net

## Abstract

In small startup software companies, agility is survival. Companies that rapidly put limited resources on the most promising opportunities and customers are able to survive and grow. Companies in this stage do not typically have significant process and rely on people. In a small organization, engineering, sales, finance, and leaders communicate every day and freely share information. The focus on getting a product out to market where it can pay the bills, plus the ease of engaging decision makers, minimizes documented design work. If a startup is fortunate enough to grow, more people and the demands of customers will begin changing the landscape. People who used to communicate constantly now have more demands on their time. Newer employees have to fit into a culture where common cause is supplanted by specialized roles, and new programmers are expected to be instantly productive with limited documentation and limited training time.

At this stage small companies consider developing a process to address these competing interests while still growing the company. Agile methods such as Scrum, Lean, eXtreme Programming (XP), or Test Driven Design (TDD) arrive with a promise to provide just enough process to allow for growth while keeping the company agile.

This paper focuses on the experience of ShiftWise; a medium sized software company that grew from tiny startup to market leader, but also from an agile company to an agile development organization and then back to an agile company. It explores the cultural and business forces that both facilitated and hindered the simple goal of being an agile company.

## Biography

*Michael Belding is the Vice President of Engineering at ShiftWise in Portland, Oregon. Mike started working with Software as a Service back when providers were called "Application Service Providers" and has worked with Software as a Service in health care, transportation, finance, and corporate governance sectors. Mike has built highly responsive and innovative teams and coached them to deliver new products such as ShiftWise Connect, while continuing to develop and improve existing VMS offerings. Mike holds a BS from the University of Oregon.*

*Phyllis Thompson is the Agile Process Coach at ShiftWise, where she is has participated in the company's agile adoption since 2012. Phyllis has worked with agile teams for nearly 10 years in a variety of roles: Scrum Master, Agile Project Manager, Product Owner, and as the manager of a PMO (project management office) that rolled out an "Agile PLC" and Scrum/Agile training to more than 100 engineering team members at Serena Software, which used Scrum to develop an agile project management tool.*

# 1 Introduction

In 2009, the technology leadership of ShiftWise was beginning to see signs that the company was transitioning from pure start-up mode to a more mature company. They began an evaluation of their capabilities and resources in preparation for addressing some issues that were preventing the company from making progress on new product offerings.

Initially ShiftWise had attracted customers who were early adopters of a new technology, which allowed them to automate their business processes, and they were willing to live with inconsistencies in delivery schedules and quality in order to get the new technology. However, after six years in business, ShiftWise was beginning to engage with new customers who had low tolerance for quality issues and much higher expectations for regular delivery of significant new features, and ShiftWise wanted to more confidently anticipate what the new markets needed and grow with those new markets.

The challenge was that the development team was collection of six individuals (five software developers and a QA engineer) who worked in isolation and did not think as a team. Each person had his or her own programming style, development philosophy, and own area of interest in the code. The company could attack five problems at once by assigning each problem to a developer to work on in parallel, but the efforts were fragmented. The result was an inconsistent and confusing user experience. There was a concern that perhaps half of each developer's day was taken up with responding to support issues, and a great deal of time was spent debating about what the issue was, whether it was an issue, and how to resolve it.

ShiftWise management felt that ShiftWise had a talented group of developers, but they were not cohesive. The leadership team in engineering had been with the company since its' founding and were proud of the team's reputation for business awareness, flexibility, and dedication, but the group was managed one-on-one, not as a team. The company made frequent changes to the application, but over time, the majority of those changes were to fix issues resulting from other changes. It seemed that fewer new features were created. Plans and ideas to introduce major functionality were postponed in the face of the endless stream of change requests and defect reports. To grow, the company needed to change how it planned, built, and delivered software.

This paper is about the lessons ShiftWise learned as a result of growing from a company of twenty five to a company of over seventy and the journey from an agile startup, to a Scrum development organization, to a mature agile company. This journey represents the combined efforts of many, many individuals over the course of more than five years and the labor of an entire company to make it happen.

# 2 A Solid Foundation

The beginning of the journey proved to be an enormous challenge. The current development team had established the current collection of practices and it fit how they liked to work. They were open to growth, but there was no internal motivation to change the work dynamic. Likewise, the operations groups had adapted to work with the current development structure. Customer expectations, performance targets, and benchmarks were all keyed to how things currently operated. Thus, making significant changes would be disruptive to the business. The challenge was to drive change so ShiftWise could increase its ability to win the new opportunities the company was aspiring to, not to address impediments it currently had.

The first step on the journey was to find that aspirational path. Management started by identifying the impediments to growing the company.

The company leadership identified four significant impediments:

*Lack of* progress *on customer feature requests*

ShiftWise had a backlog of over 3,000 potentially actionable items, including defect reports, new feature requests, feature enhancements, and performance problems, but customers were becoming increasingly frustrated from what they perceived as a lack of progress on those items.

*Complexity and inconsistency in the product*

As the product evolved, changes were introduced to solve specific issues, or address very narrow requests. This meant that configuring the system required significant expertise to align a multitude of independent settings to create the desired experience. Many features leveraged settings used by other features, resulting in areas of the system that were unexpectedly linked in obscure ways. The net effect to the customer was a confusing application accessible to only a small portion of their employees who had been well trained on how the product worked.

*Lack of visibility into release cycles*

The business had no visibility into when features currently under development would be released. A popular example was a time when the Customer Support team found out about a feature release from a customer who was having trouble with it.

*Inability to manage product development*

The business wanted the ability to drive the development roadmap, and confidently set goals and market expectations based on that roadmap.

The one area the business agreed was going well, and was vital, was the flexibility and responsiveness that engineering provided to the Customer Support and Account Management teams. These business units had grown dependent upon being able to call a software developer directly if they needed to mollify a customer or to get an issue addressed quickly, and it was a capability they did not want to lose, even though this flexibility resulted in a disjointed product and generated difficult-to-diagnose defects.

It was clear that, in time, ShiftWise could develop a more process-driven approach to accomplish their objectives, but that any attempt to rapidly transform the organization or make significant changes was not likely to succeed because of the cultural and operational constraints.

The next step was to turn the list of impediments into goals that could be measured. The goals established by the company leadership were:

- The company must have the *flexibility* to pivot to new opportunities or respond to threats in a timely manner.
- The application had to have sufficient *quality* for customers to be confident in it and in the company.
- Decisions had to be made based on the *customer outcomes* to be achieved so that changes were met positively by customers and to ensure that work the company invested in was viewed as valuable.
- Customers and support teams needed *consistent* behavior both from the platform and the development organization, to have confidence that the company was providing the value they claimed.

# 3 First Moves

The next challenge was measurement. ShiftWise was, and in many respects continues to be, a results-driven company where successes and failures against goals are the principal metrics. Incremental or process metrics were not consistently kept. Before the company could change, management needed to know what the development team was actually working on at any given time, confirm if estimates for groups of tasks were accurate, and track where the work requests were coming from. Only then would the company understand the effect any change would have, and whether the changes were positive.

Management asked the development team to start measuring their work, but this request was not well received. This group of developers had established a set of individual practices that worked well for them and met the organizational goals as they understood them. While management could have imposed a requirement that the team begin tracking work, doing so would have created tension in the organization and an adversarial relationship instead of a productive one. There was even a risk that it might lead to attrition, and the company was not in a position to take that risk at the time. The company decided that transformational change would fail if it was forced or imposed. One of the managers likened the effort to taking a long journey with your family. If they wanted to go and were excited about the journey, they could tolerate a lot of hardships along the way. If they did not, you could anticipate a lot of “are we there yet?” comments. This idea evolved over time and became the first rule ShiftWise would establish to shape change management in the company. The rule became:

***Bring people along on the journey***

At ShiftWise, this has come to mean that whenever a significant change or endeavor takes place, the people driving the effort are responsible for ensuring that everyone who participates in, or is impacted by the effort, is fully committed to, and supportive of, the effort.

As noble as the idea was, the lack of transparency still remained and had to be addressed. It was clear from the reaction of the development team that a major change was unlikely to succeed so the management team settled on making smaller changes. They started working with two members of the team who were the most supportive of the effort. Those developers were asked to do their normal work routine, but every two weeks to list out what they thought they would do for the next two weeks. They also tracked their work in a ticket system, something nobody was doing with any consistency, so it was possible to see how they spent the time.

It was not a major change, but the journey had begun. After a month of tracking, the data showed that about 50% of the time, these two developers worked on what was called “Tier 3 Support” items, which were calls from the Customer Support department that were deemed serious enough to warrant immediate action. Issues ranged from running custom reports, to fixing bugs, to researching various behaviors in the application. Tracking all the work items validated and quantified the Company’s belief that a significant portion of development time (50%) was spent on operational support issues. Of the remaining time, about 25% was spent working on planned tasks and the other 25% was spent on ”refactoring” which generally meant rewriting the code of other developers so it met standards as they individually defined the term.

These two-week iterations demonstrated to the development team that engineering could add minimal process (“record what you plan to do and track what you did”) without the feared consequences, and more members of the team agreed to start tracking work. It also showed the business how little actual capacity the development team had for new feature work. This information gave the business insight into the schedule implications of various types of requests and the 50% of engineering capacity “tax” that the current production support burden placed on new development work. The development team was still not committing to complete specific deliverables, but the business could see work moving through engineeering and it provided them confidence to more accurately message upcoming product changes to customers.

Based on this success, ShiftWise decided to continue the approach of making changes in small steps to avoid pushback, and adopted a second rule for the journey:

***Move in increments***

This rule goes beyond programming in increments, but also to making organizational changes in increments. It meant that each change had to be executed in a way that enabled the product planning, development, delivery, and support groups to internalize the change and the company leadership had to be able to measure whether the desired result had been achieved. It was a slow way to go, but it was important to keep all the groups positive about the journey while minimizing the journey's impact on them.

# 4 Major Changes

Over the next two months, engineering began to deliver on a predictable basis. The small process changes had given the business more visibility into development work in progress, and proved that engineering could deliver work more consistently. Equally important, because the changes affected only the 25% of engineering's capacity that was directed at feature work, the department's flexibility and responsiveness to customers was retained. Also, because engineering management had not significantly changed how the developers worked or were organized, the developers did not object to the effort. This incremental approach showed progress toward the goals without creating resistance.

These successes, and the care that was taken to maintain practices the company believed were critical to the business, brought broader support for the effort. The journey now included the executive team, operations teams, and the engineering team, and had demonstrated success with the goals. The company was ready to build on this success, but they still had what turned out to be an insurmountable obstacle: the development team members continued to work in silos.

There was a very important new feature on the roadmap that was so large that one individual could not deliver it in a reasonable amount of time; the whole team had to work together on it over multiple sprints. The management team focused on creating an environment where the developers could work together as a team and collaborate on a design to deliver the big new feature in a reasonable amount of time, but the developers continued to approach the work as individuals. The development team could not agree on the best way to divide the work, or which approach to take. In more than one case, a developer settled an issue by just writing a block of functionality as they saw fit so they could claim that part of the project before the team reached consensus. The designs changed often and the total effort ballooned into a nine-month project. Eventually the cost and the disruption the effort had created became greater than the value it offered and work on the feature was cancelled.

The leadership team regrouped to consider how to proceed. ShiftWise was committed to growing the company, believed that the journey was the right one, and were pleased with the initial results. They valued the development team, but the spectacular failure while attempting to build a large-scale feature presented them with a dilemma. ShiftWise could keep the people, or continue the journey toward growing the company; there was no obvious path to get both. To grow, the development organization had to either work as a team or the company needed a development group that could. The company believed that with continued effort, it was possible that the developers might be compelled to work together, but that they would not choose that path, and to attempt a top-down change would be a strategy without any predictable end point. The company determined the most viable path was to replace the development team.

This surprising step gave birth to the third rule:

***Be able and willing to make big moves***

No organization would consider something this drastic without weighing the cost, consequences, and disruption that would occur. This big move was not capricious or impulsive. ShiftWise had embarked on a journey to facilitate growth and evidence that a radical change was necessary was clear after six months of data collection and also from the six years the executive team had spent in the current organization.

In keeping with the previous rules (“bring others along on the journey” and “move in increments”), the replacement of the development team had the support of the company’s leadership. To help manage risk, they planned the transition to happen over twelve months,

The leadership team hired an experienced Agile Product Owner and immediately replaced three development team members. The remaining three development team members were offered incentives to stay on during the transition. The engineering manager hoped that the three remaining developers would integrate into the new organization and stay with the company. Over the twelve-month transition period continued issues with collaboration, transparency, and trust forced the company to abandon the effort to integrate the remaining original developers and instead complete the transition to an entirely new team.

The next task was to address the company’s 3,000 item backlog and a separate bug tracking list. These represented every known new feature request, bug report, and idea captured over the last six years. The difficulty was that there was little consensus on the correct behavior of many features. Getting agreement on whether something was a defect or correct behavior took considerable effort. As for feature requests, it was not possible to determine which described capabilities ShiftWise wanted to introduce, which features had been superseded by other work, and which would still be valuable to customers.

Rather than attempt to reconcile these lists, the executive team elected to discard them. At that moment, the application had no baggage; it was doing exactly what it was supposed to do, and was working as it should. Going forward, all new feature requests, change requests, and defects would be written as stories in the form *[As a <type of user>, I want <some goal> so that <some reason>.]*.[1] An unintended but beneficial effect was that all stories were now easily consolidated into a single prioritized backlog that the new Product Owner could manage. ShiftWise used stories to describe the desired behavior of the working application rather than documenting what was wrong with it as a defect, and eliminated the bug tracking list and the time-consuming bug scrubs that went with it.

# 5 Transition to Scrum

Throughout 2010, the company focused on the transition and building up the new team. After discussing what processes the new team should adopt, the company decided Scrum best fit the journey. Scrum allowed the organization to work incrementally, encouraged collaboration, and had a wealth of available resources. The whole team went through an in-house Scrum training and spent the first 90 days working on small items to quickly build up experience. The Product Owner presented a body of work for the team to address. The team provided estimates and then worked in two-week increments, first doing development and then doing testing, until the work was completed and ready to go out at some future date. The work was time boxed but it was acceptable for tasks to carry over between sprints, and for the application to be in a broken state at the end of an iteration. Less than six months after the transition began, the new team delivered its first significant feature, a document management module for the ShiftWise core application. The feature was delivered to the market on schedule and was so successful that it was positioned as a new product offering. It was the first significant new functionality in almost two years, and was the first significant validation of the ShiftWise agile journey.

# 6 Full Scrum

In 2011, ShiftWise embarked on a new green-field product, and a rapid expansion of the development team.

For this new project ShiftWise followed a strict interpretation of Scrum. The company put all new hires through Scrum training, practiced the prescribed Scrum activities, and implemented recommended Scrum and XP practices. Work on the new product started right away because the company leadership trusted that by moving incrementally they could concurrently manage the complexities of architecting a new product, reconciling coding styles, and communicating business goals.

Although it was not immediately apparent, the decision to dive into pure Scrum had some significant consequences:

- The team adopted Scrum measures for velocity, acceptance criteria, test coverage, and productivity. Because the company did not have a history of other measures, the Scrum metrics were heavily relied upon to guide, validate, and provide course correction to the overall effort.
- ShiftWise historically based hiring decisions primarily on technical skills and believed that the culture would evolve naturally when smart people interacted within the Scrum framework.
- The Product Owner established a feature set that was considered “minimally viable” to present to early adopter customers and the teams established a narrow “definition of done” that gave them confidence that they were meeting objectives as measured by the Scrum metrics.
- ShiftWise rigorously followed the Scrum process and inferred that the business goals were met because the Scrum measures looked good, rather than establishing metrics that measured the organizational goals they set at the beginning of the journey. For example, sprints could be closed even if the application was not fully functional as long as all the stories passed the stated acceptance criteria.
- Lack of a working application prevented non-programmers from gaining experience with the product as it took shape. The Product Owner gave demos of the features, but the operations and support teams were not able to gain a deep understanding of the new product or contribute concrete feedback about what might be missing. They were disconnected from the journey.
- The Product Owner was the single accountable authority on prioritization of work and drove all development efforts from the prioritized backlog of stories. This was generally a positive improvement; however, as the target release date neared and too much work remained, pressure built to prioritize the customer-facing features at the expense of features regarded as “back end” or “non-customer facing.” The effect was an application with very complex features, but few tools to manage them or diagnose post-release issues.
- Everyone trusted that by following the Scrum process rigorously they would get the results they expected.

In the fall of 2011, with high confidence provided by the Scrum metrics, but without customer or business validation, ShiftWise launched the large green-field product to early adopter partners, and right away they identified serious issues. Engineering had validated and tested each component, and correctly assembled the application in its minimally viable state, but the modular development approach did not require all the elements to function as a whole until late in the process. The organization had optimized for the most efficient development process at the expense of ensuring that every feature was implemented completely and did not need additional work post-launch to ensure that the whole application was supportable and maintainable. At launch, the new product was overweight on customer features and some of those features were overbuilt and therefore excessively complex for the audience.

The focus on features at the expense of back-end tools meant that the application was thin on diagnostic tools and other utilities that could give the Customer Support team insight into how the application was working so they could help customers who had trouble with the complex features. It was very common for users to make mistakes setting up the new application and there was no easy way for Customer Support to tell what was wrong. Thus it was difficult to determine when a problem was due to a misconfiguration versus an application defect.

Because the whole company had not participated in the journey, the delivery and operations side of the company was not ready to support or mange the product when it came out. Customer frustration grew as more and more issues emerged and confidence in the product faltered. Eventually the Sales organization decided to pause the rollout of the new product and the company regrouped.

# 7 Agility

It took six months to settle the issues with the new product, and another year to evolve the features to find a place in the market. This experience of initial launch using Scrum exclusively led ShiftWise to add a fourth rule:

***A working application is the measure of success***

This rule shifted the measures from process to outcomes. In practice, engineering and stakeholders became accountable to the state of the application, not the mechanics of the methodology. Velocity, unit tests, and test coverage still mattered, but only as benchmarks for the team, not as a measurement of application readiness for launch. Having access to a working application empowered the stakeholders to decide if they could launch and support the product in its current form, or if not, to see clearly what work remained.

ShiftWise now needed to find a way to apply the goals of the organization and also use the rules that resulted from recent experiences. The company elected to stay with Scrum as the development team methodology, but made a number of changes to align to the goals and keep to the rules:

*Shippable code*

All sprints now had to end with a shippable application. If a feature could not ship in a working state because it was not yet complete, it had to ship disabled. This did not mean that every sprint resulted in a deployment to production, but if the company wanted to ship a feature at the conclusion of a sprint, engineering had to be able to do it.

*The Transformation of QA to SDET*

To enable a deployment to production at the end of each sprint, all testing had to be accomplished within the two week sprint. The existing QA model did not allow for this so ShiftWise decided to transform QA. Instead of a dedicated QA department that tested after development finished, ShiftWise made the development teams accountable for testing and embedded Software Development Engineers in Test[2] (SDETs) with each team. The SDETs are automation specialists and team quality coaches. The development teams as a whole are accountable for the quality of the work, not a separate QA department.

*Milestones to define a shippable application*

To address the gap between meeting acceptance criteria at the story level and having a minimum marketable (viable) application, ShiftWise added the concept of milestones to the process and created a cross-functional group called the Design Team to define and monitor the milestones.

The Product Owner has a significant voice on the Design Team but is balanced by empowered advocates for the application. These include representatives from operations, IT, and the application architects. The Design Team members collectively determine what will be required to ship and support the product, taking into account concerns about scalability, security, supportability, and maintainability.

The use of milestones grew out of the company's past experiences with "minimally viable products." Milestones were created to force continuous review of the shippability of any large green-field effort. Today, any new effort that requires more than four sprints to be commercially viable (i.e., has enough functionality to sell), must be broken down into milestones. Milestones define what an application needs to achieve in order to be shippable with the feature set requested by Product Management. There can be one or many milestones before a minimally-viable state is reached, and the milestones inform decisions about future functionality by exercising the deployable, working application.

> Each milestone by definition is a "definition of done."[3] A milestone cannot close until the team can demonstrate, using the working application, that it has met all the milestone goals. In addition, the team must demonstrate that goals from all previous milestones are still being met. For example, a milestone could include back-end work for scalability that is not apparent in the story-level description of the functionality. Use of milestones prevents ShiftWise from putting too much emphasis on the front-end features at the expense of having a viable, supportable application at launch.
>
> At the end of each milestone, Engineering must be ready to put the application into production if the Design Team determines that the application is shippable at that point, and the business wants to release it because the minimally-viable feature set has been achieved.

In keeping with the rules, these changes were slowly implemented throughout 2012 and 2013 and into 2014. During that time the engineering organization grew from two to five agile teams and in early 2012, ShiftWise hired an Agile Process Coach to help the teams with the transition to end-of-sprint releases and to ensure continued attention to incremental process change while the organization grew.

The first new obstacle was strong resistance from the development teams to shipping at the end of each sprint. They did not believe that they could test the application well enough to ship on such short intervals. Past practice had been to package as much functionality as possible into a release because the release process itself was expensive and time-consuming, requiring a lot of complex and error-prone manual processes. Early on, the mandate to ship at the end of the sprint forced the teams to invest a significant portion of the sprint in testing and release work. Over time, the teams adjusted and rather than commit to large stories, the teams committed to much smaller stories with more effort focused on the mechanics of learning to continually release rather than delivering functionality. During the first "release" sprint, the team took four days of the two-week sprint to prepare for deployment to production. Within six sprints they had whittled the prep time down to less than half a day, and the quality of the work improved as they became good at releasing, which created another rule, the fifth:

***If it's hard, do it more***

In the first quarter of 2012, ShiftWise released the application once. After implementing the requirement to ship at the end of each two-week sprint, they released 33 more times in 2012 and 84 times in 2013. For the first half of 2014, the release count was 31.

Before the switch to a two-week release cadence, all ShiftWise releases required several weeks of post-launch stabilization, which included prioritized bug fixes and feature changes, followed by an increased support volume as the feature was adopted. Since the switch, the post-release stabilization period has vanished and ShiftWise no longer sees any increase in support queues post-release. In 2012, ShiftWise required two developers to work full time on Tier 3 production support issues. That workload has decreased to the point where ShiftWise now has one developer who works on production support issues part time.

Some production releases still have post-release issues, usually resulting from the complexity of the manual deployment process. Through 2013, about 17% of all releases had errors that were primarily caused by problems with a code merge or a configuration change between environments. In 2014 ShiftWise has made significant progress in automating this work, which has reduced issues related to configuration errors and merging of code.

A faster pace of releases, but smaller feature list, allowed the business to re-engage and return to the journey. Another benefit of frequent releases was that business always had something to talk about with the customers. Even if the value was small, the customers perceived that ShiftWise was listening to them. The original goal, to remain responsive, was served. The customers perceived value because they were seeing progress, and now the shipped features were more consistent and of higher quality.

As the company has evolved, many of the ideas they held about what makes a great team have evolved as well. Moving from "no process" to "Scrum" to "agile" has been rewarding, but it has been taxing on some team members. ShiftWise's brand of agile demands that developers take greater responsibility for the quality of the application. It requires features to be broken down into small chunks that require coding practices that might be less efficient than if larger stories were planned and delivered, but the goal is not to deliver more, but to deliver software with higher quality and on a consistent cadence.

The engineering teams are coached to understand and express ideas in business terms rather than technical terms and to invest in activities that increase their confidence that the application is shippable, not just that they met the acceptance criteria of a story.

Over the years ShiftWise has discovered that while they have benefited from the contributions of some great technical minds, they have seen far greater returns from individuals who thrive in the ShiftWise Agile environment, even when new hires have to be taught the development technologies and architectural patterns. In multiple instances, ShiftWise has paid a price in attrition, poor quality, low team morale, or cross-department friction when they made hiring decisions based more on technical skills than cultural compatibility. This realization formed the sixth, and last rule:

***Culture Matters***

Simply put, this rule means that when ShiftWise recruits or promotes employees, the attitudes and cultural values of the people come first, and ShiftWise accepts that this may require investment to close any skill gaps that may result.

These experiences, and the learning from them, have enabled company buy-in on a few additional significant moves:

*Product Owners and Engineering share responsibility for the application*

ShiftWise has product managers/product owners, but they function more like product marketing managers. The application is owned by the development teams who roll up under a team lead who is ultimately accountable for ensuring that the application as a whole can meet company goals. This allows the product managers to cover a larger portfolio and to spend more time in the market. The change has eliminated the more familiar Scrum Product Owner role that serves as a ‘throat to choke’[4], and has created a culture of shared ownership by stakeholders who are engaged and committed to outcomes.

*Focus on application health rather than velocity*

In 2011, ShiftWise continually measured team and project velocity and would adjust features to align a project burn down with target release dates. By moving to a two-week release cycle with features being delivered continuously, the company is able to see tangible progress and make decisions based on the state of the product. The regular pace of features being delivered has replaced a focus on burn-down charts with a focus on any gaps remaining in the product that prevent it from being launched. Large efforts contain milestones to force the company to continually review the working application and ensure that there is a balance between customer features, supporting tools, and production delivery infrastructure at the end of each milestone.

Prior to 2012, the engineering teams would typically spend 5% to 10% of their time in a sprint on quality-related work. As of 2013 and going forward, the teams are able to incorporate demonstrations, unit tests, UI tests, integration tests, and test automation into the sprint. They are also able to clear up technical debt[5] before it accumulates. The time spent on feature work has remained fairly constant, but ShiftWise has successfully transitioned from doing quality and stabilization tasks after a release to performing this work prior to a release.

*Whole team ownership of quality increases quality activities*

With the move to SDETs and whole-team, cross-functional ownership of quality, the effort to execute quality-related tasks is distributed throughout the team. This enables the team to complete the whole scope of work within the sprint (development plus test), and also requires the team to meet quality commitments even if the SDET is sick or on vacation.

*Investment in architecture*

About 10% of senior developer time is spent on architecture and proof-of-concept work. In the past, this number was zero.

*Increased investment in new tools and technologies*

The quality standards and sprint cadence have allowed individuals and teams to introduce an array of new technologies and tools. Where ShiftWise was once 100% on Microsoft-built tools, today more than 70% of the code base is comprised of an array of languages and tools that are curated, taught, and nurtured by the teams, giving ShiftWise greater flexibility even as application complexity grows.

# 8 Conclusion

For the last five years, ShiftWise has been on a journey to grow as a company while retaining the agility they enjoyed as a startup. At the beginning of the journey, ShiftWise executive leadership laid out a set of goals that provided guidance for the most important capabilities the company needed to have at the end of the journey. The company established and maintained a set of rules that emerged as a result of the successes (and failures) along the way.

To achieve those goals, ShiftWise switched from a largely organic, minimal-process development practice, to a formal Scrum methodology. From those experiences, the company learned to leverage the formal Scrum development process as a foundation, but blend it with outcome-driven measurements that put the emphasis on what was completed, rather than what was in progress.

This model has allowed ShiftWise to create a product development cadence focused on frequent small releases with clear customer value, and has enabled engineering to double the time investment in quality management to 40% per team, while more than tripling the size of the engineering department from six to 24 in the past five years.

These organizational changes have reduced the time ShiftWise spends on post-release production support from 50% to 1.3% of overall development team capacity, and has allowed incremental enhancements and evolution of the product.

The goal of this paper was to share the story of scaling an agile company by beginning with clearly defined company goals, developing an effective project management process to organize work, building an organization that embraces those values, and providing feedback mechanisms that measure how well the organizational goals are met.

Each company’s journey, goals, and rules will be unique, but the important thing is to define the goals and establish meaningful measurements to help stay true to them.

# 9 References

1 Cohn, Mike. 2008. “Advantages of the “As a user, I want” user story template” http://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template (accessed August 3, 2014).

2 Definition of the role of Software Development Engineers in Test (SDET), from the Microsoft description of the SDET role on their Careers page http://careers.microsoft.com/careers/en/gbl/professions.aspx (accessed August 13, 2014).

3 Definition of Done, from “The Scrum Guide” by Schwaber, Ken and Sutherland, Jeff. 2013. https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100 (accessed August 15, 2014).

4 Cohn, Mike. 2009. “The Fallacy of ‘One Throat to Choke” http://www.mountaingoatsoftware.com/blog/the-fallacy-of-one-throat-to-choke (accessed August 3, 2014).

5 Ramakrishnan, Srinath. 2013. “Managing Technical Debt” https://www.scrumalliance.org/community/articles/2013/july/managing-technical-debt (accessed August 15, 2014).

# The Changing Role of Software Tester

**Anna Royzman**

ari16a@gmail.com

## Abstract

In 2008 my company reorganized into product units and adopted Agile process methodology. As a result, my QA manager position became obsolete. The new reality was not comfortable at first, until, some time and practice later I recognized that my test manager/strategist skills are equally important and applicable to the new role of a tester on the multidisciplinary team. That synthesis emerged into a new role of a "peer leader" which I later identified as a new trend -- through conversations with coaches and thought leaders of our industry. This role is in demand now, and, as I foresee, will be in even higher demand in the future.

I firmly believe that advancement of the testing profession is calling for leaders, fully versed in testing strategies, equipped with the knowledge of psychology and team dynamics, who know how to effectively apply all available team's assets (skills, knowledge, tools, time, etc.) to optimize the delivery of a quality product. The testers in new roles are motivators and educators who can transform every team member into the quality advocate.

In this paper, I will share my observations confirming that the new role is demanded; discuss in which ways it advances testing profession, show how and where it adds value to the Agile development team. I will also assess what the "skilled testing" education entitles, as the traditional view of QA is no longer applicable to the demands of the future.

## Biography

*A context-driven scholar, professional tester for over 15 years and a thought leader, Anna is always on a quest for quality. Her passion is in discovering new techniques and creating environments that allow people to be most effective at what they do.*

*Anna is an international conference speaker; she is the AST Leadership SIG chair and CAST 2014 conference chair. She serves on Community Advisory Board for Software Test Professionals Association.*

# 1 Introduction

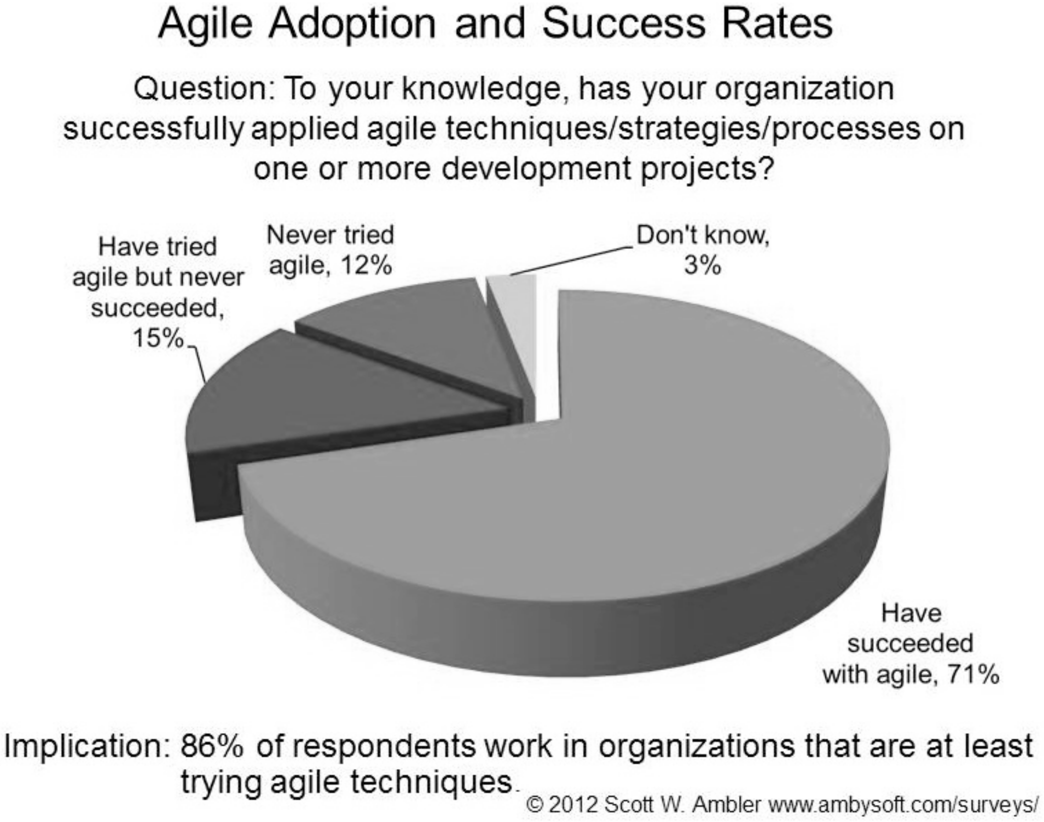
Agile process was largely introduced to technology as a concept of eliminating ‘middle men’ between the ‘user’ and the ‘development team’ in order to speed up the delivery of working software. Such original perception led to misconceptions that resulted in somewhat unsupported claims of calling ‘obsolete’ many disciplines that were traditionally involved in the product development and delivery lifecycle.

”If we replace requirements documents with stories, there is no need for Business Analysts”, “if everyone on a team does testing and we automate all our tests, there is no need for Testers”, “if the Developer shows working software to the customer and gets instant feedback, there is no need for Designers”, etc., etc. – such statements and corresponding actions are still very common for the newly converted Agile enthusiasts.

Practice has shown, however, that the need for skills of the abovementioned specialists still exists, while these professionals would take on somewhat expanded roles in the Agile team. This paper will focus on the changing role of software tester on such team.

# 2 Agile trends

Numerous industry research as well as noticeable infiltration and industry-wide promotion of the “Agile” models (Agile Coach as a new profession, multiple international conferences with ‘Agile’ in the name, Agile track on almost every technical conference) indicate that the practice “crossed the chasm”. The adoption trends that are traced in the diagram below are revealing. While the measure of success is not scientifically supported in this survey, the answers indicate that 86% of respondents work in organizations that are at least trying the agile techniques:

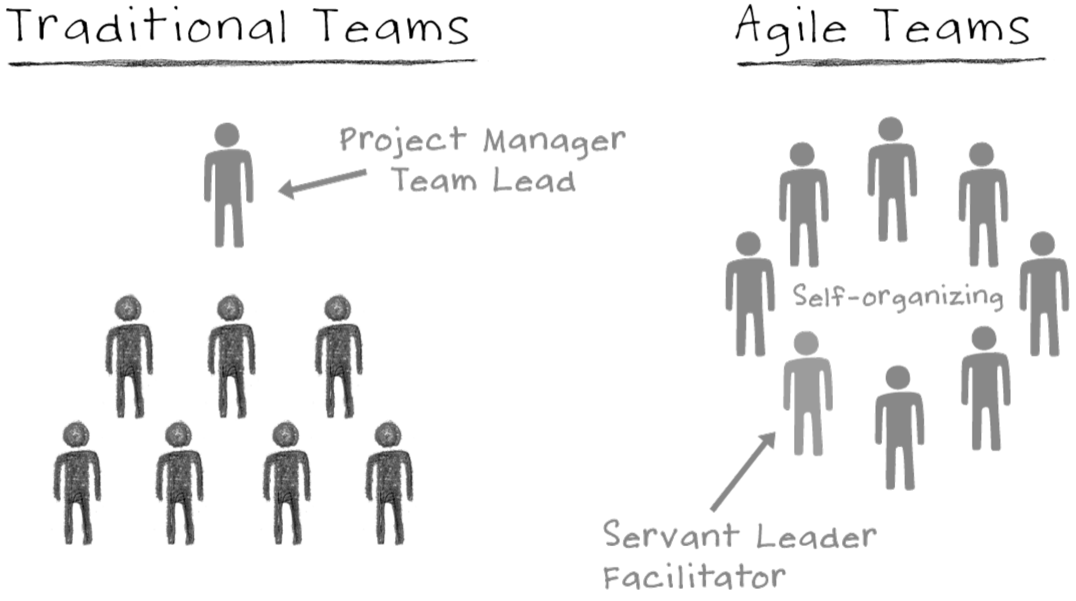


© 2012 Scott W. Ambler www.ambysoft.com/surveys/

Forrester Research data also supports that the agile methods are becoming one of the main development approaches. Through several reports, spanned from 2009 to 2013, Tom Grant, the Senior Analyst at Forrester Research, traced how agile development reached mainstream proportion (Agile Development: Mainstream Adoption Has Changed Agility, 2010; Agile in the Real World: Going Mainstream, Creating Bigger Waves, Making Course Corrections, 2013).

# 3 Self-organizing team – a mini-model of an organization

A foundation of the Agile framework is the self-organizing team. Such a team resembles the "organization in a nutshell" – a mini-model of an organization. The difference is in size and in relationships; while the traditional hierarchical structure leaves all the power and strategy with the superiors, and execution with the reports, the self-organizing team retains both the strategy and the execution with their team members. The leads of various disciplines in self-organizing teams serve the team as strategists, enablers and facilitators, while still being the members of the team. Someone on the team, as in any large hierarchical organization, has to lead the quality efforts professionally.



Picture credit: http://blogs.seapine.com/

Testers, with their unique skills and approaches (such as mastering the art of asking the right questions, not taking anything for granted, analyzing assumptions) and passion for quality are capable of adding much value to the software development teams on all stages of product conception through delivery. When the tester is also armed with the knowledge of various testing methodologies and understands how to apply the test strategy within a unique context, such a tester is better equipped to assist the team to define, and then establish the quality practices.

# 4 Demand for the new role

In 2012, I presented my experience report on fine-tuning testing strategies on product and project maturity to the Mile High Agile conference in Denver, Colorado. At the speakers' dinner later that night, I was at the table with two agile coaches, who at that time were tasked with the agile transformation of large organizations. We started discussing what skills the agile development team needs to succeed in delivering good quality products. One of the coaches mentioned that the company he is working with is having a hard time finding Quality Leaders for their agile development teams. My first intent was to clarify whether they were looking for testers. The coach replied: “We have testers. What we need are Quality Leaders – individuals who can guide developers in adopting good testing practices, and motivate the team in thinking about quality.”

The concept of Quality Leader resonated well with my own experience. In 2008 my position as a Director of Functional Testing was eliminated, as my company reoriented into business units while adopting Agile methodology. I found that my test strategist skills became very important in the new position of the single tester on an Agile team, which then transformed into the Test Lead role. Establishing a test strategy which the whole team could execute required deep knowledge of testing methodologies, along with the flexibility to adapt to ever-changing project needs. Becoming a quality advocate in cross-discipline team was challenging, as it required me to find my own reasons for encouraging quality processes. Making everyone else on the team a quality advocate required my formerly acquired managerial and facilitator skills.

Based on my experience, I concluded that the self-organizing team needs a person who:

- is fully versed in testing strategies
- has the skills to facilitate testing activities and learning
- has the end goal to transform each team member into a quality advocate so that the team can succeed in delivering high value products

When I took my message to testing conferences, I discovered that many testers and test leads find themselves in very similar conditions, where there are no more QA managers, and testers are asked to lead the testing practices, participate in pair testing, serve as test coaches, and help the teams adopt the processes that integrate quality practices. Many testers find the new environment unsettling, as there is a gap between the needs of a self-organizing team and the skills and expertise that many traditional testers bring to the table.

# 5 Skills needed to succeed in the new role

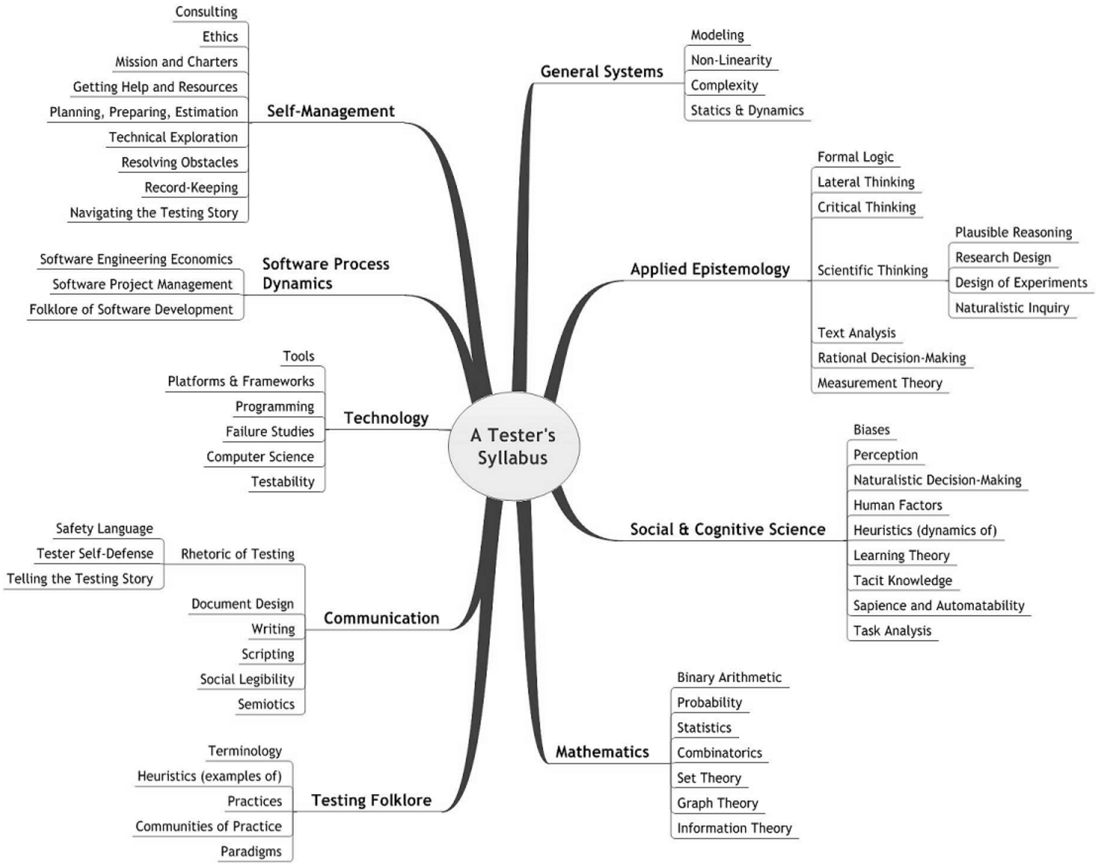
Quality Leader skills largely match the ones of a test manager and test architect who rose from the ranks of hands on testing, with the caveat that:

- The whole team and not only the testers are participating in testing activities

- The Quality Leader is a peer leader; he or she does not have people reporting to them, thus the authority and trust needs to be earned.

The skills required of the Quality Leaders in cross-functional Agile teams are:

- First and foremost, **good testing skills**. A comprehensive list of skills required to be mastered by everyone who performs software testing professionally, is best described in James Bach’s Tester’s Syllabus:



Credit: http://www.satisfice.com/

- **Understanding of test strategies** and knowing how to select an appropriate methodology within the context of project needs. This skill and experience is extremely important in a cross-functional team, where majority of the team members don't have the same level of expertise and may push for an extreme usage of one particular methodology, such as "automate all tests". To provide more light on the given argument, it may be a bad decision to have full automation of a throw-away code as is usually the case in projects in R&D stage. It may be a very costly and ineffective solution to continue 'full test automation' on a mature system that has many different workflows. Maintaining a large and complex automation suite could severely impede development. The better automation solution for large mature systems may be selecting critical workflows and functions for automation coverage, and creating a robust safety net automation suite which performs reliable checks on each new build while leaving room for exploratory testing.

- **Domain expertise, understanding of technology** and **"big picture" thinking**. The Quality Leader has to look outside of working software and develop an understanding on how the software is intended to solve the customer needs, and how it's used by the customers. Learning how the software is constructed adds another perspective to understanding how the system works and where the potential problems or bottlenecks could occur. The person possessing such combined knowledge becomes invaluable in all discussions on new features, architecture changes and other technological decisions. Knowing how the system works inside out and knowing how and why it's used by customers is essential for impact and risk analysis of any modifications introduced to the system.

- **Quality advocacy**. A Quality Leader is one who encourages everyone on the team to advocate for quality. Recognizing all the product stakeholders and what “quality” means to each is a first step of growing into the quality advocate. It is not possible without clear understanding of what each stakeholder group considers to be “quality” and what are each stakeholder’s needs. Speaking about quality with people in different positions requires knowledge of their ‘business/professional’ language and understanding how they think and what’s important to them. For example, a “quality product” means high ROI for the business manager, while it may mean hacker-protected system for security professional, and it may mean a user-friendly interface for customer support.

- **Facilitation and coaching skills**. Oftentimes, the testers on the agile teams are outnumbered by developers. It is not an issue per se, but a reflection of the changed responsibilities of the team members in agile team, where more testing activities are incorporated into development process. Testing skills however are not by default acquired with the new responsibilities. Therefore, it is vital to have a skilled test professional to coach other team members on how to do testing well, facilitate testing sessions, brainstorm test coverage with the team, or identify risk areas in need of deeper testing.

- **Leadership and communication skills.** The Quality Leader is not a manager, but an influencer. In order to change perception of the team members toward quality, the tester in this role has to lead by example and master personal communication skills as well. The leaders inspire people around them to change and improve. Sometimes it’s leading by example and sometimes it’s the art of persuasion or negotiation. In any case, the good communication skills are important in getting your point across.

- **Understanding the process and group dynamics.** From my observation, many testers who spent years in hierarchical organizations have a hard time adjusting into an Agile environment. They are facing team members who at first are not of the same mindset. A Quality Leader needs to be process-literate and be aware of the group forces and dynamics. The uniqueness of self-organizing teams requires flexibility from the leader, as by its nature each group will adopt to change on its own pace. Understanding systems constraints and human learning styles would allow gentle steering toward the end goal instead of dragging the team toward new mindset by brutal force. Changes take time and a good leader is aware of that.

- **Enabling the team** through utilizing all available team’s assets (skills, knowledge, tools, time, etc.) to optimize the delivery of a quality product. In majority of cases, the tester is not a ‘jack of all trades’; he or she does some things well, and is less experienced/skilled in other areas. The output however should not necessarily be limited to a single person’s ability, and the Quality Leader would look around and leverage all existing resources in order to help the team deliver the quality product in shorter time thus eliminating the bottlenecks.

# 6 Conclusion

### 6.1 What is missing?

In my opinion, the biggest skill which is lacking in most testers today is the ability to advocate and facilitate the process where testing becomes an integral part of every stage of the software development life cycle, be it Waterfall, Agile or any other. I mentioned ‘facilitate’ for a reason, as not all testing can and will be done by testers.

Let's look at the bigger picture: validations of business claims, system architecture, functional specifications, software frameworks or support processes are equally important for the project’s success. Testers may not be in the best position to conduct a specific type of validation or review, due to their skillset or number of people on the project, but they should be able to encourage and facilitate the testing and validation processes. To have more weight in such conversations, it's good for the testers to learn the various strategies used by different disciplines to validate their assumptions. For example, UX designers construct little experiments to test business ideas by breaking down these ideas into testable elements, and defining ‘criteria for success’ when doing interviews or eliciting user responses in mock up frameworks to test each hypothesis. In a similar way, good development validation practices include writing unit tests that isolate each part of the program and show that individual parts are correct. Production-like experiments with the failure modes, known as “NASA training”, are used in validating readiness of operations support team to respond to system failures.

### 6.2 Developing new Quality Leaders

I have noticed that the professional landscape is changing in the last few years by observing emerging discussions on testers’ roles and skills at the agile conferences, through blogs, and social media. I have seen various attempts to rediscover and redefine this role. Speaking with many people at conferences, I often hear how difficult it is for a tester to fit into an Agile team and how confused testers are in such environments.

The material presented in this paper highlights that the “traditional” tester’s skills are not matching the new demand. The new role is more advanced, and well-versed in various skills. Without these skills a tester is not adding value to the team and instead becomes an impediment.

I also see that this demand created an outbreak of certifications and training courses in “Agile testing” that often promote excessive usage of automation tools and do not offer much more besides a training in ‘scrum terminology’. There is an obvious gap in providing testers the appropriate training for the new role and my goal is to bridge that gap in the future through raising awareness of the new requirements.

Working in the Agile team has its challenges and its rewards. My biggest incentive as a Quality Leader is seeing my whole team care about quality as much as I do and having an appetite for learning. As an Agile practitioner, being part of the self-developing team is the most rewarding feeling, and it motivates me to be my best as well.

# Creating Alignment During Change Efforts: The Bridge Between Vision and Execution

**Denise Holmes**

Edge Leadership Consulting | denise@edge-leadership.com

## Abstract

Great, you are charged with implementing an exciting new initiative in your organization. Your initiative could be a process improvement effort, a software implementation, a quality initiative, or even a culture shift. The vision is clear, you have the team, you're ready to get started, and you are already anticipating the wins of a successfully implemented project. But what about those times when you start implementing and suddenly face unexpected resistance? The missing bridge between vision and execution is building ALIGNMENT so that people on your team, within your organization, and even in an external client system have buy-in and are ready to work WITH you rather than AGAINST you.

I will share three drivers of alignment that will change how you approach any type of change initiative. You will know how to tactically bridge the gap between vision and execution so that any project or change initiative you sponsor is effectively supported.

This presentation is ideal for organizational leaders sponsoring change initiatives of any type and is also applicable for project leaders frustrated with projects that fall short due to lack of buy-in from key stakeholders.

## Biography

***Denise Holmes**, Principal of Edge Leadership Consulting in Portland, is an executive coach and organizational consultant working with leaders, managers and their teams to replace unproductive conflict and lackluster results with productive interactions and high performance.*

*Denise was a conference author and presenter for the 2010 PNSQC conference.*

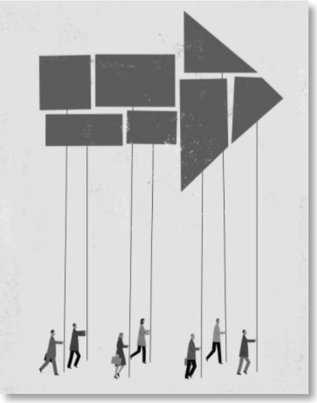
*Denise holds an M.A. in applied behavioral science from the Leadership Institute of Seattle, with a focus on coaching and consulting in organizations. She also holds an M.B.A. from Marylhurst University and a B.A. in international studies from the University of Oregon. She is a member of the Organizational Development Network (ODN) and the International Coach Federation (ICF).*

# 1 Introduction: The Importance of Creating Organizational Alignment for Successful Change Initiatives

Implementations of great software programs tend to be unsuccessful (or more difficult) if the people who are supposed to use the software do not know about it or actively oppose it.  In 2013, I was asked to lead an organizational readiness strategy for a software implementation project that had already failed dramatically. This organization needed to re-start the project because the following factors derailed the first implementation:

- Communication about the project and expected changes did not reach most end-users;
- When the software went live, employees were surprised when they logged onto their computer and found an unexpected user interface;
- Advance training was limited in scope so the few end-users who did receive training still not have the skills to use the program for their specific work processes;
- Employees were upset that many processes and issues they considered important were not taken into account when the software was changed. In fact, the project team seemed unaware of critical processes the software needed to support and manage…and thus ultimately doomed their own project.

The project team members were all smart people with strong technical skills.  They had a clear vision, or picture, of what they needed to accomplish and had detailed project plans for executing that vision.  The missing link was ensuring organizational alignment so that anyone affected by a project or initiative knew what was occurring, why it was important, how they were going to be impacted, how it was going to be implemented, and how their knowledge and concerns would be taken into account. This example may seem extreme, but leaders of strategic initiatives, process improvement efforts, or any kind of technical implementation often underestimate (or ignore) the people-side of their effort and are surprised by negative emotional reactions and active resistance by the people they are trying to help.



Creating organizational alignment is more than having a communication plan and making sure everyone has the same information.  **Having alignment means that individuals are rationally and emotionally committed to the change effort:  they know what is going on, know what they need to do, agree that the applicable initiative or change effort is a great decision, and are willing to help make it succeed.**

In this paper, I will share how you can build organizational alignment by clearly communicating purpose and vision, creating opportunities for discussion, and providing inspiration so that everyone is committed and moving towards the same goal (Straw, et al.).  I will include common mistakes that get leaders and project teams into trouble and on the path to misalignment.  For examples of practical application, I will share how these concepts applied to the software project mentioned above.

# 2 Clearly Communicate Purpose and Vision

**The first driver for building alignment among anyone affected by a change initiative is to concisely communicate the purpose and vision of the project, including why the project or initiative is important and what the impacts are expected to be on the people, company and customers**.   Your message must be structured in a way that keeps people from becoming overwhelmed by details.

People affected by any change need to clearly understand the logical rationale driving the initiative or project, the desired result, and their role throughout the process. When people do not have an accurate idea of what or why something is happening, the normal human reaction is to make up and assume information, often incorrectly.  Instead of organizational alignment, the result is then confusion, rumors, fear, blame and an unsuccessful initiative.

## 2.1 How to Create Clarity

As you create and implement your communication plan:

- Think about what you would want to know if you were first hearing the details. Remember that you have been exposed to the idea of your initiative longer than most people and are probably taking many details for granted, such as why the project is important.
- Develop a headline for key messages about the project; perhaps even test each headline on a few people for what creates a positive reaction.
- Take time to plan out the top two or three points you want people to remember. Some things to think about include:
    - a) What is the most compelling reason the initiative is being undertaken?
    - b) What are the costs of not taking it?
    - c) Why now?
    - d) Who is involved?
    - e) Who will be impacted by the change and how?
    - f) How can details be grouped together in a way that's simple and logical?
    - g) Where can people go for more information?
- If you have specific audience groups affected by the change in different ways (such as marketing, manufacturing, sales, etc.), create individual messages that are specific to each group. Answer the "what's in it for me" question by communicating the advantages unique to that group and/or the specific pains that will go away as a result of the change(s).
- Avoid using acronyms and other jargon. Find simple language everyone can relate to.
- After you have created a message, use the points consistently and repeatedly in different communication mediums: meetings, email, print, break room monitor displays, etc.

## 2.2 Common Mistakes

The following are some common communication mistakes that create lack of clarity and therefore become barriers to organizational alignment:

- × Assuming the audience already knows why the project is important, so you skip the "why" and go straight to details;
- × Being so excited by technical aspects of the change that you overwhelm the audience with too much information and jargon;
- × Creating talking points on the spot without organizing your key messages, so talking points become unclear and inconsistent. When different audiences receive different or conflicting information, you (and the change initiative) are at high risk for losing credibility;
- × Thinking you have to communicate the rationale just once and people will remember it; and
- × Assuming the same rationale appeals to all the stakeholder audiences.

## 2.3 Application

In the failed software implementation I mentioned in the Introduction, company-wide communication happened primarily via email and did not take into account that many end-users rarely used email and instead used team meetings at the start of a shift as the main communication method. Ongoing communication occurred primarily between the project team members, with occasional progress checks given to senior leadership who sponsored the upgrade. The project leader incorrectly assumed that project members would get input from, and share information with, the functional groups they represented.

In the re-implementation effort, the first goal of the new communication plan was to clearly explain the purpose and vision of the project, and to share this in ways that was specific to different functional groups (Quality, Testing, Finance, Purchasing, and Logistics, just to name a few). In this case, communications also had to address why the project was restarted when it had failed dramatically in so many ways. The same key headlines and important messages were used multiple times and through a variety of communication mediums that included: a launch presentation scheduled at all main locations and at different times to accommodate different shifts, company newsletter, a "talking points" document for supervisors to use in shift meetings, and a continuous presentation that ran on monitors in employee break rooms.

# 3 Create Discussion Opportunities

**The second driver for building alignment is talking with people**. The Gallup Organization scientifically showed that employees believing their opinions count is one of twelve key factors for employee to feel engaged at work (Wagner and Harter). When you create a safe environment for multi-directional conversations that allow an exchange of perspectives, and when you take others' ideas and concerns seriously, people feel respected and begin to have a sense of accountability and ownership for the success of the project. This means you must be open to discussing ideas and concerns not yet thought of by you or the project team.

Holding productive discussions is often the hardest part of building organizational alignment for several reasons:

- It can feel uncomfortable to be questioned about decisions that have already been made;
- It can be messy and emotional if people have a lot of fear and anxiety about the anticipated impacts of the project, or have past negative history influencing their perspectives; and
- For leaders and teams who are focused on completing their tasks, it can seem like a waste of time because scheduling more meetings and talking with people is taking them away from getting things done.

However, soliciting others' concerns and ideas can save you and your project team significant time and expense. You might hear about concerns that if you moved forward without knowing about them, could cause serious problems later on. You might hear good ideas that you could use to improve on the original project plan. By drawing out people's concerns early, you also have a way to address them so that it minimizes more aggressive resistance at the implementation stage.

## 3.1 How to Have Productive Discussions

An important aspect of productive discussion is to make it safe for people to share what they think. This means creating an environment that is relaxed and informal, monitoring your body language so it looks welcoming, and not immediately discounting others' concerns and ideas, especially if you disagree with what they say. Other key tips for effective discussion sessions include:

- Use multiple types of engagement opportunities, designed for specific target audiences that are impacted by your project. The chart below lists various types of meetings that you can use to encourage discussion and active dialogue.

**Examples of potential discussion forums:**

□ One-on-one meetings.
□ Team meetings, with or without the supervisor (choose whichever scenario will help participants feel safe to share what they really think).
□ Focus groups sessions, with cross-functional representation.
□ Casual conversational opportunities, such as in a break-room. Bring doughnuts, pizza or other refreshments and be open to whatever people bring up.
□ Formal question and answer sessions, such as where people can hear a status update and submit questions in advance or during the session. It can be positioned as an "ask the project team" event.
□ Group conference calls / video-conference calls.
□ Topic-specific meetings, such as with leadership to discuss sponsorship needs and hear

their concerns and expectations or meetings with stakeholders where you purposefully invite (or challenge) them to bring up concerns or potential problems that haven't yet been addressed.

☐ A moderated chat or blog site on the company intranet that invites questions, ideas and concerns.

- Develop a way to track concerns people bring up, such as an issues list. For common concerns and questions, you might create a Frequently Asked Questions (FAQ) document and make it easily accessible.
- Say "tell me more," when someone shares a perspective or idea you disagree with or think won't work. Often, being able to share their thoughts fully helps people accept change.
- Practice active listening. Paraphrase and summarize what others say, to make sure you capture it. Empathize with people who seem to be having more fear or anxiety related to the initiative.
- Tell people where to turn if more ideas and concerns come to mind, and where they can learn about project updates.
- Thank people for sharing their perspectives. Let them know their participation in those discussions will help the project be successful. It is also a good time to remind people why the project is important.

## 3.2 Common Mistakes

The following thoughts and actions can get in the way of effective discussions:

- Thinking you have to have all the answers;
- Doing more talking than listening;
- Making a minimizing or dismissive comment every time someone brings up a concern, like "we've already thought of that and this is why it won't work…;"
- Focusing only on executive leadership rather than including those directly impacted; and
- Soliciting suggestions and concerns but not sharing the outcomes or resolutions for the input you collected.

Because one of the hardest parts of encouraging discussions is to actually listen rather than present and/or defend the project, the chart below may be a helpful reminder of active listening skills to use.

## 3.3 Active Listening Skills: Use for Creating Productive Discussions

| Technique | Example | Purpose | How To |
|---|---|---|---|
| **Invite More Information** | *Tell me more.*<br>*Can you give me an example…?* | ▪ Convey interest<br>▪ Get more information<br>▪ To help the speaker see other points of view | ▪ Don't agree or disagree.<br>▪ Use neutral words.<br>▪ Ask questions to get more specific information. |
| **Clarifying or Paraphrasing** | *What I heard you just say was…*<br>*My interpretation of what you said is… is that correct?* | ▪ Test your understanding<br>▪ Let the speaker(s) know you are trying to understand<br>▪ Clarify meaning | Restate your understanding of what the person said. |

| | | | |
|---|---|---|---|
| **Perception Check** | *What you seem to be concerned about is…*<br>*Is that accurate?* | Test the accuracy of your perception of another's meaning | State your understanding and ask for validation. |
| **Empathizing**<br>**(Acknowledge Other's Experience)** | *If I were in your place, I'd be concerned, as well.*<br>*Wow, that sounds like it is really important to you/your team.* | ▪ Acknowledge the other's feelings<br>▪ Help the speaker evaluate his/her feelings<br>▪ Open up communication about feelings | ▪ Imagine yourself in the other person's shoes.<br>▪ Reflect and describe how you imagine the person is feeling. |
| **Summarizing** | *These seem to be the key ideas you've talked about…*<br>*I want to be sure I have the main concerns you raised…* | ▪ Review progress<br>▪ Pull together important ideas and facts<br>▪ Establish a basis for further discussion | List or sum up the major ideas or issues. |

### 3.4 Application

In the re-implementation initiative, a variety of strategies were used to encourage productive discussions, particularly with the goal to hear ideas and solicit concerns. We held one-on-ones with team supervisors, scheduled interviews with functional groups, and brought coffee and doughnuts for informal chats with teams in the warehouse and production areas.  Some meetings had specific questions that were asked, such as "what concerns do you have?" and "what might go wrong?" and other meetings were just opportunities to share updates and answer questions.  The project leader avoided responding to concerns defensively and often used "tell me more" and "let me take that back to the team and follow up with what we find out."  The project lead learned that it was okay if he did not have all the answers.  He could trust the various stakeholder groups to come up with questions that needed answers, sometimes come up with answers themselves that he could use, and then work with the project team and follow up with responses as needed.

We wanted to let people know that their ideas and concerns were actively contributing to the project.  An issues list was maintained in Excel, and resolutions were communicated to the person who first brought up the issue.  If an issue was postponed or determined to not be in the project scope, that information and the rationale were still communicated to the originator.  Ideas were seriously considered and when implemented, the originator was acknowledged and sent a personal note thanking them for submitting the suggestion.  We also wrote project updates for the company newsletters which answered common concerns, shared success stories of suggestions that were submitted and used by the project team, and that consistently let people know where to find more information and who to go to with concerns and questions.

# 4 Inspire Others to Move with You

The first driver for alignment, clarity of purpose and vision, taps into the rational mind so people understand the logic behind the initiative you are leading.  In some cases, if the vision you describe is especially enticing, you may have already started inspiring others to move with you.  The second driver, productive discussions, begins to bring diverse perspectives together and lets people know you care about their ideas and concerns, but people are usually not yet aligned and working together for the success of the project.  That comes with **the third driver for alignment, providing inspiration so that**

**people are excited by what the initiative is doing for them, the organization, and the customers.** When you inspire others, you are influencing their emotions to create optimism and positive perceptions of your change effort. You are creating an emotional state where your stakeholders care enough about the outcome of the project that they feel a sense of ownership to help make it successful. This is very different from just getting compliance, where people go along with the project because they think they have to.

Even leaders who are not typically very expressive and can inspire others. You inspire others by expressing your enthusiasm and passion for the project outcomes and by offering encouragement so people feel good about the work they are doing.

## 4.1 How to Provide Inspiration

To inspire others to care about the project you're leading, you do not have be theatrical or lead a cheer, but you do need to be authentic and able to express emotions through words, tone of voice and body language. Here are some tips to make inspiration work for you:

- Clarify in your own mind why you're passionate about the vision you're implementing. What excites you about it? What feelings do you have when you think about not moving forward with it?
- Find two or three things that will speak to people's emotions. This might be answering the question "what 3 things will we feel best about when we successfully achieve our vision?"
- Consider having a project slogan that can be a rallying point – simple, easy to remember, and based on a shared aspiration or hope of what the project will deliver.
- Take time to acknowledge effort and word towards the goals. Celebrate achieving milestones, beyond just the project team.
- Thank people for their contributions, whether they are concerns that would otherwise not have been addressed or ideas that improve the outcome of the project.
- Know what motivates different audiences, and have a different message for each one. How is each audience's world going to be improved after xyz is implemented?

## 4.2 Common Mistakes

Here are some common mistakes for inspiring others:

- Thinking that showing enthusiasm, hope and even passion for your project and its potential impacts is emotional and will cause people to disengage;
- Forgetting to thank people for their contributions;
- Being so optimistic that you minimize others' concerns; and
- Assuming that what motivates you is the same thing that motivates others.

## 4.3 Application

The project leader and project team members let their enthusiasm for the project show in their words, tone of voice, and expressions. Part of the messaging focused on building stakeholder confidence that this time was different and explaining how it was different. The project members didn't try to hide the failures made during the first implementation, but used them as learning experiences which made them more passionate about doing it right this time.

When hearing concerns and complaints from people, they said "thank you for letting us know" and empathized with the person's point of view instead of saying "yes, but…"

Many of the discussion forums provided information that allowed the project leader to tailor communications specifically to what each group cared most about, such as the specific pain that would be solved by the project, an improvement to how they did their work, or an improvement for the customers.

A key lesson for building inspiration is that the project leader and team members had to find ways to highlight project successes, and to make those successes into human interest stories that applied in practical ways to people's work. One way we did this was to write short articles that highlighted how end-user input and suggestions made positive contributions to the project. In this way, the project team shared credit for successes, reinforced small wins that rebuilt people's confidence in the project, kept the project in people's minds, and helped build shared accountability for project success. Employees on the floor saw and heard that they were taken seriously and that their input had a measurable, positive impact.

# 5 Alignment: An Outcome of a Well-Executed Communication Plan

A communication plan is the map that helps organize how information will be exchanged among all stakeholders. It identifies the communication goals and tactically how those goals will be reached. One of the outcomes of a well-implemented communication plan is organizational alignment, because the plan identifies the key messages and opportunities for discussion and other input. Communication plan elements that help build alignment include:

- Clearly articulated purpose and goals for the communication plan. Some of the goals set for the re-implementation plan were:
    - *Create positive perceptions towards (name of project or initiative).*
    - *Identify consistent key messages and diverse mediums for communicating them, targeted to unique stakeholder groups.*
    - *Proactively solicit and address stakeholder concerns.*
- Communication plan assumptions. Some of the assumptions used in the re-implementation project were:
    - *Communication will need to be targeted to different types of stakeholders as well as to different events and milestones related to the project.*
    - *Multiple channels of communication will be needed: electronic, print, in person – group, in person – one-to-one.*
    - *While some stakeholders may be interested enough to search for information about the project, most information will need to be communicated via a "push" strategy, i.e. – proactively communicated and put in front of people.*
- Communication mediums (e.g. – meetings, email, reporting, in person, etc.) by identified stakeholder groups, including the frequency, the type of information that will be communicated, and the schedule for that medium.
- Event-driven communication strategies, such as at already scheduled company meetings, project phases (e.g. – kick-off, milestone completion, final completion), when issues are resolved, and/or when exciting progress is made.

# 6 Conclusion

When you build alignment, you conserve time and energy. You "slow down to speed up," meaning you take time to explore issues and ideas early in the process that then prevent you from having to make costly fixes or start over. You encourage everyone impacted by the project to have a sense of ownership. Finally, alignment unites and excites people around a vision. Without that sense of excitement, people become bored, exhausted, and demoralized. In some cases, they refuse to accept the change(s) and create workarounds or even sabotage the project so it has no chance of working. Once people understand and make sense of something that they can commit to it. Most people have experienced being on the receiving end of a project or initiative where the prevailing thoughts were things like:

> "Why are they doing this to us again?"
> "If they'd talked to us we could have told them it wouldn't work."

“Another waste of money down the drain for the latest flavor of the month.”
“Whatever!”

By proactively having a plan to build alignment for your initiative, the prevailing thoughts are more likely to be:

“It wasn’t easy, but I know it’s going to work.”
“This is so cool!”
“They really put a lot of thought into this. I’m glad we’re doing this.”
“They took the time to figure out what we needed, rather than just what they thought we needed.”
“This new (widget) will help me do x, y, and z.”

One you have alignment, executing all the steps to your initiative is much easier and likely to lead to success.

## References

Inscape Publishing, Everything DiSC Work of Leaders® Sample Profile. 2013.

James. Straw, Scullard and others. 2013. *The Work of Leaders: How Vision, Alignment, and Execution Will Change the Way You Lead.* San Francisco: Wiley.

Wagner, Rodd and James K. Harter, 2006. *12: The Elements of Great Managing.* New York: Gallup Press.

# Cultivating Software Quality Engineers

**Tim Farley**

Timothy.Farley@cambiahealth.com

## Abstract

How did you learn to be an effective SQA Engineer? Chances are, you didn't learn it in college. Recent graduates might have only a single class or a single lecture about software testing, with no mention at all of activities to prevent defects, enable development productivity, or deliver required product quality. If you're currently working as an SQA Engineer, you probably taught yourself.

This presents a clear challenge to companies needing to grow their SQA Engineering organizations. Few job candidates will have all the needed skills and experience when they begin, and hiring only senior-level SQA engineers is not sustainable. In addition, not all individuals interested in SQA will take the initiative to learn the necessary job skills on their own. Internal training programs can also be a risky investment if SQA Engineers have no incentive to stay with a company in an SQA role, or see an SQA Engineer job only as a stepping-stone to a career as a Developer, Project Manager or Scrum Master.

Cambia Health Solutions recently began a comprehensive program to grow SQA engineering capability and capacity. The initiative ranges from recruiting and cultivating interns interested in SQA, to offering training programs and coaching to newly hired and existing SQA Engineers, to providing a clear career path within SQA to keep people in the profession, help them advance in their careers, and ensure the company benefits from the investment made in them.

This paper presents several elements of the program, including training and coaching for SQA Engineers, developing new job descriptions and career paths, and partnering with HR and hiring managers to identify and qualify good job candidates. Examples of new employee onboarding, test automation training, and career path advancement will be reviewed. Integrating social media elements into the program is also discussed.

## Biography

*Tim Farley has 25 years experience in software quality assurance. He has led teams working on products ranging from critical medical devices to iPhone apps. For the past 10 years, Tim has focused on test automation, tools, and enabling software organizations to improve their predictability, productivity, and product quality. Tim first presented a paper at PNSQC in 1996 about creating "intranet" sites, which has since proven to be a pretty good idea.*

*Tim has degrees in Computer Science and Anthropology from Brown University.*

# 1 Why Aren’t There More SQA Engineers?

For most recent computer science graduates, software quality assurance (SQA) wasn’t something they were taught in school. Unless they were in a specially tailored Masters program, testing and quality assurance may have been covered in a single lecture, or, if they were lucky, a single class on the subject. Their professors may have just assumed that their students picked up quality assurance and testing skills while completing programming assignments.

Some recent graduates may have had an internship that included software quality assurance and testing. Whether they completed their internships with a favorable view of SQA would depend largely on the assignments and mentoring they received. Interns that spent most of their time doing manual testing or were paired with manual tester mentors probably won’t be interested in pursuing a career in SQA. Indeed, some may assume from this experience that manual test execution is all there is to SQA and that they want no part of it.

Many people have fallen into SQA accidentally because they were subject matter experts (SMEs) for the software being developed. While this can be extremely useful for functional testing focused on end user acceptance, it is not especially helpful for prevention-oriented software development activities, or for testing that is focused on software structure. These untrained SQA engineers may not consider enabling development productivity a part of their job and may only focus on final product validation for end users.

To ensure an adequate supply of capable SQA Engineers, companies will have to do more than hope that their staff have picked up the skills to do the job, or that their new hires are each willing to take the initiative to learn their craft. Companies need to start with college outreach to ensure candidates get the right impression about the profession, are given the right assignments during internships, receive the right onboarding when hired, and continue to receive the right training and promotions to keep them in a career in SQA.

# 2 Initiative to Create SQA Engineers

In 2013, Cambia Health Solutions began a comprehensive program to address this need. Cambia Health Solutions, the parent of a family of 25 companies focused on creating a customer-centered and sustainable healthcare industry, initiated several training programs to address the needs of current staff and new hires that may have gaps in their experience with software testing and quality assurance, and college interns that may be considering a career in SQA. The initiative also created a better-defined career path to keep trained staff in SQA positions, and provided opportunities to bring together SQA Engineers across the Cambia family of companies to share best practices, find support among their peers, and build a strong culture of quality.

## 2.1 Identifying the SQA Body of Knowledge

The initiative began by identifying the business needs for SQA and testing. The bulk of Cambia companies practice agile software development, in which responsiveness is a key enabler for efficient delivery. SQA Engineering staff members need to be active participants and collaborators throughout development to ensure errors are prevented and developers receive immediate feedback about their latest work. SQA Engineers need to have a variety of skills to be effective in this role, including the ability to write and understand code, define requirements, improve the testability of software solutions, develop quality and testing strategies, automate tests, and create and use metrics.

Based on these needs, the Quality Engineering & Specialized Testing (QUEST) team, the quality practices team that works across the Cambia family of companies, began identifying the body of knowledge (BOK) that SQA Engineers would need in order to participate effectively on development teams and meet development team member expectations. The BOK was assembled from a variety of sources, including:

- The experiences of QUEST team members
- Review of SQA Engineer BOKs from several sources, including IEEE, ASQ and ISTQB
- Review of SQA and testing professional certificate programs
- Review of SQA Engineering job descriptions from a variety of companies, including Amazon, REI, Microsoft, and Intel.

## 2.2 Aligning SQA Body of Knowledge with Job Descriptions

The QUEST team reviewed the current SQA Engineering job descriptions for Cambia. There were significant gaps in the skills, experience, and expectations for the family of SQA Engineering jobs based on the desired BOK. The progression from SQA Engineer I to II to III to IV seemed haphazard and in some aspects overly specialized, as if the job description had been written for a specific individual.

Anecdotally, this seemed to be affecting the decisions SQA Engineers were making about whether to remain in SQA Engineering roles. Being an SQA Engineer was often viewed as a stepping-stone to another career rather than a career in and of itself. After some time in the position, SQA Engineers would move out of SQA and into Project Manager, Scrum Manager, or Developer positions.

The QUEST team rewrote the SQA Engineering job descriptions to align better with industry standards, improve coherence with company needs, and provide a clear career path within SQA to keep skilled SQA Engineers within that role in the company. The career path was defined as:

- SQA Engineer I – Beginner. No experience in software testing or quality assurance. Cambia expects to train you to do the job.
- SQA Engineer II – Competent SQA professional with broad knowledge of SQA and testing methods, tools and processes. Typically at least 3 years experience.
- SQA Engineer III – SQA Project Leader able to create, manage and execute software quality and testing strategies. Able to coach/mentor other SQA Engineers and continuously improve processes. Typically at least 5 years experience.
- SQA Engineer IV – SQA Enterprise Leader able to identify new business opportunities, initiate improvements and innovate across multiple project teams and Cambia companies. Typically at least 7 years experience.

Cambia Human Resources and the Compensation Committee reviewed and approved the revised job descriptions, then presented them to the Cambia Leadership Team.

QUEST team management reviewed the new job descriptions with each development team manager to ensure that they understood the intent of the new positions and how the new positions supported a defined career path. The QUEST team also recommended that each manager work to define training goals and career objectives for each of their SQA Engineers based on the new job descriptions and career path. The intent was not to have managers identify and demote SQA Engineers that were not performing to new job requirements, but to address skills gaps and ensure continued growth in the profession.

## 2.3 Identifying Training to Support SQA Career Path

The QUEST team identified the training needed for SQA Engineers at each job level so that SQA Engineers could remain effective in their current positions and acquire the skills and experience needed to advance to the next SQA Engineering position. Since each position builds on the knowledge and experience gained in the previous one, it was clear that some foundational training would be required for everyone working in an SQA role. This would ensure that all SQA Engineers have the same understanding of essential concepts and terminology that they could build upon while furthering their careers.

Creating a comprehensive training program was larger than the QUEST team could accomplish on its own within the time available. The QUEST team numbered only 6 at the time, and development of the training materials was only one of many quality practice initiatives being driven by the team. In order to be successful, the QUEST team needed to prioritize which training materials could be provided by 3rd party vendors and which needed to be developed in-house because of their focus on Cambia-specific practices.

During the development of the Cambia SQA Engineer BOK, the QUEST team became familiar with the SQA Engineer certification programs offered by several organizations. The QUEST team reviewed the content of these certification programs against the Cambia BOK and business priorities to see if any of the certification training programs could be used to provide basic SQA training. This would allow the QUEST team to focus on the Cambia-specific practices. There were differences in what was covered by the various certification programs.

- Some programs emphasized auditing
- Some programs emphasized compliance with government and regulatory requirements
- Some programs emphasized only specialized testing, such as security or reliability

In the end, the QUEST team selected the International Software Testing Quality Board (ISTQB) Software Testing Foundations certification training offered by Skillsoft. It covered the essential skills with an emphasis on testing. The QUEST team would then fill in the gaps for newer methodologies, tools and processes, such as Test-Driven Development (TDD) and scrum. SQA Engineers could take the classes online at their own speed, or participate in weekly QUEST-led classroom sessions that included reviews of course content and discussions of additional examples.

Taking the exam for certification was left to individuals to decide for themselves. For Cambia, the top priority was that all SQA Engineers acquire a common body of knowledge. Becoming certified SQA Engineers was not required. For those interested in attaining the certification, the QUEST team offered to pay the certification exam fee for the first 20 individuals that completed the training. Otherwise, completion of training would be tracked as part of individual performance objectives for each SQA Engineer.

## 2.4 Building Management Support

QUEST team management met with the managers of all Cambia development teams to ensure that they understood the requirements for the new SQA Engineering job descriptions and the expectation that all SQA Engineers complete the ISTQB Software Testing Foundations course by the end of 2014. Development managers were encouraged to include completion of the training in the yearly performance objectives for their SQA Engineers. QUEST monitored participation in and completion of the training through onsite class attendance and Skillsoft reporting.

QUEST team management also met with HR and executive management to ensure that training could be completed during regular working hours and to establish the expectation that acquiring new and needed skills was part of the job. The QUEST team scheduled training sessions at a variety of different times, including at lunch and towards the end of the day, to minimize the impact on project schedules.

## 2.5 Creating a Common Foundation for SQA Engineers

### 2.5.1 ISTQB Software Testing Foundations

All SQA Engineers and SMEs involved in customer acceptance testing received Skillsoft accounts for ISTQB training. The QUEST team scheduled weekly onsite and online training sessions to review the ISTQB training videos. These sessions typically ran one hour. Though the Skillsoft training runs about 14 hours in total, the QUEST training sessions ran significantly longer due to the added discussions, company-specific examples, and supplemental topics, including TDD and scrum.

### 2.5.2 Onboarding

Another major component of the training program was to establish a common understanding of fundamental software testing and quality assurance concepts. This was accomplished through a more formal onboarding process for SQA Engineers. Onboarding, which had previously been left up to the individual companies and development teams to plan and execute, would now be a repeatable process with defined content and learning objectives. While targeted at new employees, the QUEST onboarding program was also used to train existing SQA staff. The topics covered were specific to Cambia and helped to establish the “Cambia way of the working” in the minds of the SQA Engineers. Topics included:

- Software Quality Assurance & Testing
- Agile Team Participation for SQA
- Metrics for SQA
- Tools for SQA

The *Tools for SQA* onboarding session focused on the use of Rally for workload, defect and test management throughout Cambia.

Onboarding sessions typically ran 90 minutes and featured several in-house developed 3-5 minute videos to explain key concepts. Discussion questions were developed for each concept, with the discussion led by QUEST team members. While most onboarding session were open to anyone, some special onboarding sessions were requested so entire teams could attend and target the discussions to their particular projects, processes, and problems.

### 2.5.3 Programming Proficiency

While new SQA Engineers were required to be programming proficient, it was not the case for existing SQA Engineers. Many had no programming skills or even an appreciation for how software worked. This prevented these engineers from participating fully in test automation efforts or participating in conversations with Developers about the impact of software changes, design for testability, and regression testing. This lack of programming proficiency would also hamper their efforts to advance their careers in SQA.

The QUEST team began addressing the programming proficiency gap by introducing a series of training courses focused on automated testing. Using FitNesse as the test automation platform, the QUEST team was able to introduce automated testing concepts to non-technical SQA staff and provide a transition from manual testing to automated testing. The series advanced from basic automated test design, to writing test cases, to creating automated tests, and finally to creating the underlying fixture code that implemented the automated tests. The QUEST team also developed several reusable fixture frameworks that allowed teams to engage in automated testing and become programming proficient starting with very little technical knowledge.

Each programming proficiency class ran 90 minutes. This allowed for a reasonable amount of content to be delivered, and also minimized project disruptions and made it more likely that SQA staff could attend. Survey Monkey was used to collect feedback from participants to ensure that the right content was being delivered at the right pace, and that SQA Engineers were learning something that could be applied immediately to their jobs.

### 2.5.4 Getting Started Guides

The QUEST team also developed a series of Getting Started guides to supplement the more formal training sessions and allow SQA Engineers to continue to learn at their own pace. Each Getting Started web page focused on a specific topic and provided:

- Installation & Setup Instructions
- How To Instructions
- Contacts for Further Help
- Related Topics List

The QUEST team developed content for over 30 Getting Started topics, including: Git, Jenkins, Maven, JMeter, Selenium, FitNesse, Automated Testing, Regression Testing, and Quality Strategy Development.

The Getting Started web pages were implemented in SharePoint, and while not implemented as true Wikis, the pages and their content were treated as such and were updated frequently by QUEST team members and other SMEs across the Cambia family of companies. In addition to the Getting Started pages, the QUEST web site also includes templates, examples, recorded training sessions, and a training schedule calendar.

### 2.5.5 Best Practice Exchanges, Meetups & Quality Town Halls

The QUEST team also worked to establish support structures for the approximately 100 SQA Engineers across the Cambia family of companies. There is no centralized SQA team at Cambia. All SQA Engineers are members of project development teams at each of the separate companies. While some Cambia companies are large with many SQA Engineers, others are small startups with perhaps only a single SQA Engineer for the entire company. Making sure that these individuals felt supported in their efforts to grow their skills was an essential part of the program.

The QUEST team created new Best Practice Exchanges (BPEs) and Meetups focused on SQA, Testing, QA Lead Practices, Continuous Delivery, JMeter and FitNesse. These teams met anywhere from once a week to once a month, and were in addition to the weekly ISTQB and Onboarding sessions. The QUEST team also hosted quarterly QA Town Hall meetings where SQA Engineers from across all the companies could meet and discuss a single topic, such as training or metrics, with a series of speakers, exercises, and discussions.

Together, these provided a solid base for supporting SQA Engineer I and II staff.  Support for SQA Engineer III staff included more peer support and coaching for SQA Managers, SQA Project Leads, and specialists in performance testing and test automation. QUEST team members paired with SQA Engineers to provide training, work through problems on their projects, and create quality and testing strategies and test automation solutions. The QUEST team also created space in their team area to co-locate project SQA Engineers learning test automation tools, programming languages and project planning.

## 2.6 Intern Outreach & Hiring Support

Cambia participates in the PDX Cooperative Education Program (PCEP) supported by Portland State University. This program cycles participants through 4 six-month internships at 4 different companies in at least 3 different positions (Development, SQA, and Operations). The QUEST team participated in recruitment fairs, interviewed internship program candidates, and hosted interns on the QUEST team. QUEST team management also met with other PCEP interns working in SQA positions throughout Cambia to talk to them about their experience and aspirations, and present a positive image of SQA Engineering as a career.

The QUEST team worked with Human Resources to screen, interview, and recommend skilled SQA Engineer candidates. The QUEST team provided questions for non-technical staff to conduct initial phone screens of SQA Engineers. The QUEST team also interviewed SQA Engineers for project teams, including project teams hiring their first SQA Engineer.

The QUEST team hopes that these efforts will help to minimize skills gaps by ensuring that qualified candidates are recognized, approached, and hired for full time SQA Engineering positions.

# 3 Impact

It is still early to measure the impact the initiative has had across the Cambia family of companies. However, there has been significant participation in the program since its start in February of 2014:

- Over 100 SQA Engineers have registered for ISTQB training through Skillsoft.
- Over 50 SQA Engineers have participated in weekly ISTQB training sessions.
- Over 75 SQA Engineers participate in at least 1 regularly scheduled BPE meeting.
- Over 40 SQA Engineers have participated in at least 1 Programming Proficiency class, with most taking several classes in the series.
- Most SQA Engineer job posting reuse language and requirements from the Cambia SQA Engineer Job Family Descriptions.
- QUEST team members have participated in screening and interviewing job candidates for positions across the Cambia family of companies.
- QUEST has provided Human Resources with initial screening questions for SQA Engineering candidates.

The QUEST team is currently establishing metrics to track whether this level of participation translates into more capable SQA Engineers that remain in SQA roles at Cambia companies.

# 4 Future Work

## 4.1 SQA Engineer IV Training & Support

Most of the training and coaching to date has been focused on supporting SQA Engineer I, II and III staff. The QUEST team saw this as having the greatest potential impact. The QUEST team expects to expand the focus of the initiative to SQA Engineer IV staff in 2015.

## 4.2 Accelerated Onboarding

The QUEST team is considering combining several onboarding sessions into a day long *Boot Camp* to quickly bring new SQA Engineers up to speed. A day long programming proficiency *Boot Camp* may also be created. The QUEST team is assessing the need for accelerated training for non-SQA Engineers that are engaged only in customer acceptance testing.

## 4.3 Social Media Engagement

The QUEST team is investigating using social media to provide support and motivation to SQA Engineers. Hubbub, part of the Cambia family of companies, provides gamification services to help motive people to lead healthy lifestyles. This includes joining teams to find support for completing activities, competing against other teams, and receiving reminders of goals and progress. The same service could be used to help motive SQA Engineers to complete ISTQB training, complete the programming proficiency series of classes, or attend BPE meetings. Rewards could also be incorporated to increase motivation.

The QUEST team may also post training videos for SQA and testing concepts to social media sites like YouTube. Creating a training concept channel on YouTube would help self-select SQA Engineering candidates and establish in their minds that Cambia is a company with a commitment to software quality matching theirs. Over time, this could increase the flow of qualified candidates to the Cambia family of companies.

Content created for the QUEST team web site for weekly topics and weekly quality quotes may also be made available to other companies as a service. This too would help to establish the Cambia brand for software quality, build common knowledge of fundamental SQA concepts and practices, and ultimately increase the number of qualified candidates interested in working at Cambia.

## 4.4 Self-Study

While many SQA Engineers read blogs about SQA and testing, they are not as engaged in reading books and technical journals. The QUEST team is investigating several options to engage SQA Engineers in reading to learn more about their craft, apply lessons learned from others, and broaden their knowledge of the field. Options include:

- Continuing to populate the Cambia Technical Library with physical books and journals.
- Replacing physical books and journals with e-readers.
- Offering Safari Books Online accounts to all SQA Engineers
- Offering access to the IEEE Digital Library to all SQA Engineers

Currently, the QUEST team only offers reading lists and recommendations through their web site, with links to purchase personal copies of books and reprints of articles.

## 4.5 University Partnership

The QUEST team may investigate partnering with Portland State University for the creation of a continuing education SQA and testing certificate program. Such programs have been successful for the University of Washington and Bellevue College in the Seattle, WA, area with programs focused on supporting Microsoft and other companies needing SQA Engineers experienced with .Net and C#. The QUEST team may also investigate partnering with Portland State University to create an SQA track within the computer science undergraduate or Masters degree programs.

## 4.6 Program Metrics

The QUEST team is creating several baselines against which program progress can be measured, including:

- SQA Engineers Engaged in Creating Automated Tests
- Projects Meeting Milestone Requirements for SQA Deliverables
- Projects Meeting Test Case Management Requirements
- SQA Engineers with Defined Career Path Objectives
- SQA Engineers Transferring from SQA Engineering Jobs
- SQA Engineers Participating in Best Practice Exchanges

The QUEST team will chart the trend for each of these over the next several years using a combination of surveys, queries of our test management systems and automated test repositories, and collaboration with Program Management and Human Resources. The QUEST team will also track the cost of ensuring adequate SQA Engineering staff for the Cambia family of companies, including:

- Cost of creating and delivering training
- Cost of building communities of practice
- Cost of developing, enhancing and supporting SQA career paths, and interviewing and hiring practices

The QUEST team will also work with SQA Engineers and their managers to determine whether these efforts have had any unintended side-effects, including demotions, unexpected transfers from SQA Engineering positions, and any changes in the frequency of SQA Engineering promotions.

# 5 Conclusion

Ensuring sufficient SQA engineering capability and capacity is essential to the continued growth of the Cambia Health Solutions family of companies. This can only be ensured by actively working to attract the right candidates, train them well, and provide a clear career path that keeps them SQA. Cambia has created a comprehensive program to increase the likelihood that qualified SQA Engineers will choose to work at Cambia, will learn to be competent SQA Engineers by working at Cambia, and will choose to stay with Cambia because of their growth potential in the profession.

The program includes training, peer and management support, and the overall development of a culture of quality that recognizes the value of outstanding SQA Engineers. It leverages 3rd party as well as in-house developed training to provide a solution tailored to the specific needs of the business. And it recognizes the good work of SQA Engineers across the Cambia family of companies to encourage others to strive to emulate their success.

This is how Cambia Health Solutions grows SQA Engineers.

# References

## Body of Knowledge

IEEE Computer Society, Software & Systems Engineering Standards (C/S2ESC) Committee, September 2012. "IEEE P730™/D8 Draft Standard for Software Quality Assurance Processes."

International Software Testing Qualifications Board, 2011. "Certified Tester Foundation Level Syllabus."

American Society for Quality, 2008. "Certified Software Quality Engineer Body of Knowledge."

International Institute for Software Testing, "Certified Software Test Professional (CSTP)", http://www.testinginstitute.com/cstp.php#bok (accessed June 29, 2014).

## Online Training Resources

Skillsoft, http://www.skillsoft.com (accessed June 29, 2014).

Safari Books Online, http://www.safaribooksonline.com (accessed July, 27, 2014)

IEEE Xplore Digital Library, http://ieeexplore.ieee.org (accessed July 27, 2014)

## Certificates & Certifications

American Software Testing Qualifications Board, http://www.astqb.org (accessed June 29, 2014).

American Society for Quality, Software Quality Engineer Certification – CSQE, http://asq.org/cert/software-quality-engineer/right-for-you (accessed June 29, 2014).

University of Washington Professional & Continuing Education, Certificate in Software Testing & Quality Assurance, http://www.pce.uw.edu/certificates/software-testing.html (accessed June 29, 2014).

University of Washington Professional & Continuing Education, Certificate in Software Test Automation, http://www.pce.uw.edu/certificates/software-test-automation.html (accessed June 29, 2014).

Bellevue College Continuing Education, Software Test Engineer Certificate Program, http://www.bellevuecollege.edu/ce/ste-cert-overview (accessed June 29, 2014).

## Other Resources

Survey Monkey, https://www.surveymonkey.com (accessed June 29, 2014).

Lean Coffee, http://leancoffee.org (accessed June 29, 2014).

Meetup, http://www.meetup.com (accessed June 29, 2014).

Rally Software, http://www.rallydev.com (accessed June 29, 2014).

SharePoint, http://office.microsoft.com/en-us/sharepoint (accessed June 29, 2014).

PSU/PDC Cooperative Education Program (PCEP), http://web.cecs.pdx.edu/~pcep (accessed June 29, 2014).

# This Would Be Easy If it Weren’t for the People

**Danny R. Faught**

danny.faught@gmail.com

## Abstract

Soft skills are important for software professionals to learn. This paper explores soft skills using a series of vignettes drawn from the author’s experiences -

1) “Collaborative design”
2) “But you didn’t fix my bug”
3) “You’re getting in my way”
4) “You have a big problem, wait, no you don’t”

## Biography

*Danny R. Faught is a 21-year veteran of the software industry, focusing on software testing and customer service. He is a graduate of Jerry Weinberg's PSL and Change Shop workshops.*

*Danny is currently a Senior Software Developer in Test at AgileAssets, Inc. He has a B.S. in Computer Science from the University of North Texas.*

# 1 Introduction

Software professionals encounter frustrations on the job just about every day. Many of these frustrations don't come from the technology that's at the core of their work, but from their interactions with the people they work with. This paper will present a series of vignettes based on real experiences to illustrate a variety of soft skills that have made me more effective on the job.

The vignettes are:

1) "Collaborative design" – accepting design input from a co-worker, and using Myers-Briggs to figure out how.
2) "But you didn't fix my bug" – understanding how our ideas about what's important often differ from those who make the decisions, illustrated by a security bug that didn't get fixed.
3) "You're getting in my way" – avoiding inflicting unwanted help on a co-worker who was in the critical path.
4) "You have a big problem, wait, no you don't" – when following the compulsion to tell a programmer about a major design flaw, "reply all" is not the way to do it.

Jerry Weinberg taught me much about using soft skills on the job. I especially remember the concept of "center, enter, turn" approach that he first learned from Aikido. George Dinwiddie offers a nice description (Dinwiddie, 2007) -

> "Center: Be aware of yourself, who you are, and what you want to accomplish.
>
> Enter: Be aware of the other. Enter their world and stand beside them, rather than in opposition.
>
> Turn: Together with them, turn their energy in a more effective direction."

I often find, however, that I'm redirecting my own energy as often as trying to change someone else.

The stories in this paper are drawn from various companies I've worked with. Some details have been altered in order to avoid sharing proprietary information, and all of the names of the people have been changed.

This talk builds on ideas first published in "Testing, Zen, and Positive Forces" (Faught, 2004).

# 2 Collaborative design

In this story, I had to deal with a challenge to one of my designs.

I was trying to convince the QA team to change their existing functional test automation framework so that it used a keyword-driven approach. These are the things I wanted to accomplish:

- Have automated tests that were maintainable. I felt that this was critical, after seeing so many automation efforts fail miserably because of maintainability issues.
- Use an incremental approach, so that we wouldn't break what was already working, and so that we would get frequent feedback on whether the design was feasible (in short, be agile).
- Develop a good working relationship with my new co-workers and earn their respect.

I was brainstorming the design with John, another test developer on my team, and we were stuck. We were writing our first test in the not-yet-developed keyword framework as a way to illustrate the design. The simplest test we could think of just tested the login feature. After some debate, I had written something like this on the whiteboard for the test case:

```
browser start
app start
app login user-type=admin
```

I was thinking about the big picture, wanting to encapsulate as much as possible into each keyword, which helped with the goal of better maintainability. The validation would be built in to the keyword implementation as much as possible, without having to explicitly put it in the test case. I liked the fact that it was only three lines long. But John didn't like my test, and said that he didn't even think that using the login feature for the test was a good idea.

I valued John's opinion, so I wanted to understand the basis of his objections. I felt that the design I had sketched out was a solid plan for our implementation, but it wouldn't be wise for me to dismiss his concerns if I didn't really understand what they were.

Earlier in the day, I had discussed the test framework design with John and our co-worker, Bill. I knew that Bill's was an "extravert" according to the Myers-Briggs Type Indicator®, and so was I (see Weinberg, 1994, for more on Myers-Briggs). That means, among other things, that we tend to say things out loud as a part of processing our thoughts. At one point, in response to something Bill directed at him, John asked for a moment to think and process it, which is typical of a Myers-Briggs introvert (introverts prefer to process a thought fully before saying it out loud). Out of respect for that, I asked Bill that we put a hold on our discussion to give John his moment of silence, and I resisted our habit of continuing to think out loud any time there's a pause. This interaction put the Myers-Briggs model at the forefront of our thoughts, and that's what helped get us past our test design impasse later in the day.

I realized during our test design discussion that I was thinking top-down, looking at the big picture, without fleshing out any of the details. That's typical of a Myers-Briggs "intuition" thinker. It occurred to me that John might be on the opposite end of the Myers-Briggs scale in that regard ("sensing"), which would mean that he would want to start with the details of something that is clearly implementable before trying to see the big picture. That led us to produce something more detailed on the whiteboard:

```
browser start
browser go $start-page
ui type username user-type=admin
ui type password user-type=admin
ui submit submit_button
```

Now, I feel strongly that most of our tests should not go into this level of detail. I want to encapsulate the interaction with the user interface into higher-level keywords like the "app login" line of my first draft test case. John, however, wasn't able to evaluate this more detailed view of my idea, and frankly, neither could I, until we fleshed the low-level details. In fact, we decided put off implementing a feature that allows for aggregating keywords into the "subroutine" calls that my first attempt at a test case relied on, and this meant we were taking a more incremental approach. We agreed that we would not develop a large number of tests at this low level, but it would be beneficial to write the first several tests this way.

I was glad that I was able to swallow my ego and accept feedback. My goal of earning the respect of my co-workers had led to a better way to achieve my more important goal of following an incremental approach.

I didn't ask right away what John's Myers-Briggs type was. Like many models, it wasn't important if I was right about what his preferred way of thinking was. If the Myers-Briggs model helped me to find a better way to interact with someone, that's really all that matters, and I didn't need to explicitly apply any labels. I've found this to be true with many psychological concepts.

# 3 But you didn't fix my bug

What are we here for, if no one bothers to fix the bugs we're finding? This story explores why we should understand our stakeholder need, and why we shouldn't assume our bug reports aren't useful.

Once, I was doing a quick quality survey of a web-based software application developed by a startup. I found that I could get to the screen to edit the account profile for any account on the system, without having administrative access to the system. I sent the CEO a screen shot of the profile edit screen for his own account. I was surprised to find out that the organization did not plan to fix the security hole in the foreseeable future. It's not that my bug report was ignored – it just wasn't the kind of bug they wanted to know about. This was an early phase startup, focused on building enough of a prototype to attract investors, but they were not trying to build production quality code.

I have been reminded many times since then that I have to tailor my deliverables and my expectations to the current context (see Kaner et al., 2002 for more on context-driven testing, e.g., p. 2, "Your mission drives everything you do"). When I begin testing, I gather information about what kinds of bugs my stakeholders most want to know about. It may be difficult to get a good answer this this in a vacuum, so I will also look in the bug database to see what kinds of bugs have already been fixed, and which ones are languishing in the queue. As I proceed through the project, when I get concrete examples of bugs, I will occasionally get feedback on whether the bug is important to report.

Though I generally prioritize my bug finding activities so that I report bugs that are likely to get fixed, I don't stop there. Usually stakeholders want me to report most of the bugs that I find, even if they're not going to be fixed for a long time, and perhaps never fixed at all. The information provided in the bug reports can be used in a variety of ways, including looking for clusters of minor annoyances in the software, planning bug fixing resources for future releases, including information about known issues in release notes, and handling technical support calls.

There is certainly a place for bug advocacy – sometimes you should be the champion for a particular bug and advocate for it to be fixed. Choose wisely when you do this. See Cem Kaner's "Bug Advocacy" presentation for more on this topic (Kaner, 2000).

Don't be discouraged if the bugs you reported aren't getting fixed. There are several uses for the information you provide in a bug report. Find out all of the things your organizations uses bug reports for, and cater to those needs.

# 4 You're getting in my way

This is a story about trying to inflict unwanted help.

I had a deliverable that had dependencies on the work of two other teams. I got a report from a technical contributor on one of these teams, Jack, that his part was going to take longer than anticipated because of the complexity of the task. I was fairly sure, based on Jack's brief description of the problem, that his solution was overly complicated. I thought that I could make his job easier and get the solution more quickly by giving him a better understanding of what I needed.

I tried to meet with Jack to discuss this with him. However, Jack asked me to include a manager who was involved with the project in the meeting. That would mean I should also invite my manager. I was hoping to have a technical discussion just with Jack, and then give a recommendation to the affected management afterward. I was nervous about turning the meeting into a big affair, having seen some dysfunctional meetings in the organization before. I prefer to solve problems in small meetings. So I consulted with my manager, who requested that I not stir the pot. He was willing to accept a delay in my deliverable, so we agreed that the help I wanted to offer was unlikely to be helpful. I'll never know exactly what kind of brouhaha I avoided, but I'm glad I avoided it nonetheless.

For further reading on inflicting help, see Weinberg, 2011.

# 5 You have a big problem, wait, no you don't

I was sure I had found a big bug, but should I shout it from the mountaintop?

A software developer, David, send an email out to a mail alias to all of the developers and QA staff asking for feedback on a database query he wanted to change. Though I wasn't qualified to answer the question he was asking, I looked at the code and thought I saw a glaring security hole, making the code vulnerable to a SQL injection attack.

I replied to the email and expressed my concern about the security hole and suggested how to fix it. David replied and explained that this was static SQL code, so there was no way for someone to modify it externally, and thus there was no vulnerability. Upon further reflection, I realized he was right, and my suggested fix couldn't even be done in pure SQL anyway.

It was at this point that I heaved a sigh of relief that I hadn't done a group reply. My mistake could have hurt my credibility with the entire development organization and on my own team, and perhaps made it less likely for anyone to pay attention to other issues I tried to raise later.

While it was embarrassing to admit to David that I was wrong, I'm glad I did speak up, because it was a learning experience for me. I think that the better understanding of our product's implementation that I gained was worth suffering a small gaffe in an interaction with just one other person.

Of course, this also reinforced my policy of not doing a "group reply" for an email without carefully considering the ramifications.

# 6 Conclusions

A few common themes emerge from these vignettes –

- Build relationships with your co-workers. Use each interaction as a stepping stone to build on. When a difficult situation arises, you can use those relationships to find your way to a solution more easily.
- Do no harm. Carefully choose your audience when sending potentially bad news or corrections. Sometimes it shouldn't be sent at all. Be bold when there is good reason to share the news.

I've found that the biggest challenges in my career have not been the technical challenges, but working with the people. Whatever technical work you're doing, you're almost certainly going to need to interact with other people along the way. It's not a "them vs. me" sort of challenge. It's not often a simple case of incompetence. It's just other people like me, who are trying to do their job just like I'm trying to do mine. The more you can refine your soft skills, the more effective you'll be on the job.

## References

Dinwiddie, George, July 22, 2007. “Blocking.” http://blog.gdinwiddie.com/2007/07/22/blocking/ (accessed June 16, 2014).

Faught, Danny. March 11, 2004. "Testing, Zen, and Positive Forces". http://www.stickyminds.com/article/testing-zen-and-positive-forces (accessed June 16, 2014).

Feathers, Michael. 2004. *Working Effectively with Legacy Code*. Prentice Hall PTR.

Kaner, Cem. 2000. “Bug Advocacy”. http://www.kaner.com/pdfs/bugadvoc.pdf (accessed June 16, 2014).

Kaner, Cem, James Bach, and Brett Pettichord. 2002. *Lessons Learned in Software Testing: A Context-Driven Approach*. New York: John Wiley & Sons, Inc.

Weinberg, Gerald M. 1994. *Quality Software Management*, Vol. 3: Congruent Action. New York: Dorset House Publishing.

Weinberg, Gerald M. March 29, 2011. “Knowing What to Leave Alone.” http://secretsofconsulting.blogspot.com/2011/03/knowing-what-to-leave-alone.html (accessed June 16, 2014).

# Double Loop Learning: A Powerful Force for Organizational Excellence

**Jean Richardson**

jean@azuregate.net

## Abstract

Double loop learning is built into many Agile processes and is the objective of the lessons learned process prescribed by the Project Management Institute Project Management Body of Knowledge. In Agile, the purpose of retrospection at the end of an iteration or sprint is for the team to inspect its work processes in the last cycle and learn where they can do things differently in the next sprint. This helps them position themselves to deliver more, faster and/or with higher quality.

In traditional project management, the lessons learned process is designed to provide a basis for organizational learning specifically with regard to completing similar projects in the future. Within the human condition, double loop learning occurs whenever an individual changes her mental model, or understanding of how things work or how the world is, based on past experience or education (which includes self-education, such as reading or experimenting) and goes on to apply the new model in the future.

In life—its how we grow and become better, happier, more genuinely successful people.

This paper:

- Describes double loop learning and the theory it is based on.
- Helps the audience become alert to mental models in their environments.
- Introduces Positive Psychology concepts.
- Shows how Positive Psychology and double loop learning can be paired to deliver organizational learning.

## Biography

*Jean Richardson is experienced in Agile methods, adaptive and predictive project management, writing, training, public speaking, and requirements and business analysis. Her background includes thirteen years' experience as a court-based mediator. She brings strongly committed to her practice as healing work. She has a no-nonsense approach to the exigencies of business, and her orientation to work as path and the workplace as a context for character formation combined with her skills in encountering difficult situations result in a wide array of creative consulting approaches to solve frequently encountered problems.*

*Jean has a Master of Arts in Interdisciplinary Studies with an emphasis in Organizational Communication and holds PMP, PMI-ACP, CSM, CSPO, and ITIL certifications.*

# 1 Introduction

All organizations have to wrestle with the challenges of decision latency and decision quality. Decision latency is the amount of time between the moment a need for a decision is identified and the moment the decision is made. Decision quality is the suitability of the decision to the problem to be solved. Double loop learning helps organizations diminish or limit decision latency and improve decision quality through deep iterative learning from everyday work situations.

Double loop learning is built into many Agile processes, and it is the objective of the lessons learned process prescribed by the Project Management Institute Project Management Body of Knowledge. In Agile, the purpose of retrospection at the end of an iteration or sprint is for the team to inspect its work processes in the last cycle and learn where they can do things differently in the next sprint. This helps them position themselves to deliver more, faster and/or with higher quality.

In traditional project management, the lessons learned process is designed to provide a basis for organizational learning specifically with regard to completing similar projects in the future. Within the human condition, double loop learning occurs whenever an individual changes her mental model, or understanding of how things work or how the world is and goes on to apply the new model in the future. This change can be based on experience or education (which includes self-education, such as reading or experimenting).

In life—double loop learning is how we grow and become better, happier, more genuinely successful people.

Fostering this kind of learning results in continuous improvement at the individual *and* organizational level. And, it's not easy. The organizational learning theory that double loop learning theory is based on is clear and uncompromising. It sets a high, almost superhuman standard of reflexive thinking, candor, and commitment to personal and organizational change. Most organizations, and, arguably, most individuals, fail to learn from their mistakes. Yet, double loop learning remains a remarkably rich means of improving our circumstances—and our profitability.

What's missing? Why can't we learn our lessons?

Recently, a new field of Positive Psychology has emerged and promises not only to help already reasonably happy people thrive but also to provide a way of setting an organizational context that facilitates the exigencies of double loop learning in organizations.

This paper:

- Describes double loop learning and the theory it is based on.
- Helps the audience become alert to mental models in their environments.
- Introduces Positive Psychology concepts.
- Shows how Positive Psychology and double loop learning can be paired to deliver organizational learning.

# 2 Organizational Learning Theory

Christopher Argyris and Donald Schon first proposed a model for organizational learning in 1978 in their book *Organizational Learning: A Theory of Action Perspective*. Argyris extended this work over the course of his lifetime as a business theorist and professor at Harvard Business School. He brought several useful and related concepts to business leaders over the years including:

- Skilled Incompetence, whereby sophisticated skills are inculcated in professionals such that those skills serve to perpetuate incompetence in communication and group decision making.
- Organizational defensive routines which are organizational cultural patterns that prevent professionals from confronting failure, experiencing embarrassment about falling short of the mark, and learning from that failure.
- Double loop learning which is a specific sort of learning that goes beyond error correction and works to identify and correct mental models which underlie the thinking patterns and actions that result in errors.

## 2.1 Double Loop Learning

Argyris coined the term “double loop learning” to distinguish it from single loop learning, which is mere correction of error when it is encountered. The advantage of double loop learning is that it can prevent errors in the future—before they occur.

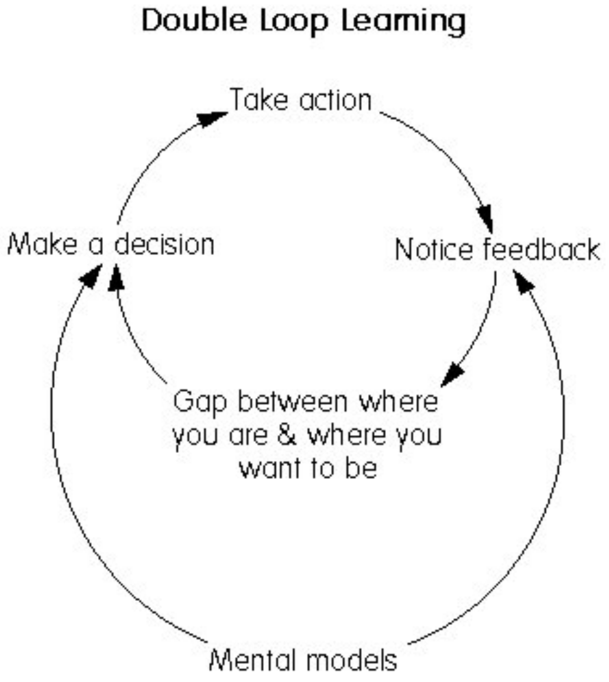


Figure 1 Illustration of Double Loop Learning

As shown in Figure 1, double loop learning consists of two loops, the example above being one of the simplest illustrations readily available. Single loop learning is shown in the top loop—take action, notice feedback, notice the gap between where you are and where you want to be (which constitutes the error), make a decision (which constitutes the fixing of the error), and start the loop again. Single loop learning tends to result in generating strategies and checklists to identify and fix the error more quickly the next time it occurs. Double loop learning adds the lower loop which is initiated at the “notice feedback” phase of single loop learning. At that point, the learner drops down into a reflective or reflexive mode that helps support the identification of mental models that may have motivated the actions that resulted in the error. This loop does not bypass the gap analysis but incorporates it into the reflexive evaluation of mental models.

For example, Team A has external dependencies on Team B. Team B is required by the organization to use a ticketing system to prioritize and track their work. Team A on their planning day dutifully plans their work for a sprint and logs all the tickets they need Team B to complete by the end of the sprint. The next to last day of the sprint arrives and Team A is in a panic because almost none of their requests have been fulfilled. They are very angry with Team B and escalate to management that Team B is blocking their work and that Team A will therefore fail in this sprint. Management immediately investigates and finds that all tickets of a certain uncommon category, which is where Team A's work exists, are being lost by the ticketing system.

Upon reflection, Team A realizes that their mental model said "The ticketing system is the system of record. All work logged there can be expected to be done as requested by the requested date." It did not occur to them that the ticketing system could fail, and that they had the option of walking over and talking to or calling Team B on the phone early in the sprint when they noticed the work was not being started. The lesson for Team A is that next time it appears that an external dependency is not being satisfied as expected, they need to be sure someone follows up in person to verify the target team is aware of the work and is able to get it done as requested. A second mental model that may be motivating how Team A handled their external dependency is that management is more likely to be able to sort out a situation outside of their team than any one on their team is.

## 2.2 Mental Models

Mental models are all around us. They are layered beliefs, assumptions, and structures that we carry around in our heads which help us make sense of the world and navigate it. They can become outdated, be wrong or unhelpful from their inception, or simply be less useful or desirable than other alternative models.

For instance, there was a time when a book was contained in a scroll comprised of animal skin and stored in a stone jar. Over the centuries, our mental model of a book has progressed from something hand written on compressed sheets of wood pulp and sewn together in pages by hand, to stacks of such pages sewn together between wooden boards. Eventually mass produced printed and glued "paperbacks" came into existence. They have been superseded by ephemeral digital representations of type displayed on screens and audio recordings contained on silicon discs. Sometimes now when we refer to a "book" we are actually referring to a digital file uploaded to a device that may be smaller than our forefinger and fit in our ear. All are books, but the mental model of a book has changed drastically over time.

Imagine Thomas Jefferson, who was a great lover of books, being introduced to the electronic images and audio files that comprise books today. He would likely experience tremendous cognitive dissonance and feel a great sense of loss for a period of time as he tried to grapple with the new mental model—no matter how much more efficient it may be to search, annotate, replicate, and share electronic books. Maybe you know someone who went through something similar if they made the leap from music recorded on vinyl to music downloaded as MP3 files and playable through many kinds of devices.

Our mental models can be quite intractable and affect our perception of how things *are*. It can be quite challenging to envision how things *ought to be* and even more difficult to envision how *we* ought to be and then to make that beneficial change to prevent an error in the future. As, for instance, Team A, above, is faced with doing. Their best course of action may be to resort to basic, real time, person to person communication about needs, desires, and technical implementation alternatives rather than feeding a request into a computer system and having that request delivered without interacting with the person doing the work at the other end of the request.

Several years ago I received a real shock while working in an organization where almost the entire population wore headsets all day long and only sat in their cubes and talked to each other in teleconferences, whether the person they were talking to was two cubes down or on the other side of the planet. After receiving a hostile response from someone I had walked down the hall to talk to, I was advised by a colleague that walking down the hall to talk to someone face to face generally indicated a desire to complain or escalate a conflict. The culture had evolved such that face-to-face communication was only used for escalations. It's important to be aware of the mental models both you and the organization are operating from!

### 2.3 Challenges of Double Loop Learning

Argyris's work is virtually unassailable. His thinking is clear and uncompromising. His analysis and description of organizational defensive routines and the desire to avoid embarrassment that motivates them, the cultural structures that make it difficult if not impossible to talk about certain kinds of problems, and the necessity of confronting mental models in order to be a truly proactive and productive knowledge worker is both helpful and accurate. However, it is also daunting and exhausting. On any given day, we could notice several—even dozens—of mental models that could be helpfully changed in order to create a more productive and humane workplace. However, our very human psychology can only confront and make so many changes at a time—usually only two or three. And the emotional work of noticing the gap, searching for the mental model, potentially the "fault" in ourselves, and correcting it takes a great deal of energy and courage.

Because of this, most people quickly give up. They don't have enough energy—or courage. It takes a great deal of energy and courage just to enter the workplace and face the day for some people.

However, we must begin somewhere if we want happier work lives and better functioning, organizations that are more productive. Fortunately, Martin Seligman had a similar idea.

## 3 Positive Psychology

In a YouTube video recorded in July of 2013, Seligman talks about being "media trained" while he was president of the American Psychological Association. Part of their strategy was to help him speak in sound bites. They asked him to respond in one word to the question "What is the state of psychology today?" His response was "Good." They felt this was an insufficient response, so they asked him to respond in two words. His response was "Not good." Wanting a little more information they allowed him three words, and then his response was "Not good enough."

Until the emergence of positive psychology in 1998 when Martin Seligman chose it as the theme for his term as president of the American Psychological Association, psychology focused on fixing broken people, and, as Seligman puts it in his Ted talk, "the disease model." Positive psychology focuses on helping people who are already reasonably happy and functioning well to be happier and thrive. First the field developed authentic happiness theory and then, finding shortcomings in that theory, developed well-being theory.

### 3.1 Authentic Happiness Theory

Authentic Happiness theory looks at what makes us happy in a long term and stable fashion—not the temporary good feeling we get from a new car, a fat raise, or a new puppy. In fact, that kind of happiness tends to be quite transitory, and then we re-stabilize around a new level of dissatisfaction if this is the only kind of happiness we pursue. This kind of happiness is not durable. Authentic Happiness is composed of:

- Positive emotion
- Engagement
- Meaning

Of these three, positive emotion supplies the least durable happiness. So, for instance, if writing really good code is deeply engaging for you (helps you move to a flow state) and the products you create or even the code itself imbue your life with meaning—and you get to write code for a living, you are more likely to be authentically happy than someone who gets a series of frequent small raises or a new car on an annual basis but who doesn't experience engagement and meaning in his or her work.

While authentic happiness theory did move psychologists out of the disease model since it encouraged them to help more or less happy people be more durably happy, it did not help them move their clients to a state of flourishing. Further, researchers began to notice that some people were durably happy without conforming to the three tenets of authentic happiness theory; in fact, these people were sometimes choosing to do things that would not necessarily generate positive emotion in themselves—in the pursuit of happiness. As Seligman says in his book *Flourish: AVisionary New Understand of Happiness and Well Being*, "I now think that the topic of positive psychology is well-being, that the gold standard for measuring well-being is flourishing, and that the goal of positive psychology is to increase flourishing" (Seligman, 2013, 13).

## 3.2 Well Being Theory

Well-being theory developed because of three inadequacies in authentic happiness theory. Those inadequacies were a cultural alignment of happiness with cheerful mood; a cultural notion of happiness as a feeling state or mood; and social research that shows that people seek out experiences and a range of life and lifestyle choices for their own sake. In fact, as Seligman helped nurture a community of positive psychology researchers, they learned that people sometimes *sought out experiences that would not generate positive emotion but which they believed contributed to their overall well-being*.

Well-being theorists developed a notion of PERMA, the elements of well-being:

- **P**ositive emotion
- **E**ngagement
- positive **R**elationships
- **M**eaning
- **A**ccomplishment

While all these elements can be satisfied in the workplace, many people work in environments where it is difficult to find these elements. This presents a challenge to double loop learning.

In order to move to the vulnerable state that allows us to confront mental models requiring change in ourselves and our sense of how our world—or workplace—works, we must confront the embarrassment, however strong or mild, that recognizing this failure to measure up often entails. To do this, we need a general sense of continued well-being in the work context. We need to be fairly sure, for instance, that noticing and acknowledging this gap—even asking for help in fixing it—will not result in ostracism from the group or loss of our status, income, or employment. We need to know that we can survive the experience of finding a lack—however minor or serious in ourselves—and that we will live through it, even thrive, as result of fixing the problem, changing the mental model.

## 3.3 Positive Psychology as a Support for Double Loop Learning

Since its inception in the late 90's, positive psychology has been rapidly contributing research and methods for optimizing PERMA and helping people flourish. Videos and books are increasingly available in the main stream, and many of these techniques are flowing into the software development field through Agile and its coaching mechanism, among other channels. The promise apparent in using positive psychology understandings and methods to set an organization context that will foster double loop learning is exciting and is being tested, however informally, in a range of organizations. This paper now provides a few insights into how to test it in yours.

## 3.4 Losada Ratio

The Losada ratio is an example of a method the positive psychology movement has developed to help reasonably happy people move toward flourishing. Barbara Fredrickson is known for her laboratory research into positive psychology. She developed a "broaden and build" theory of positive emotion. She identifies negative emotions as “firefighting emotions, which identify, isolate, and combat external irritants” while “the positive emotions broaden and build abiding psychological resources that we can call on later in life.” Further, “(p)ositive emotion does much more than just feel pleasant; it is a neon sign that growth is under way, that psychological capital is accumulating” (Seligman, 2013, 65-66). Research Fredrickson did in organizations showed that the ratio of positive to negative language in meetings correlates with the extent to which the company is flourishing economically. The ratio at and above which organizations flourish is 2.9:1 positive expressions to negative expressions. This ratio came to be known as the Losada ratio. It has been more famously applied in marital research done by John Gottman. Gottman’s research shows the ratio of positive to negative experiences that is the hallmark of a happy marriage is 5:1. It’s interesting to note that you can overdo positivity in organizations: A ratio of 13:1 results in a lack of credibility (Seligman, 2013, 65-66).

So, Argyris alerts us the critical need for feedback about failure and the many social and intellectual mechanisms we have for avoiding giving, receiving, and internalizing feedback about failure. And his work also helps us understand the critical need to identify and grapple effectively with failure while highlighting a learning model that challenges us quite deeply to change the underlying thinking that generates failure, double loop learning. And, we know that knowledge workers and their productivity are even more broadly and deeply affected by the psychological demands of double loop learning.

A comment on “Teaching Smart People How to Learn” published as an addendum to that article provides additional insight into the special situation of knowledge workers. Haridimos Tsoukas notes that “As organizational ethnographers, such as Julian Orr (1996) and Etienne Wenger (1998), have shown, daily work in information-rich companies is more decision intensive—more loci for decision making by employees are created.” He then draws the conclusion that the more “informated” a workplace is the more reflexive or self-reflection-oriented the workplace is capable of being.

Tsoukas points out that Argyris, in his body of work, repeatedly points to the difficulty practitioners have in doing reflexive thinking—“double loop learning.” Tsoukas states that this is particularly true for knowledge workers who, by definition, work in highly informated environments “because, to the extent that they are more psychologically present at work, they expose more of themselves to others; hence, they are more vulnerable.” Therefore, the reason it is important to short-circuit defensive reasoning so that a knowledge worker can engage in reflexive reasoning and in double loop learning is that knowledge workers bear a greater burden of “constantly challenging yourself, of expanding your horizons, of ‘knowing thyself.’” And, therefore, Tsoukas reasons that “Argyris invites knowledge workers to undertake a primarily moral, not just technical task: to be open to criticism, to be willing to test their claims publicly against evidence, to accept that they too are partly responsible for the problems they are confronted with” (as cited in Argyris, 1991, p. 15).

Clearly, organizational members occupying leadership positions have a role to play in setting a context where double loop learning can reasonably happen.

## 3.5 Leader’s Role in Setting Context for Double Loop Learning

As C. Otto Scharmer shows repeatedly in *Theory U: Learning from the Future as It Emerges, a Social Technology of Presencing*, “(t)he primary job of leadership, I have come to believe through my work with Schein, is to enhance the individual and systemic capacity to see, to deeply attend to the reality that people face and enact. Thus the leader’s real work is to help people discover the power of seeing and seeing together” (2009, Scharmer, 136). You can’t change what you can’t see, and as Argyris has pointed out, our organizational defensive routines are very good at preventing us from seeing such that we cannot work on real problems, identify the mental models that drive them, and change those models or adopt new ones.

In addition to helping the organization to see itself, the leader can provide the service of showing others how to engage with failure productively, and to use it as a legitimate and authentic springboard to greater success, rather than letting it tip you over into a grief spiral or defensive routines so that you cannot see the failure. To do this, leaders have to admit error in themselves and publicly go through the process of rectifying that error in their words and actions to the benefit of the organization.

## 3.6 Argyris' Advice to Leaders

Additionally, Argyris provides key points of advice to leaders to help others engage in real learning. Moreover, these three pieces of advice actively support the decrease of decision latency and increase of decision quality through a highly engaged coaching mechanism. Argyris urges:

- Don't just "fix" concerns.

  Help employees focus on self-awareness and initiative rather than acting on blaming. Listen for your part in problems and demonstrate the kind of response you'd like to see from them.

- Don't make promises you shouldn't keep.

  Mentor and coach employees on reasonable and unreasonable requests. Educate them on the market and support creative responses to market demands.

- Respect employees.

  They need to take their punches and focus on solutions—just like the leadership is expected to.

  (1994, Argyris, 85)

Being the "fixer" can leave the learner stuck in the single loop learning cycle. Making promises you shouldn't keep rather than educating workers about why some promises they seek are not reasonable (such as an assurance of lifelong employment in their current organization without progressive improvement) prevents them from adopting mental models that support their successful functioning. And, protecting employees from the consequences of their actions prevents them from testing their mental models in the real world.

## 3.7 Teach By Example

In order to encourage double loop learning in the workplace where you would propose to lead, you need to enact it yourself—and be seen to be doing so:

- Read widely, as well as in your field, to increase your knowledge and judgment.
- Take classes in and outside of your field to both increase knowledge and judgment but also to pursue your own well-being.
- Express interest in the learning of others as well as sharing your own learning naturally and in the course of doing business. Show that learning, changing, and growing are all part of the expected human experience, which may sometimes be bumpy and have moments of embarrassment and self-correction, but which is also desirable in your organization and leads to personal and organizational success. Act out the assumption that learning is part of the work.
- Recognize failure and, especially, effective responses to it. While failure can identify an opportunity for learning, it is not only learning but also the change in the self and the organization to prevent the same failure again in similar circumstances that is desirable. And, the skill of learning and changing based on evidence is as valuable to the organization as any technical skill.

- Share where you have fallen short, how you have recovered, and what you learned. Show that it possible to grapple with failure effectively, and that it does not have to result in exclusion from the group or unrecoverable loss of status or income.
- Pursue, and share your pursuit of, PERMA both inside and outside of work.

# 4 Conclusion

Double loop learning is different from single loop learning in that it can prevent the same kind of error from occurring again, rather than just help us spot and correct it more quickly in the future. It is an extremely valuable skill for knowledge workers. However, few individuals or organizations are skilled at it. Knowledge workers are especially in need of double loop learning skills though they are also especially vulnerable to its hazards because they bring more of themselves to work than the assembly line worker and are more psychologically vulnerable as a result.

Double loop learning, while extremely valuable in addressing the organizational imperatives of decreasing decision latency and increasing decision quality, is also exhausting to the human psyche. All participants, and especially individuals assigned to organizationally identified leadership roles, will benefit from learning about positive psychology and using it to set a context in which double loop learning can happen more easily. The Losada ratio is one example of a technique contributed by positive psychology which can be used on a daily basis to ensure that an environment most likely to foster general well-being among organizational participants is brought into being.

Those who would lead healthy, vital organizations populated by knowledge workers have a role in ensuring that a context is set that makes double loop learning easier to engage in. And, they also have a role in modeling this skill. Positive psychology provides us with tools and techniques to set this context, and the work of Christopher Argyris clearly marks the path to the skills required in and challenges of double loop learning.

## References

*Argyris, Chris. 1991. "Teaching Smart People How to Learn."* Harvard Business Review *69, no. 3: 99-109.* Business Source Complete*, EBSCO*host *(accessed July 27, 2014).*

*Argyris, Chris. 1994. "Good Communication That Blocks Learning. (cover story)."* Harvard Business Review *72, no. 4: 77.* Business Source Complete*, EBSCO*host *(accessed July 27, 2014).*Argyris, C. (2002). Double loop Learning, Teaching, and Research. *Academy of Management Learning & Education, 1*(2), 206-218. doi:10.5465/AMLE.2002.8509400

*Argyris, Chris. 1986. "Skilled incompetence."* Harvard Business Review *64, no. 5: 74-79.* Business Source Complete*, EBSCO*host *(accessed July 27, 2014).*

*Argyris, C. and Schön, D. 1978. Organizational Learning: A Theory of Action Perspective.* Reading: Addison Wesley.

*Scharmer, C. O. 2009. Theory U: Leading from the future as it emerges*. San Francisco: Berrett-Koehler.

*Seligman, M. E. P.* 2013. *Flourish: A Visionary New Understanding of Happiness and Well-being*. New York: Atria.

*Seligman, M. E. P. On Positive Psychology*. TED-Ed. https://www.youtube.com/watch?v=5CpLEOO5oyo Accessed July 27, 2014.

# Scalability: Pushing the limits

**Neha Rai (neha_rai@mcafee.com)**

**Tim Schooley (tim_schooley@mcafee.com)**

**Tejas Patil (tejas_patil@mcafee.com)**

**Intel Security Group (McAfee)**

## Abstract

How do we determine if software is a success? For server-based applications, success could mean more load, either as a web app or a client-server application. As the numbers of end users grow, the load on a server often increases substantially. Hence it becomes crucial to address the scalability challenges of the software to meet the rapidly growing need.

Most scalability flaws are deep in the architecture, which are very hard and expensive to change later. The quick and easy fix is to add more servers. However, by including more servers, one can hit diminishing returns very quickly, unless the software is designed to scale that way. So, is there a magical way to determining how much software can scale up?

This paper talks about the scalability of client-server software, which is common in an enterprise environment. Here we want to stress the importance of performing scalability tests on any client server application by creating a simulated environment. We are going to address some key questions that arise during the process, like:

- What should be covered in scalability testing? What is the target?
- How to determine is the scalability limit?
- How to identify scalability bottleneck?
- How to determine if the test results are complete and testing should stop?

## Biography

*Neha Rai is a QA Engineer at Intel Security Group (McAfee) with more than 6 years of quality assurance experience. She holds a Master degree in Computer Application from National Institute of Technology, Karnataka and currently working in Brighton, UK*. Being *highly passionate about quality, she strives to look for ways and methods to improve software reliability and usability. In Last couple of years she has become extensively involved in scale testing of McAfee Drive Encryption product.*

*Tim Schooley is an Agile Product Owner and Senior Software Engineer at Intel Security Group, and holds a Master of Science degree in Computer Science from UCL, London. Currently working in Brighton, UK, he develops a number of data protection products and manages their respective backlogs. He is a strong proponent of delivering maximum value and quality.*

*Tejas Patil is a QA Lead and Scrum Master at Intel Security Group (McAfee) in Brighton, UK. He has worked for past 9 years in McAfee developing, testing and managing software. His main responsibilities include driving the concept of software quality throughout the product development lifecycle. He is passionate about building high quality products that excite customers.*

# 1. Introduction

Scalability is necessary in technology and business settings. The base concept is consistency – the ability for a business or technology to accept increased volume without impacting the contribution margin, which is deduced by subtracting variable costs from the total revenue. For example, a given piece of equipment may have a capacity for 1–1000 users, while beyond 1000 users additional equipment is needed or performance will decline (variable costs will increase and reduce the contribution margin).

Recently, HBO aired the much-anticipated season finale of 'True Detective' on their network, which included a live streaming of their program on HBO Go app. Apparently, they underestimated the popularity of the finale. The live stream and the app crashed due to the massive number of people trying to watch the show, resulting in a blackout for majority of users. And… the debacle continued, the same thing happened again during Game of Thrones premiere. Furious fans went to social media to express their unhappiness at being unable to watch the episode.

Another example where software is facing scalability issues is social networking websites. Many of us are familiar that Facebook, Digg and Twitter have been making improvements to make their apps more scalable. Unsurprisingly, the challenge is not limited to web based applications, but any client-server model can trip over.

Scalability Testing refers to performance testing that is focused on understanding how an application scales. As the product is deployed on larger number of systems or as more loads is applied to it, we need to understand what happens to it. Our goal is to understand at what point the application stops scaling and identify the reasons for this. Scalability testing helps to determine whether your management application scales with your workload growth as the managed network grow in numbers and complexity. Hence, scalability testing forms an essential part of the entire development and testing process of any server based application.

# 2. Start with understanding the product vision

Before you start scalability testing, be sure that you understand the current project scaling vision. The project vision is the foundation for determining what kind of testing is necessary and what is valuable. For example when we started scale testing for our product we started with the vision to support 'hundred thousand client nodes per server'. Or for any other application the vision could be to support 100,000 concurrent users.

By having scalability vision, we set the scalability expectation for our product to meet. So, by having a clear vision, when testing is done, we will know if the product meets our quality goals.

# 3. Identify the test environment

Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. To achieve the goals of scalability testing, a unique test bed would need to be built. It is important to create a separate testing environment that is comparable to production. If the machine configuration, speed, and setup aren't the same, extrapolating performance in production is nearly impossible. To make sure your environment is right, be sure to consider:

- Hardware where server software runs. Identify where database, primary server and additional servers will run.

- Hardware that the client software (or simulator) runs on.
- Software you plan to use to capture the performance data like SQL profiler, perfmon etc.

In addition to the resources necessary to run the scalability lab, success of your load testing effort depends on other roles within your organization. Identify a scalability testing team, there will be scenarios when we encounter issues that are hard to debug. This would require re-testing with a debugger attached and would need involvement of the development team. Considering these situations well ahead will save you from last minute surprises. Other than QA and a Development manager for the project, you need support of other team members to execute the scalability testing successfully. These elements are key to that success:

- Scale lab team - Takes ownership of the effort and runs the system test. At least one person is required.
- Development team - Identifies and tracks problems involving stability and performance. At least one person is required.
- Database Engineer - Identifies and solves database problems. At least one person is required.

Scalability problems are most likely rooted in the database, the data access strategy (such as stored procedures, prepared statements, or inline SQL) or data access technologies (such as ADO, ODBC, and so on). A well-qualified DBA available as a full-time resource plays a key role in scale testing effort.

## 4. Identify Acceptance Criteria

When identifying acceptance criteria, we need to identify the response time, throughput, resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern and resource utilization is a system concern.

Counter to this, sometime you would need to start scale testing without knowing what the desired response time; throughput or resource utilization is for an application. Start testing with a smaller number of clients in the setup, in other words simulate only the number of clients that you think your application can handle without any problems. Once you confirm that the test case is successful, start increasing the number of clients. Continue with this approach and work slowly to identify the sweet point (or target throughput), the point where we can say the scale performance is its most efficient. It may be that you reach to a point where you observe a decline in the performance, like the server response time will go down, or when you add more servers and you don't see any performance improvements. In this situation, revisit the last test you did and record the details, like the number of clients, number of servers or any other parameters. This gives you a baseline to calculate the current product throughput and response time that will help to identify if the current state of the product meets the product vision.

Another approach could be to design your test cases around these key questions:

- Where is the system bottleneck, and how many synchronized concurrent requests can it handle?
- How many additional clients can your system support without diminishing results?
- Do the results scale linearly as you add additional hardware?
- Are there any stability issues that prevent the server from operating in a production environment?

This approach uses additional information from the development team, which anticipates where problems might arise.

During scale testing, if we can determine the targeted throughput required to support our application in a production environment, then we can easily determine if the current scalability meets the expectation, the ultimate goal of scale testing!!

# 5. Plan and Define test scenarios

This is most important part of the scale testing. In most cases the major part of your success comes from correct planning.

To plan successfully, be sure to address these key points:

- Identify key scenarios,
- Determine variability among different parameters and how to simulate that variability,
- Define test data
- Establish metrics to collect.
- Consolidate this information into one or more models of system usage to implement, execute and analyze.

At this stage, one should be able to determine if their setup or test environment needs a separate tool for simulating test data. For instance, if your test scenario involves putting load on the server using multiple users or multiple client machines, then you would require an additional tool, which can simulate the load. For a scale-testing project to be successful, both the approach to testing scalability and the testing itself must be relevant to the context of the project. If you don't have a clear understanding of your project context, scale testing is bound to focus on only those items that the performance tester or test team believes to be important, as opposed to those that truly are important. So by using something, which nearly simulates the application behavior, you can cover most of the functionality and discover the scaling bottlenecks.

# 6. Creating a simulation tool

You should identify all the test scenarios that need to be executed as part of scalability testing before implementing a simulation tool so that we can eliminate any detailed work.  In other words, you need to be clear about what is expected of a simulator so that you do not spend time developing features that are unnecessary.  Either we use some existing software or develop our own simulator, but identifying the right simulation tool is vital.

When choosing a simulator, remember that in order to determine the capacity of a target environment, we need to adjust the variables to find the maximum scalable point of the system.  In addition, the test should simulate other expected loads that will run on the same production environment as the application under test.  Since simulation costs typically increase quickly with the level of details, we need to fine-tune the balance.

## 6.1 Testing the simulator before relying on it:

Perform an acceptance test on the simulator you choose.  The acceptance tests should be ready before selecting the simulator; this will help confirm that the simulator meets all the feature expectations.  All the result and testing are going to rely on how closely the simulator mimics your actual product.

The acceptance test should cover all the scenarios that would be executed as part full-fledged scale test. This will help you judge the simulator's capabilities and let you know about any limitations well in advance. Furthermore, you can use this as an opportunity to educate yourself about the tool you are going to rely on.

# 7. Executing Tests

## 7.1 Collecting performance data:

Performance counters should be set to capture the system state as well as the application state. Use counters to provide information as to how well the operating system, the application, service, and driver are performing.

When you need to measure how much of system resources your application consumes, you need to monitor these items:

- Disk I/O Read and write disk activity. I/O bottlenecks occur if read and write operations begin to queue.
- Memory Available memory, virtual memory, and cache utilization.
- Network Available bandwidth being utilized, network bottlenecks.
- Processor utilization, context switches, interrupts and so on

It is important to choose performance counters according to the application under test so that you can measure the performance of these components efficiently.

The counter data can help identify system bottlenecks, it is also used fine-tune the system and application performance. However, performance counters are only useful to identify the symptoms of a problem, not the cause. Therefore, it’s important to use application level logging as well. Every software system has logging requirements so application processing can be monitored and tracked. Enable logging in debug mode, which provides detailed information about an event. For application-level data generated by a logging system, it is a good idea to use a viewer that lets you to get immediate access to error and performance information.

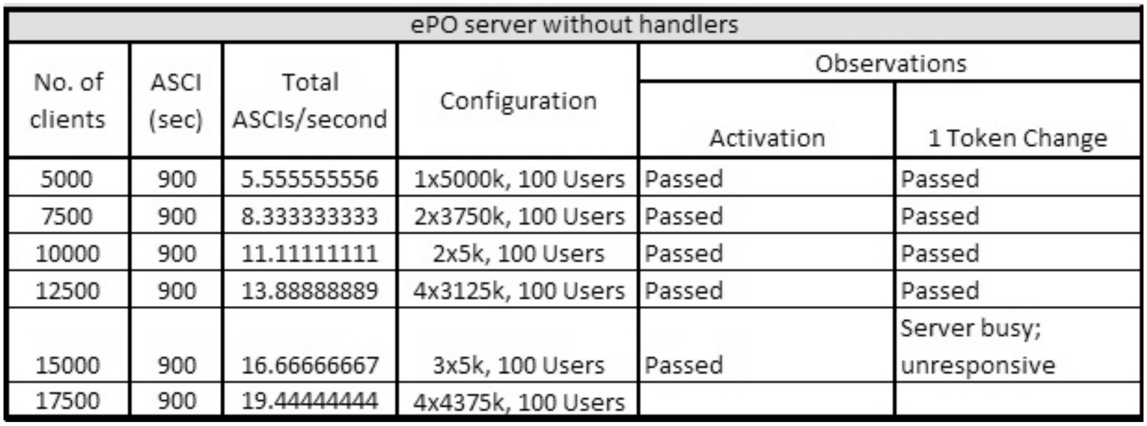
## 7.2 Running small-scale tests (eliminating bugs):

Setting up a big scale test with lots of machines and simulators could turn out to be expensive if on the first run you find blocking bugs.  First, you need to rectify all the bugs that reduce the scalability, like thread contentions, memory leak, deadlocks etc.  To achieve this, start your test on small-scale setup but try to keep impact on the server high.  In other words, stress the server with lower number of clients by changing other parameters.  For example, you target to support 100K client nodes on one server within a randomized interval of suppose 4 hours.  In order to keep the impact on the server the same, reduce the number of clients and the number of hours so the overall client request being processed by the server remains the same. Repeating small-scale tests are more easily compared to full-fledged scale tests which we can save for later.

In our applications scalability is affected by the number of users, the number of nodes and agent-server-communication interval. We reduced the number of nodes and reduced the agent-server-communication interval (ASCI) as well to keep the throughput nearly the same. This allowed us to stress the server and to expose issues like database deadlock and memory leaks.  During our small-scale test cycle, we introduced an additional server while keeping other parameters constant to observe thread contention. The intent here is to keep the server under load, not by increasing the client node count, but by increasing other parameters that affect the product scale performance.  What we found that, 90% of the scalability issues came out during small-scale tests and the remaining 10% were observed when small-scale test data was extrapolated on a large-scale setup.  Another advantage of starting with a small-scale setup is that they are easy to reset in case you need to verify any bugs found and fixed during the scale testing.

Try identifying a scalability sweet spot in the setup, the point where the scale performance is at its peak with the given parameters. For example, start testing with smaller number of clients with your server. Once the test passes, add more clients while keeping all other parameters constant. If no issues arise, continue to add few more clients, if there is no drop in the server performance you can continue adding the clients until you are concerned with server performance. Once you find the maximum scaling point, write down the configuration details so that you can use it to extrapolate the results to a larger scale. If your application supports additional servers, you should be running similar tests by increasing the number of servers so that you can identify how many additional servers can be included in the system before you observe the scalability loss.

For example in the table below, tests have gone to the point where the number of clients is 12500 and agent server communication (ASCI) per second is 13.88. Beyond this point, the tests started to fail at a higher ASCI/sec rate. Hence 13.88 ASCI/sec is the sweet spot in this setup.



| ePO server without handlers | | | | | |
|---|---|---|---|---|---|
| No. of clients | ASCI (sec) | Total ASCIs/second | Configuration | Observations | |
| | | | | Activation | 1 Token Change |
| 5000 | 900 | 5.555555556 | 1x5000k, 100 Users | Passed | Passed |
| 7500 | 900 | 8.333333333 | 2x3750k, 100 Users | Passed | Passed |
| 10000 | 900 | 11.11111111 | 2x5k, 100 Users | Passed | Passed |
| 12500 | 900 | 13.88888889 | 4x3125k, 100 Users | Passed | Passed |
| 15000 | 900 | 16.66666667 | 3x5k, 100 Users | Passed | Server busy; unresponsive |
| 17500 | 900 | 19.44444444 | 4x4375k, 100 Users | | |

## 7.3 Extrapolating (verifying results on big scale):

Once you have results from small-scale tests you can use the data to extrapolate the results to a larger scale setup. As in the previous example, we have seen that system works well when ASCI/second is 13.88. Hence by general calculation, if we intend to support 100K nodes per server, we need to distribute the clients over 7204 seconds (nearly 4 hours). Once you have extrapolated the data, the numbers should be verified by testing in a larger scale setup.

Here you can choose to test directly with the target number if you find your build quite stable. Otherwise, depending upon the test results, rate and type of bugs found, you can choose to start with half of the target number or less. For example, test with 50,000 clients and if no issues were found, then you are good to verify the results with your targeted number of clients, 100,000. While extrapolating our small-scale results, we observed issues related with thread contention and database deadlocks.

Extrapolating and verifying the data is a mandatory step in the whole process. During our testing, we couldn’t validate the results obtained by small-scale tests because of the existing issues that were bringing the performance down. These issues only got exposed while testing on a really large-scale setup. Therefore, do not just extrapolate the results and give a recommendation, make sure any extrapolations done are well tested.

## 7.4 Identifying possible bottlenecks:

When scaling results in degraded product performance, it is typically the result of a bottleneck in one or more resources. Bottlenecks are elements of your system that impede the normal flow of traffic.

When your application does not meet performance requirements, you should analyze data from the test results to identify bottlenecks in the system and to formulate a hypothesis of the cause. Sometimes the test data are not sufficient to form a hypothesis, and you must run additional tests using other performance-monitoring tools or with a debugger attached to isolate the cause of the bottleneck. Alternately, you can eliminate the parameters involved in the test to pinpoint the problem.

Once you've taken steps to identify and potentially resolve the bottleneck, repeat the tests to see if the fix has introduced or uncovered any new bottlenecks. Test to see if the scalability limit has increased.

# 8. Analyze Results and Report

Technical team members need more than just results — they need analysis, comparisons, and details of how the results were obtained. Your technical report should contain information that can be used to re-run the scalability test on later versions. The report should contain all the bugs found during the scale testing cycle and details of fixes made so that one can design the next test cycle accordingly. The document should state any concerns and recommendations. And it should clearly state the maximum scalability limit the product has so they can be verified in subsequent test cycles.

On the other hand, non-technical team member like Managers and other stakeholders need more than just the results from various tests — they need conclusions based on those results, and consolidated data that support those conclusions.

These two groups tend to look for very different things in a performance report and are inclined to prefer different presentation methods. When reporting, make sure that you identify which group you are reporting to and know what their expectations are before deciding on the best way to present the results you have collected.

# 9. Conclusion

Scalability testing should be implemented as part of the development effort. By implementing a scalability testing plan early, you can minimize any surprises at deployment time. Before going to production, scalability testing is the only way to uncover major problems that are inherent in the architecture. To make scalability testing successful, we need a separate environment with comparable production hardware, a robust simulator, and the teamwork of people in your organization.

Hopefully you will find this paper helpful when planning scalability testing for your projects. I'd like to leave you with the following final suggestions for your next project:

- Start Early
- Include scale testing as part of the development effort
- Choose the right simulator
- Start with small-scaled setups and work up to your bigger scale setup.
- Establish distributed logging
- Monitor and interpret your results carefully.
- Analyze the results and report
- Celebrate your success!

# 10. References

[1]. Scaling Your Internet Business
http://www.scribd.com/doc/15493750/GoGrid-Scaling-Your-Internet-Business

[2]. Testing for Scalability
http://msdn.microsoft.com/en-us/library/aa292189(v=vs.71).aspx

[3]. Estimating the Throughput
http://msdn.microsoft.com/en-us/library/ff648064.aspx

[4]. Avoid bottleneck when your web app goes live
http://msdn.microsoft.com/en-us/magazine/cc188783.aspx

# Early Performance Testing

**Eric Proegler**

eproegler@gmail.com, @ericproegler

## Abstract

Development and deployment contexts have changed considerably over the last decade. The discipline of performance testing has had difficulty keeping up with modern testing principles and software development and deployment processes.

Most people still see performance testing as a single experiment, run against a completely assembled, code-frozen, production-resourced system, with the "accuracy" of simulation and environment considered critical to the value of the data the test provides. But what can we do to provide actionable and timely information about performance and reliability when the software is not complete, when the system is not yet assembled, or when the software will be deployed in more than one environment?

## Biography

I have worked in testing for 15 years, and specialized in performance and reliability testing for 12. I work for Mentora Group, a national testing consultancy, from my home in Mountain View, California.

I design and conducts experiments that use synthetic load to help customers assess and reduce risk, validate architecture and engineering, and evaluate compliance with non-functional systems requirements. I test in a wide variety of contexts, using tools appropriate to the job at hand. Some of the applications I've tested recently include Oracle E-Business ERP systems, a hospital's provider portal, a large scale "second-screen" mobile app, a large B2B EDI Translation clearing house, a custom SaaS application, and e-Commerce web sites.

I'm an organizer for WOPR, the Workshop on Performance and Reliability, and a Community Advisory Board member for STPCon. I've presented and facilitated at CAST, WOPR, STPCon, and STiFS. I have recently started engaging with tester certification and testing standards to help our community best respond to these economic tactics.

In my free time, I spend time with my family, read, see a lot of stand-up comedy, seek out street food from all over, play video games, and follow professional basketball.

# 1 Introduction

Iterative development processes has changed software testing from a gating activity occurring at the end of a development process into a collaborative one that occurs throughout the development process. Testers participate earlier in projects, and the role of automation in testing has increased. The orchestration of building software, installing it to test environments, and running extensive automated checks without human intervention is now widely practiced. Software under test is changing frequently or even constantly; some would say continuously. What hasn't happened broadly is the adaptation of performance testing practices to keep up with these new development methodologies.

A traditional performance test involves creation of a high-fidelity simulation of production workload, run against software that is fully developed and has completed functional testing, executing on hardware that is equivalent in resource capacity and configuration to the production environment. Realism can be simulated to various levels of "accuracy", depending on time and budget, with various degrees of success, depending on available information about the system under test.

## 1.1 Challenges in Simulation Tests

We will call this approach a "Simulation Test" for the rest of this paper, to distinguish from a "Load Test". A Load Test is a multi-threaded automation test that could be a Simulation Test, or it could be some other load that is not intended to Simulate a specific workload. Load testing tools are used to create Load Tests; the application of a "Realistic" user model, test environment, and completed software is necessary for a Load Test to become a Simulation Test, as we mean it here.

A frequently encountered challenge with Simulation Tests is that testing doesn't take place until software is completed. Waiting until software is complete means that testing occurs at the very end of the project. This exposes testing to time pressure, particularly when earlier phases of the project run past milestones.

Finding problems at a point in the schedule where the testing is the last gate to going live is of limited value. Either they are minor enough to be deferred, or serious enough to delay the entire project.

This problem is shared with functional testing, and was one of the major factors that led to the adoption of iterative software development processes. Most people still see performance testing as a single experiment, run against a completely assembled, code-frozen, production-resourced system, with the "accuracy" of simulation and environment considered critical to the value of the data the test provides. But what can we do to provide actionable and timely information about performance and reliability when the software is not complete, when the system is not yet assembled, or when the software will be deployed in more than one environment?

## 1.2 About This Paper

This paper describes approaches and techniques for providing performance feedback in iterative development processes, particularly when software isn't complete, components are not yet implemented, the production workload is not known, and/or production-class hardware is not available.

- *Section 2: Risks Addressed by Performance Testing*
- *Section 3: Getting Real About Realism*
- *Section 4: Performance Testing Iteratively*
- *Section 5: Alternative Performance Testing Techniques*
- *Section 6: Testing Incomplete Systems*
- *Section 7: Communicating Performance During Iterative Development*

# 2 Risks Addressed by Performance Testing

To engage with these new contexts, we should consider what risks lead us to performance testing in order to mitigate them. Here is one formulation of the classes of performance and scalability risks:

## 2.1 Scalability – Expensive Operations

If an activity take a long time to complete, that is, has a response time of a second or longer, it is likely to be of interest for optimization. This is because these longer running activities will almost certainly create a heavy computational load on the server, depending on their frequency.

These types of issues are often a result of implementing tall stacks of scripting languages, technical frameworks, and APIs. Situations like this are opportunities to observe Wirth's Law: "Software is getting slower more rapidly than hardware gets faster."

When inefficiency is introduced, an increase in response time is the most likely way to find it. When inefficiency increases resource pressure, it can be difficult to trace resources back to a specific activity.

## 2.2 Capacity – System's Ability to Handle Load

In Simulation Tests (a single complex experiment that attempts to simulate a full workload), capacity is usually what is being tested. The capacity of a system is determined by a combination of hard and soft resources. When a resource is exhausted, the first limitation is encountered. This is where the system's capacity limit is until that resource is increased. These kinds of problems are often first presented as response times that are very sensitive to load.

Sometimes this first limitation is encountered against a hard resource, such as available CPU, memory, storage, or network capacity. This is the reason why "realism" in performance tests is prized; if a resource is available in less quantity than it will be in production, the limitation occurs sooner than it should.

Often, the bottleneck is a soft resource instead, such as a limitation on the number of database connections or sessions, licensing restrictions, or the configuration of runtime environments such as Java. Tests can expose these limitations, allowing for configuration changes or code refactoring to correct them or increase capacities.

## 2.3 Concurrency – Race Conditions and Contentions

A certain class of risk in software only appears under load in a multi-user system. These problems can be very difficult to recreate, but when they occur, they can cause significant problems for all users of the system, including causing the entire system to crash.

Generally speaking, they occur when a specific combination of states occur as a multi-user system serves multiple users. These conditions may be confused with very rarely occurring errors. Any rare set of conditions, whether based on concurrency or not, is much more likely to be found in load tests than typical (single-threaded) testing.

## 2.4 Reliability and Recoverability

Multi-user systems are required to service many thousands (sometimes millions) of sessions while maintaining acceptable performance characteristics, sometimes over extended periods of time. Problems such as memory leaks, disk space for database files, or programming logic errors can be exposed by long-running load tests, frequently called "Soak Tests". Most Simulation Tests end after an hour or two, meaning that they might not expose these problems if they are slower to appear.

# 3 Getting Real About Realism

Performance Testing, as traditionally practiced, is usually based on the idea that a single large-scale simulation (or series of repeated simulations) provides an opportunity to evaluate a multi-user system under operational conditions. Automated tools are used to create load against a system, making it behave similarly to how it will when users are operating the system after go-live. This thinking places a great deal of value on "Realism", or the degree to which the simulation is thought to resemble expected operational conditions. "Realism" is a very loaded term that implies several things.

The idea that performance testing must achieve "Realism" in order to have value is inaccurate, outdated, and often impossible. It must be set aside when we are testing rapidly changing or incomplete software, when we don't know what production will look like, or when testing against a different environment and configuration than we expect in production.

The Simulation Test constructs a scenario with many variables, attempting to create equivalent load on an equivalent system in order to create comparable conditions for evaluating the performance of the system under test. Under closer examination, creating an equivalency seems to become even more difficult. The model looks like this:

***Will the complete, deployed System support:***

***(a, b) classes of users***
***performing (c, d) activities***
***at (e, f) rates***
***on (m) environment/configuration***
***under (n) external conditions***
***and meet x response time goals?***

We usually believe those conditions to be the most representative conditions we can design with the knowledge we have in order to conduct a single performance experiment. We should be humble and cautious about how close to "Reality" and "User Experience" we actually get.

## 3.1 Realism in Workload

Variables (**a-f**) are components of the workload model. When a production system already exists, this information might be derived from log files, though some prediction of future state is still necessary. Without actual workload information, a model must be created, usually with input from a couple of sources, but always subject to possibly being very different from what reality turns out to be.

Whatever the source of the model and its "accuracy", load tooling imposes some conditions that are fundamentally different than how people use a system. Examples of this include rigid session arrival schedules, identical activity sequences per script, users that never abandon sessions, and traffic that is well distributed over a test instead of having organic resonances associated with start of day, end of day, lunch, and meeting times. Testers frequently reduce fidelity by simplifying the data used.

The author's experience is that load models are frequently intended, at least initially, to be an example of a peak hour on a peak day. Estimates, projections, and outright guesses are common at this stage. Stakeholders who may not have much knowledge of load modeling and/or the applications reality might insist on certain round number thresholds that must be accommodated. Finally, there is some rounding up, some "horse-trading" of less frequently used or difficult to script activities, and eventually, a model emerges that is believed to be representative – but turns out to not be very much like how it will be used as a live system.

Data density and caching is rarely considered. In production, some data will be accessed often, and some will be accessed rarely. Testing often attempts to achieve something like this, but rarely succeeds.

The nature of people using software is essentially ignored. People use different navigation paths to the same places, click multiple times, when software isn't fast enough, get distracted and delay or leave active sessions, and show a variety of behaviors that simulation ignores. Scripts execute software in identical fashion, avoiding wide swathes of what people would do in the software.

None of this accounts for how load evolves and varies over days, months, and years.

### 3.2 Realism in Environment

Other challenges are encountered in addressing variables (**m, n**) – the environment. Most people start off thinking about this in terms of machine resources being equivalent; too many insist on "Identical". When machine resources were dedicated to a specific group of physical servers, it was possible to assess and project available resources with some degree of accuracy. In a world where resources are abstracted in large, dynamically adjusted pools by virtualization, SANs (Storage Area Networks), and cloud-based infrastructure, this is an incredibly difficult task. Comparable resources are critical to assessing capacity, but it can be extremely difficult to understand how much capacity is available at any one moment – and it changes constantly.

This dynamism in deployed resources is due to the impact of other business activities that contend for the shared resources. Soft resources like authentication are also subject to pressures from other applications. The net result of sharing with all of the other applications is that the pool of available resources is changing second-to-second, based on demands from other applications that are almost certainly out of scope for the project.

Modern execution environments include virtualization software, storage firmware, operating systems, presentation frameworks, and many more software components. The trend of hardware is to be more and more abstracted by software. Not only does this increase the variability of available resources moment to moment, it means that there is a whole other class of environment configuration associated with software versions and patching.

### 3.3 Realism in Response Times

The nature of client software has also changed considerably. The most significant factors here include execution environments on mobile devices, the amount of code and interpreted code running at the client, and highly variable network conditions over wireless links. While server requests can be successfully abstracted by load tools, the reported response times do not include client rendering, which in many cases is now the majority of the human-experienced response time.

## 4 Performance Testing Iteratively

Simulation Tests as a final verification before go-live have difficult pre-requisites to meet in a waterfall process. In iterative development, quality feedback is desired throughout the process. Performance tests in this context have to yield information in minutes, not hours, in order to keep up with the speed of builds. To provide feedback at the speed of iterative development, it is necessary to make tests faster, simpler, cheaper, and more repeatable.

Once we've moved on from the expectation for simulated workloads to represent "Realism", we can design performance tests that meet these requirements. Instead of testing 10 workflows with specific numbers, proportions, and pacing of activity, we can test 1 at a time with tests designed for repeatability.

The concept I'd like to introduce here is Calibration – between builds, between environments, and between changed configurations. It is more important to track improvement and degradation as the project proceeds, and less important to attempt a specific projection of production experience. By tightening the feedback loop, we can provide actionable performance feedback while the changes that have been made are still well-known and fresh in people's minds.

## 4.1 Test Cases for Calibration

Test cases should be simple and easy to repeat. They should not seek to cover every permutation in the software, but to exercise the most commonly used paths in the software, and help reliably calibrate across builds of software.

Consider the example of establishing a user session. As software continues to be built, the login process for each user will include more and more code as a user session model becomes more sophisticated. As new elements are added, the efficiency and cost of processing the new elements can be evaluated. If a change is expensive, it can be quickly identified as such, and optimized or reworked.

As the available user session options increase, e.g., levels of application privilege, additional test cases can be added as appropriate.

## 4.2 Load Models for Calibration

Traditional approaches to performance testing use delays between activities, and between iterations. These delays serve an important purpose in spreading out load demand across a test, and make an individual session's flow more like a human-generated session.

Delays make it easy for the system under test to handle the load, and minimizes contention (See Section 2.3, Concurrency – Race Conditions and Contention). If we are testing to mitigate risk, that may be against our purpose. Delays make it harder to see small differences between two builds.

In some cases, we can better serve the goal of identifying small differences between builds by removing delays altogether. Consider a test of 10 threads, executing 10 iterations each. These 100 total iterations might consist of several steps each. Traditional performance testing spreads this test out over many minutes, using think times and pacing to make it easier for the server to handle.

Instead, we can start 10 threads with no delays and intentionally swamp the server. Then, we watch how long it takes the server to complete all the work, in addition to how long individual activities take. This is a more pure test of computation, by forcing the system to work as fast as it can throughout. This kind of test also helps mitigate concurrency risks, by creating more opportunities for collision between sessions.

Another approach would be 100 sessions run consecutively on a single thread with no delays. This will help expose even slight increases in response time by multiplying their effects.

## 4.3 Horizontal Scalability

Virtually every application uses a horizontal scaling pattern of multiple servers with load balancing to quickly increase capacity to the desired level (in addition to providing high availability). This pattern assumes 2 servers can support twice as many users as 1, and 4 can support four times.

This approach can be wound backwards. If production is going to support 500 users with 4 servers, 1 server should support 125. It is still necessary to validate the number of users 1 server can support in order to project how many servers to provision.

Testing with one server is easier to analyze, reduces the number of variables, and allows us to calibrate between software builds.

## 4.4 Testing Infrastructure

Many deployment contexts depend heavily on virtualization. This can greatly complicate repeatability. A recent desktop-class machine should be more than enough for calibrating results across builds and configurations reliably in a way that virtualization cannot match.

The goal shouldn't be "comparable" capacity – it should be reliable, repeatable capacity achieved through dedicated resources. By controlling variables tightly, we can trust our results now and in the future.

Load tools, separated components and services, even databases can be put on the same box as the primary system under test. This allows for the simultaneous measurement of all of the components at once, and removes the chance for network conditions or other demands to impact response times.

## 4.5 Performance Testing Tools and Performance Testing in CI

Performance testing tools are simply multi-threaded automation tools. They can be scripted to execute automatically, which means they can be added to a Continuous Integration (CI) process. Load Tool Vendors such as SOASTA and BlazeMeter are recommending their tools for this specific purpose.

Automating interpretation requires some thoughtfulness. What will be measured? What increase in response time should lead to further investigation (or turn a status yellow or red, depending on the process)? Who is the person who understands well enough what has changed in the last build that they can properly interpret and triage the change?

## 4.6 Extending existing/single-threaded automation

Any language used for automation should have the ability to record elapsed times between and during activities, even if it is of the form (endtime – starttime). This data can then be logged, providing a measurement of change across many builds. This also captures client-side and rendering time accurately, unlike load test tools.

Automation extends senses – but doesn't replace them. Watch for trends, and be ready to follow up and drill down when something has changed.

## 4.7 Recordkeeping

It is essential to be confident that results mean what they seem to mean. We can have that confidence if we can reproduce previous findings – or measure the delta. Environments and components change all the time. With careful records, we can reconstruct previous results, and use calibration techniques to measure the effects of specific components changing. We can also verify our instruments and environment by repeating and calibrating against previous tests.

# 5 Alternative Performance Testing Techniques

There are other techniques besides load testing for gathering performance feedback faster. The cost and startup time for getting load testing set up and started can be high. Some of these approaches can provide value that load testing can't. These techniques generally do not consider the resource characteristics of a loaded system or consider concurrency, but they instead look very closely at the responsiveness of a single user's interaction with the system.

## 5.1 Stopwatch

A simple, fast performance test is to use a stopwatch to record the time between a click, and when the software has completed servicing the request. These times can be retaken and reproduced very cheaply and quickly. This provides a quick, simple measurement of performance.

Quantifying user experience is frequently an implicit or explicit goal of load testing. Since load testing typically plays server requests back, it does not include client-side rendering and script execution time. Interacting with the software under test's interface captures client-side and rendering time accurately, unlike load test tools.

## 5.2 Extending Functional Automation

Adding timers to automation scripts is relatively simple. Setting (and abstracting) a threshold for flagging results is a more difficult. Recording the timers over time takes a little more work. The net result of this can be well worth the effort; responsiveness of these user activities can be tracked throughout the project.

When automation is GUI-based, instrumenting automation can provide user experience response times.

## 5.3 Screen Captures/Videos

Screen capture programs can record how software responds from a user perspective, preserving that exact experience indefinitely. For usability studies, this technique is invaluable; for performance testing without a load tool, it provides a mechanism for capturing user experience in a sharable way that is easy to share with non-technical stakeholders, and includes client-side and rendering time.

This technique can be leveraged with Load Testing to provide a specific representation of how the system performs at peak load, communicating real user experience in a way a table of response times cannot.

## 5.4 Log Files

Web-based systems write every request to a w3 log format, which can be configured to include server-response times (time-taken). Depending on the software under observation, other logging information might be available – or logging might be added early on as a supportability/testability story/requirement.

When a log file exists, it can be mined for data after the fact. It can be processed to generate summary data, and that summary data can be stored for future use in comparisons and trending.

## 5.5 Waterfall Charts

For websites and web applications, a waterfall chart can help understand the components of a web page. There are many ways to generate these, including free online sites like http://www.webpagetest.org/.

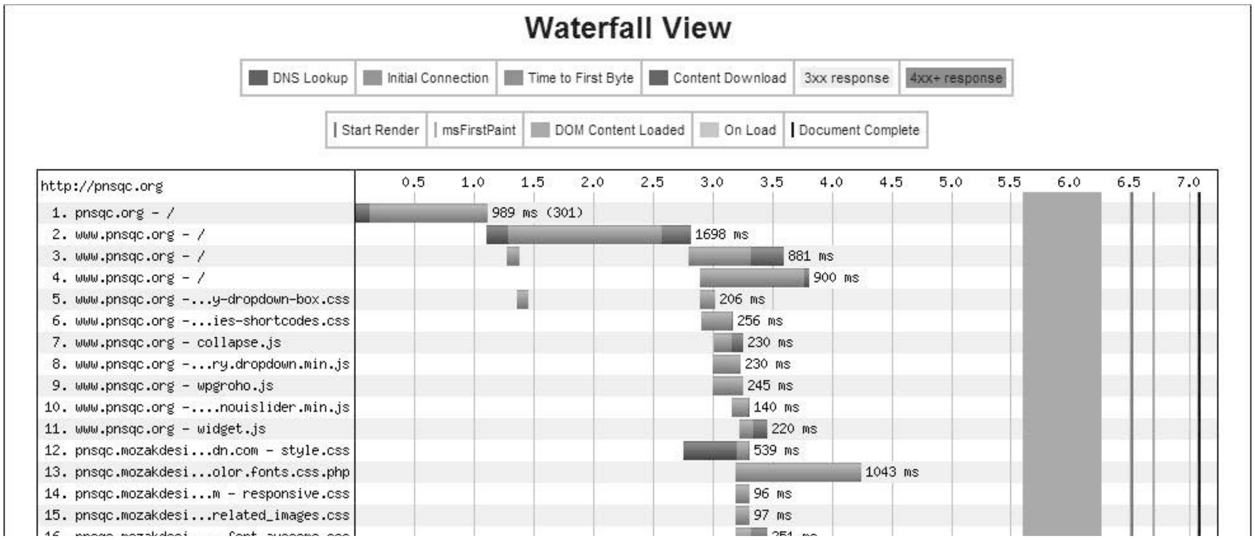
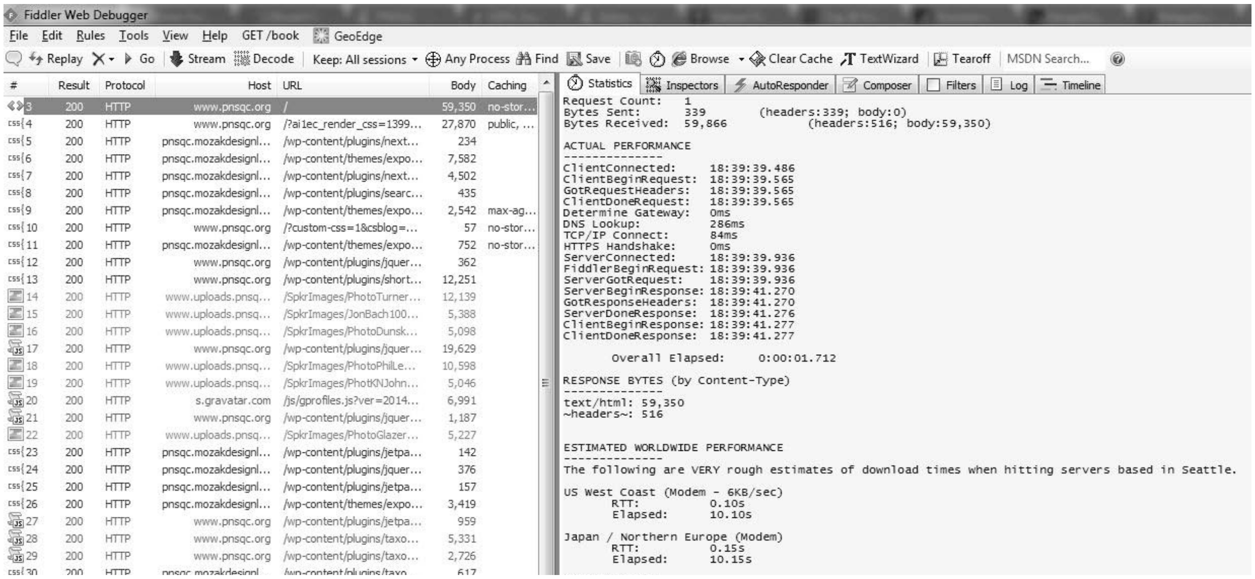


*Figure 1: Waterfall Chart*

These can be captured throughout the project, and used as a mechanism for tracking progress and identifying problem areas. Tools such as YSlow use this kind of data to suggest web page optimizations.

## 5.6 Browser Proxies

Browser proxies such as Telerik's Fiddler or Charles Proxy collect highly detailed data at the request level, including payload sizes and response times. Results can be compiled throughout a project.


Fiddler Web Debugger
File Edit Rules Tools View Help GET /book GeoEdge
Replay Go Stream Decode Keep: All sessions Any Process Find Save Browse Clear Cache TextWizard Tearoff MSDN Search...
Statistics Inspectors AutoResponder Composer Filters Log Timeline

| # | Result | Protocol | Host | URL | Body | Caching |
|---|---|---|---|---|---|---|
| 3 | 200 | HTTP | www.pnsqc.org | / | 59,350 | no-stor… |
| 4 | 200 | HTTP | www.pnsqc.org | /?ai1ec_render_css=1399… | 27,870 | public, … |
| 5 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/next… | 234 | |
| 6 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/themes/expo… | 7,582 | |
| 7 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/next… | 4,502 | |
| 8 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/searc… | 435 | |
| 9 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/themes/expo… | 2,542 | max-ag… |
| 10 | 200 | HTTP | www.pnsqc.org | /?custom-css=1&csblog=… | 57 | no-stor… |
| 11 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/themes/expo… | 752 | no-stor… |
| 12 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/jquer… | 362 | |
| 13 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/short… | 12,251 | |
| 14 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/PhotoTurner… | 12,139 | |
| 15 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/JonBach100… | 5,388 | |
| 16 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/PhotoDunsk… | 5,098 | |
| 17 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/jquer… | 19,629 | |
| 18 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/PhotoPhilLe… | 10,598 | |
| 19 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/PhotKNJohn… | 5,046 | |
| 20 | 200 | HTTP | s.gravatar.com | /js/gprofiles.js?ver=2014… | 6,991 | |
| 21 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/jquer… | 1,187 | |
| 22 | 200 | HTTP | www.uploads.pnsq… | /SpkrImages/PhotoGlazer… | 5,227 | |
| 23 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/jetpa… | 142 | |
| 24 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/jquer… | 376 | |
| 25 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/jetpa… | 157 | |
| 26 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/themes/expo… | 3,419 | |
| 27 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/jetpa… | 959 | |
| 28 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/taxo… | 5,331 | |
| 29 | 200 | HTTP | www.pnsqc.org | /wp-content/plugins/taxo… | 2,726 | |
| 30 | 200 | HTTP | pnsqc.mozakdesignl… | /wp-content/plugins/taxo… | 617 | |

```
Request Count:   1
Bytes Sent:      339        (headers:339; body:0)
Bytes Received:  59,866           (headers:516; body:59,350)

ACTUAL PERFORMANCE
--------------
ClientConnected:    18:39:39.486
ClientBeginRequest: 18:39:39.565
GotRequestHeaders:  18:39:39.565
ClientDoneRequest:  18:39:39.565
Determine Gateway:  0ms
DNS Lookup:         286ms
TCP/IP Connect:     84ms
HTTPS Handshake:    0ms
ServerConnected:    18:39:39.936
FiddlerBeginRequest: 18:39:39.936
ServerGotRequest:   18:39:39.936
ServerBeginResponse: 18:39:41.270
GotResponseHeaders: 18:39:41.270
ServerDoneResponse: 18:39:41.276
ClientBeginResponse: 18:39:41.277
ClientDoneResponse: 18:39:41.277

       Overall Elapsed:     0:00:01.712

RESPONSE BYTES (by Content-Type)
--------------
text/html: 59,350
~headers~: 516


ESTIMATED WORLDWIDE PERFORMANCE
--------------
The following are VERY rough estimates of download times when hitting servers based in Seattle.

US West Coast (Modem - 6KB/sec)
       RTT:            0.10s
       Elapsed:        10.10s

Japan / Northern Europe (Modem)
       RTT:            0.15s
       Elapsed:        10.15s
```

*Figure 2: Fiddler Capture*

## 5.7 Application Performance Management (APM)

APM is an approach to monitoring and measurement that captures response time and workload metrics from deployed software. By taking detailed measurements for every execution of the software, these tools can gather large aggregations of data at the provide insight into which pieces of a user activity take more time at any one point, and are responsible for increased response times.

These measurements are taken at the network level, at runtime (e.g., Java process instrumentation), as instrumentation included in code libraries, and increasingly, as defined business transactions. Feeding this data into big data platforms supports deep operational and analytical analysis, helping performance get better visibility outside of engineering groups and connect it more directly to productivity and revenue.

APM tools were initially conceived as an operations management tool, supporting performance troubleshooting of deployed software. Some of us immediately adapted APM for working with systems under test. Later tools were designed for Developers. Tools in this space include Compuware (DynaTrace, Vantage, etc), Riverbed's SteelCentral (formerly OPNET), AppDynamics, and New Relic.

APM approaches have replaced synthetic performance tests for large website such as Facebook and Bing. This is an exciting and disruptive technology that has already significantly changed how people think about and value performance testing.

# 6 Testing Incomplete Systems

Testing incomplete systems requires a strong knowledge of the system's architecture, understanding the difference between test coverage in functional and performance testing, and the ability to adapt tools to the job at hand. Most components have programming interfaces, which is a handy entry point for test code, particularly when it can be borrowed from developers and their test harnesses.

In this section, we'll consider a common architecture pattern – a Presentation Layer communicating via Web Service calls to a Business Logic Layer. These servers then communicate with the rest of the application, which includes a Message Bus, a Transactional Database and a Data Warehouse (with Metrics Capture Server), an Authentication Store, and an Application/Concurrent Processing Server. Diagrams and deeper analysis start on the next page.

## 6.1 Test Case Design

To design test cases, identifying key paths in the software as soon as they appear. Think about user activities like road systems – where are the highways? What will be meaningful to test throughout the project, and calibrate on?

One example is Login. Login/logoff and the related creation and destruction of session objects are expensive, and matter for all users. See Section 4.1 for more on this. Other examples include search and browse functions that are likely to appear early in the project, and are likely to represent key functionality for users.

## 6.2 Different Meanings of "Interface"

Seasoned automation developers have experienced the problem of brittle tests because of changing user interfaces. We can borrow from their experience and strategies to make tests that will last longer. We still need to assume that we will spend significant time maintaining and updating tests as software changes.

One easy place to start is at APIs (Application Programming Interfaces, or colloquially: Amateur Programming Interfaces). These are the same abstraction points developers typically use, meaning that in addition to be good points to measure at, they is likely to be documentation and test harness code already sitting around that we can adapt for performance tests.

When components are missing, techniques such as mocking, stubs, and auto-responders can be used to simulate components that are not in place yet. Here is another opportunity to talk with Developers about how they test these systems in progress, and repurpose existing code for test harnesses and automation.

## 6.3 Testing Pieces of a System

With these techniques, we can test pieces of a system. If the system is still being developed, or components are not yet assembled, we can still test what is there and assess how it performs and scales. Starting on the next page, we will discuss how to test parts and subsets of a system by examining the sample architecture.

## 6.4 Example Application – Presentation Layer

First, let's consider abstracting the Presentation Layer, and addressing the backend servers (and supporting pieces). The Business Logic layer is frequently the scaling bottleneck in this architecture pattern. Web services are big buckets of functionality usually running a lot of interpreted code, and a lot of code written for the specific project.

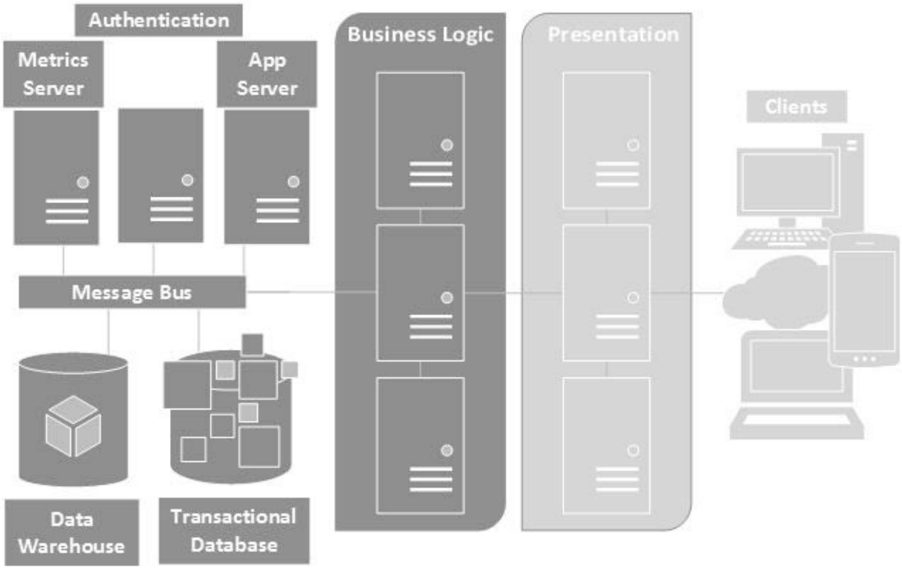


*Figure 3: Example Architecture*

We can test web services directly, making our tests more durable to changes to the user interface at the Presentation layer. We can abstract the presentation layer, even borrow code from it, and test the rest of the system back.

## 6.5 Example Component - Authentication

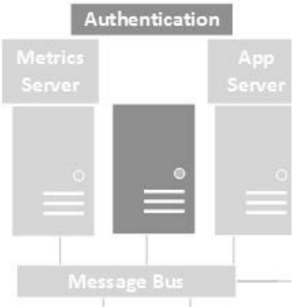


*Figure 4: Authentication*

At the opposite end of the spectrum from excluding one component is testing just one component.

For Authentication, we can first create a test that just creates sessions. Once we have that, we can answer some important questions about this essential component of the system, such as response time to create a session, the number of sessions that can be created, the arrival rate of sessions that can be supported, and the reliability of creating sessions.

### 6.6 Example Component: Message Bus

Many systems implement a Message Bus and publish-subscribe methods for communicating to other components. It is typically designed to make it easy to create and consume messages, so that components can easily be connected.

Message Bus

*Figure 5: Message Bus*

Architectural assumptions are made at this stage that must be correct for the project to be successful. As a key component and potential scalability challenge, this is an opportunity to reduce risk and add value to the project very early on.

First, find or make a test harness that allows creation (push) and consumption (pull) of messages. A good place to start is to ask how developers test. Once this harness is in hand, you can start to track metrics and answer important questions.

What is the response time to push a message? What is the response time to pull a message?

At what rate can messages be created? With one publisher? With multiple publishers?

At what rate can messages be consumed? What if message servers/dispatchers are added?

# 7 Communicating Performance During Iterative Development

Performance Testing generates a lot of data, and too frequently, stops there. Turning over large chunks of data generally isn't helpful to stakeholders, yet it continues to be a frequent output format from performance testing, jeopardizing value by making too difficult to understand what happened.

To get value from a test, we don't need data, we need Information. Analysis of data leads to Information – knowledge that we can trust and act on, and that our stakeholders are able to trust and act on.

Your value as a tester is the information you provide, and the action(s) it inspires. After information is generated, it needs to be shared. Make results visible, and recruit consumers of your information. Become an important project status indicator. Help people who can do something about it understand and care about the information you provide.

Here are some strategies for sharing information. Consider using all of them to share what is learned, and to continually demonstrate the value you are adding to the project.

### 7.1 Informal Communication

Meetings with peers and supervisors in attendance can be a tough place to receive feedback gracefully. It's also a terrible place to be wrong if you are providing that feedback, as it will impact your credibility and make it harder for you to be heard.

If you are confident in your information and the venue is appropriate, you may be willing to be declarative, certain it will be received well and you are prepared to handle any follow-up questions or challenges. If you are not sure what you see, or how it will be consumed, you might instead choose a more cautions path. Approach a key engineer and ask, "Can I show you something?" Vet findings before you broadcast.

### 7.2 Information Pushes

Once you have actionable information, push it out to the team. Emails about results are frequently used, but is critical to remember to start with a short summary that can be scanned on a mobile device. Information sources that are unclear or of low value are likely to be ignored.

Other venues for pushing out information are status reports, stand-up meetings, and gossip/grapevines. When you work in performance, you will probably be asked "How is performance these days?"

## 7.3 Reporting Frequency and Feedback Loops

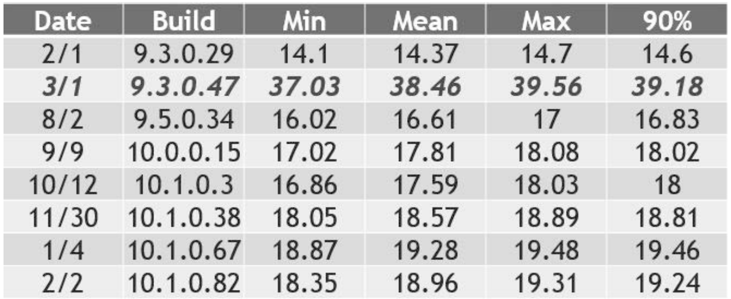
A frequently encountered challenge with Simulation Tests is that testing doesn't take place until software is completed. Waiting until software is complete means that testing occurs at the very end of the project. This exposes testing to time pressure, particularly when earlier phases of the project run past milestones.

Finding problems at a point in the schedule where the testing is the last gate to going live is of limited value. Either the problems are minor enough to be lived with and are not fixed, or serious enough to delay the entire project. Even positively identifying the source of the problems may be too time consuming to delay go-live. This decision is rarely made in a vacuum; stakeholders or team members who feel prestige or jobs are at stake may not think like engineers.

With iterative-speed performance testing, we should try to report as often as we can. Even if it we can't retest every single sprint, we reduce the cost of rework by reporting performance and reliability issues as soon as possible.

## 7.4 Information Radiators

Intranet pages and whiteboards are two ways to make information available permanently, in a way that supports self-service consumption. Consider ways to present information that make it easy for consumers to understand progress over time, whether are deeply involved with the project or not.



| Date | Build | Min | Mean | Max | 90% |
|---|---|---|---|---|---|
| 2/1 | 9.3.0.29 | 14.1 | 14.37 | 14.7 | 14.6 |
| ***3/1*** | ***9.3.0.47*** | ***37.03*** | ***38.46*** | ***39.56*** | ***39.18*** |
| 8/2 | 9.5.0.34 | 16.02 | 16.61 | 17 | 16.83 |
| 9/9 | 10.0.0.15 | 17.02 | 17.81 | 18.08 | 18.02 |
| 10/12 | 10.1.0.3 | 16.86 | 17.59 | 18.03 | 18 |
| 11/30 | 10.1.0.38 | 18.05 | 18.57 | 18.89 | 18.81 |
| 1/4 | 10.1.0.67 | 18.87 | 19.28 | 19.48 | 19.46 |
| 2/2 | 10.1.0.82 | 18.35 | 18.96 | 19.31 | 19.24 |

*Calibration Results for Web Login, Search, View. Burst 10 threads iterating 10 times*

# 8 Conclusions/Takeaways

I believe that these are the most important takeaways from this paper:

1. Traditional Performance Testing was originally conceived as a gate to production for completed software, depending on a perception of "realism" to prove value to stakeholders.
2. Iterative Development Processes caused functional testing to take place earlier and more often. Non-functional testing (performance, reliability, etc.) should follow.
3. Performance Testing addresses specific risks. These risks can be tested against with "unrealistic" load tests just as well (or better) than with "realistic" tests. Spend time thinking and talking about "realistic" if you encounter resistance.
4. Simple test cases and scripts help make tests cheaper and faster to run, allowing measurements to take place much more often. Once you have a reliable measurement, you can calibrate with these measurement between builds, servers, components, and environments, and revisit previous results to ensure that all of the performance variables are understood.
5. There are other performance test techniques besides load testing. Consider using them to provide immediate feedback.

6. Incomplete Systems can be performance tested by analyzing the components of the system, testing individual and partial combinations of these components, and testing what is there, as it arrives.
7. Communicating performance feedback, actively and passively, is essential for performance testing effectively. Information that does not have any impact doesn't add value.

# 9 References

Many of these concepts were discussed at WOPR22 (The 22nd Workshop on Performance and Reliability, performance-workshop.org). WOPR22 was held May 21-23, 2014, in Malmö, Sweden, on the topic of "Early Performance Testing". Participants in the workshop included Fredrik Fristedt, Andy Hohenner, Paul Holland, Martin Hynie, Emil Johansson, Maria Kedemo, John Meza, Eric Proegler, Bob Sklar, Paul Stapleton, Andy Still, Neil Taitt, and Mais Tawfik Ashkar.

The rules of attribution for LAWST-style and LAWST–inspired workshops include a requirement that any publications from the meeting will list all attendees as contributors.

# Effective Security Testing

**Author: Jeyasekar Marimuthu**

Jeyasekar.m@gmail.com

## Abstract

The number of software vulnerabilities exposed in recent times and the magnitude of impacts they have on customers is the key reason why effective security testing is required for software development organizations.

When proper security measures are not taken, a company not only puts their reputation at stake, this also puts their customers and their customers' sensitive data at risk. The magnitude of a data breach will vary - from financial loss, to loss of life - depending on the software deployment. Data exploitation in any form through software vulnerabilities affects the credibility of system and the company.

Effective security testing is achieved by and by understanding security at deeper levels and adopting key security measures in the PLF (Project Life Cycle). Software experts write tests for each layer, then measured via manual and tool based audits.

It is important to plan measures targeting the prevention of security breaches. Some of the known practices to prevent security breaches in software include vulnerability scans, security assessment, penetration tests, and security audits. Then, when issues arise, it is important to deploy counter measures before the problem magnifies.

This paper discusses various forms of software vulnerabilities and the measures to be taken to prevent them. The paper also details out some of the key security tools recommended for identifying security flaws.

## Biography

*Jeyasekar Marimuthu is a lead software developer at Intel Security, Beaverton (formerly McAfee) with comprehensive experience in architecting and developing solutions, project execution, process improvement, cross product development, and client relations. His strengths are: understanding business requirements, evaluating risks, providing architecture and designs, and strategizing profitable execution. A skilled developer and technical leader, Jeyasekar communicates effectively with management, vendors, team members, and staff at all skill levels. He brings innovation with ideas and drives improvements in development and processes. He has 14 years of Development experience in Java, J2EE, database technologies, and B2B/ Enterprise-security products.*

# 1. Introduction

Every organization that deals with sensitive data should expect exploits from all possible directions. Widely classifying, the attacks target external facing systems, hosted applications, client side applications, company networks and individuals. It is important to understand the various kinds of attacks that have occurred and the damages they have caused.

Exploits can happen through all sorts of medium, security testing is not a well-defined term and it can never be fool proof. How can companies be smart in investing in right areas and protect their core values?

It is impossible to prevent every possible attack and it is a daunting task for organizations to block every possible channel.

This paper focuses security testing on the exploits. If attacks are expected then the magnitude of impacts due to the exploit should be as minimal as possible. Let us see how.

# 2. Understand the organizational need

Companies should prioritize their important assets and start protecting them based on their importance. It is wise to minimize the impact of exploitation on critical systems before it happens. The difference between security testing and an advanced security testing is similar to black box vs white box testing. In black box testing, one would look for the valid and invalid functional tests. In white box testing, it is required to understand the internal perspective of the system to write and perform tests. It is difficult to quantify the level of system protection without understanding the applicable vulnerabilities and the possible damages they may cause.

The rules of thumb for advanced security testing:

1. Study the recent disasters and motives behind each attack.
2. Evaluate the damages of each targeted attack and understand the consequences after attack.
3. Pick the relevant exploits that are applicable to your domain or area of operations.
4. Study the consequences of exploits on assets, networks, company reputations, customer retention etc.
5. Prioritize the assets that need to be protected according to business and operation needs.
6. Make your investments on security products wisely.

Understand that not all security items can be implemented; some systems may have no countermeasures for attacks. Choose the one that is most important to you and address the vulnerabilities as effectively as possible.

# 3. Need for security testing: A quick overview

In order to implement the rules of thumb, an organization must study the type of vulnerabilities and possible attacks. Without having a deeper understanding, it will not be easy to choose the relevant items that are applicable to the system or organization.

Types of attack/Exploits:

- Social engineering
- Session hijacking
- Network mapping
- Fuzzing
- Zero day attack

- SQL Injection
- Password Cracking
- DNS Poisoning
- ARP Poisoning
- Wardriving

Type of vulnerabilities:

- Arbitrary Code Execution
- Buffer Overflow
- Code Injection
- Heap Spraying
- Web Exploitation (client-side)
- Cross-site scripting
- HTTP header injection
- HTTP Request Smuggling
- Web Exploitation (server-side)
- DNS Rebinding
- Clickjacking
- CSRF

A short summary of 2013 disasters:

- Social media giants Facebook, LinkedIn, among others, get hacked repeatedly
- Nearly 40 million Target customers' credit and debit card numbers were stolen in the midst of the holiday shopping rush.
- Hacker group anonymous targets Twitter accounts.
- Adobe breach snowballs into multi-network security risk.
- System bug exposes 6 million Facebook users' personal data in yearlong breach.
- Upwards of 50 million LivingSocial user emails and passwords are stolen.
- Evernote resets about 50 million account passwords after data breach.
- The U.S. Department of Homeland Security finally corrected a four-year error in the software it uses to process employees' background checks.
- Federal Reserve Bank website hacked by Anonymous.

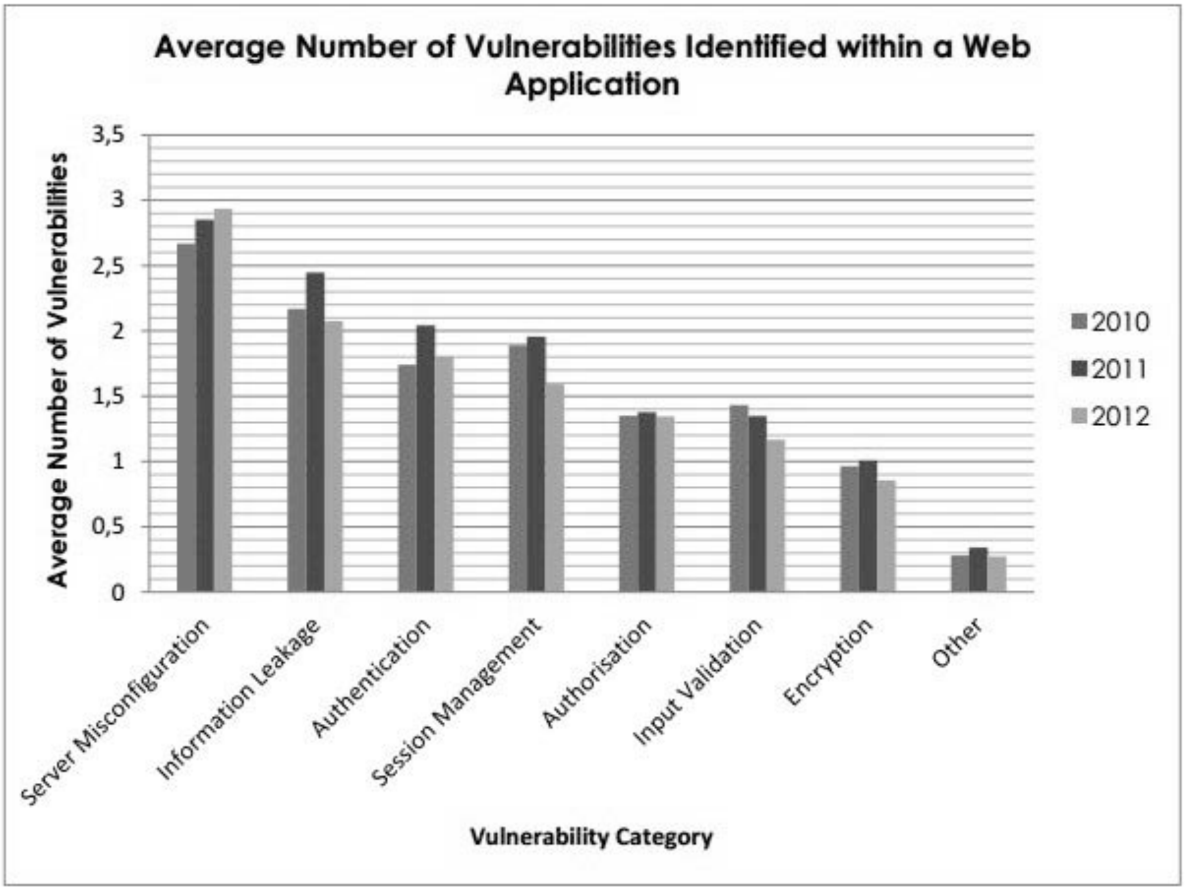
These breaches fall into 4 broad categories.

- **Network security**: This involves looking for vulnerabilities in the network infrastructure (resources and policies).
- **System software security**: This involves assessing weaknesses in the various software (operating system, database system, and other software) application that the system depends on.
- **Client-side application security**: This ensures the client such as browser cannot be manipulated.
- **Server-side application security**: This makes sure that the server code and its technologies are robust enough to fend off any intrusion.

Web application findings summary 2013:

Server-side application security is something test professionals need to pay more attention to.  The Web Application Vulnerability chart below shows the category of breaches.

| **Category** | **Description** |
|---|---|
| Server configuration | Insecure server configuration settings that result in security |
| Information leakage | Information leaked by the application that could be used<br>by an attacker to help mount an attack |
| Authentication weaknesses | Issues related to the application's authentication mechanism that could be exploited by an unauthenticated attacker to gain or assist in the gaining of authenticated access |
| Session management<br>weaknesses | Session management issues that could allow an attacker to hijack or assist in the hijacking of other users' sessions |
| Authorization weaknesses | Issues concerning access controls that could allow an attacker to perform either horizontal or vertical privilege escalation |
| Input validation weaknesses | Issues created by weaknesses in input validation processes. |
| Encryption vulnerabilities | Issues that concern the confidentiality of data during<br>transport and in storage |
| Other | Any other issues identified that do not fit into the categories listed above |



## 4. Effective tests

The previous section highlighted some of the key vulnerabilities and exploits. Based on the facts presented, prepare the applicability score against each of them and see what tops your findings. Pick those top 3 or 5 items from each vulnerability and exploit category and identify the tools and man-power required to perform those tests.

A sample Security Assessment sheet would look as follows.

| Type of vulnerability | Applicable? | Estimated damage (in $) | Impact on Business sustainability |
|---|---|---|---|
| Buffer overflow | Yes | 2 Million | Medium |
| Code injection | Yes | 2 Million | Medium |
| Cross Site scripting | Yes | 10 Million | High |

The sample Security Assessment table helps to prioritize the items to focus. The next big question is how to determine whether the vulnerability is applicable to your organization or not. Companies like Intel Security offer various vulnerability detector tools that can detect the vulnerabilities easily. There are even manual ways to identify the flaws in the system.

According to ISACA (Information Systems Audit and Control Association), large-scale, covert penetration tests can be an effective tool for governments, private companies, and other national and international organizations to assess the security of their critical resources. According to the US National Institute of Standards and Technology, the purpose of covert security testing is to “examine the damage or impact an adversary can cause,” rather than to identify specific vulnerabilities.

The penetration test team should target to find the following vulnerabilities.

The majority of the vulnerabilities exploited could be categorized as follows:

- Weak and/or unchanged default passwords
- Default system and application configurations
- Failure to patch known vulnerabilities and use secure configurations enterprise-wide
- Failure to consistently apply the least-privilege access control model
- Failure to use secure coding
- Lack of security awareness by system users

# 5. Tools that can detect specific vulnerabilities

There are many tools on the market that detects specific vulnerabilities. Some of these are listed below.

## 5.1 Testing for DOM XSS

- DOMinator Pro - https://dominator.mindedsecurity.com

## 5.2 Testing AJAX

- OWASP Sprajax Project

## 5.3 Testing for SQL Injection

- OWASP SQLiX
- Sqlninja: a SQL Server Injection & Takeover Tool - *http://sqlninja.sourceforge.net*
- Bernardo Damele A. G.: sqlmap, automatic SQL injection tool - *http://sqlmap.org/*
- Absinthe 1.1 (formerly SQLSqueal) - *http://sourceforge.net/projects/absinthe/*
- SQLInjector - Uses inference techniques to extract data and determine the backend database server. *http://www.databasesecurity.com/sql-injector.htm*
- Bsqlbf-v2: A Perl script allows extraction of data from Blind SQL Injections - *http://code.google.com/p/bsqlbf-v2/*
- Pangolin: An automatic SQL injection penetration testing tool - *http://www.darknet.org.uk/2009/05/pangolin-automatic-sql-injection-tool/*
- Antonio Parata: Dump Files by sql inference on Mysql - SqlDumper - *http://www.ruizata.com/*
- Multiple DBMS Sql Injection tool - SQL Power Injector - *http://www.sqlpowerinjector.com/*
- MySql Blind Injection Bruteforcing, Reversing.org - sqlbftools - *http://packetstormsecurity.org/files/43795/sqlbftools-1.2.tar.gz.html*

### 5.4 Testing SSL

- Foundstone SSL Digger - *http://www.mcafee.com/us/downloads/free-tools/ssldigger.aspx*

### 5.5 Testing for Brute Force Password

- THC Hydra - *http://www.thc.org/thc-hydra/*
- John the Ripper - *http://www.openwall.com/john/*
- Brutus - *http://www.hoobie.net/brutus/*
- Medusa - *http://www.foofus.net/~jmk/medusa/medusa.html*
- Ncat - *http://nmap.org/ncat/*

### 5.6 Testing Buffer Overflow

- OllyDbg - A windows based debugger used for analyzing buffer overflow vulnerabilities - *http://www.ollydbg.de*
- Spike - A fuzzer framework that can be used to explore vulnerabilities and perform length testing *http://www.immunitysec.com/downloads/SPIKE2.9.tgz*
- Brute Force Binary Tester (BFB) - *http://bfbtester.sourceforge.net* A proactive binary checker
- Metasploit - *http://www.metasploit.com/* - A rapid exploit development and Testing frame work

### 5.7 Fuzzer

- OWASP WSFuzzer
- Wfuzz - *http://www.darknet.org.uk/2007/07/wfuzz-a-tool-for-bruteforcingfuzzing-web-applications/*

## 6. Best practices / Countermeasures

After detecting vulnerabilities, it is important to fix them by adopting some best practices and deploying countermeasures.

- Form a dedicated security audit team that can periodically conduct audits and report security issues.
- Make sure all systems in the organization are behind the firewall.
- Make sure all systems are patched periodically.
- Allow network traffic only via firewall.
- Make sure you have a well-defined security policy for your org and enforce the rules on each system via products like McAfee’s ePolicy Orchestrator.
- Deploy countermeasures on all critical systems.
- Educate employees about phishing attacks through phone, emails and alert them to exercise caution.
- Make sure every project release is certified by your Security testing experts.

## 7. Qualities of Good Security Testers

A security tester is preferred to have thorough understanding on all types of attacks and their potential impacts on a system. They should analyze the SuD (System under development) in length and breadth and should be able to identify the areas that need to be tested against a bunch of security tests. A person with certification like CISSP/CompTIA Security+ would be a good fit. More importantly, she/he should follow the recent security breaches and should be able to map the breaches with the system they work on.

A security test plan capturing the areas and applicable tests should be rolled out as soon as the system design is ready. A security tester is expected to have mastery in one or more security tools that can detect and analyze potential weakness in the system. She/he shall additionally have white box testing abilities to conduct manual security audits on certain portion of code where tools have limited reach. She/he may urge organizations to treat security violations with high priority/important.

## 8. Management's Role

Having a great security tester without management support can be like having no tester at all. Management has the primary role in implementing good security testing. Management must balance coverage of the product(s), budget, and time required to implement quality measures.

Security testing needs to be treated as critical in risk management. When the breach happens, the amount of damage (loss of trust, reputation) that can occur from an uncaught vulnerability is much higher than any functional product defect.

It is wise to have one or more full time dedicated security analyst(s) working on projects. No project should be allowed to run without having a security test plan in place. A dedicated team of testers should have a role throughout the project life cycle. It is also good to have measures to evaluate the efficiency of security analyst and testers so that the quality of security testing constantly improves.

## 9. Conclusion

This paper has discussed the importance of conducting an effective security testing and the measures to identify the applicable areas for an organization. Various vulnerabilities and attacks discussed in the paper would serve as good reference points to get educated and to pick a right tool. Adopting the best practices and guidelines is vital to anybody who wants to keep the attackers and hackers at bay. Having all best practices followed and audited the system with relevant tools, the amount of damages that could have caused is significantly reduced. Despite all, keep in mind one thing. No system is fool proof. A continuous monitoring of security flaws only help for betterment of any system.

## 10. References

- http://softwaretestingfundamentals.com/security-testing/
- http://thinkprogress.org/security/2013/12/31/3108661/10-biggest-privacy-security-breachesrocked2013/
- http://www.isaca.org/Journal/Past-Issues/2012/Volume-2/Pages/Security-Through-Effective-Penetration-Testing.aspx
- https://tysonmax20042003.wordpress.com/tag/types-of-exploits/

# How Identity and Access Management Can Enable Business Outcomes and Enterprise Security

**Srikanth Thanjavur Ravindran**

**Suresh Chandra Bose, Ganesh Bose**

Srikanth.ThanjavurRavindran@cognizant.com

SureshChandra.GaneshBose@cognizant.com

## Abstract

In today's consumer driven enterprises, infrastructures are remote and distributed while business and IT operations are pervasive through mobile and virtual technology. Customer collaboration and communication on social platforms and disruptive service models such as (Bring Your Own Device) BYOD and Bring Your Own Application (BYOA) have broadened the security perimeter and increased the risk exposure points. In this scenario, the need for an identity and access management (IAM) solution has become paramount and is a top agenda item for most chief information officers (CIOs). In this session we will discuss key focus areas to establish holistic IAM solutions such as effective governance, automated role management, authentication, user profiling and integration.

## Biography

***Srikanth Thanjavur Ravindran** is a Senior Consultant with Cognizant's Business Consulting practice. Ravindran has diverse global experience with multiple Fortune 500 companies within the technology, telecommunications, oil & gas domains. His specialties include IT transformation, IT Strategy, ITSM, IT governance, risk management, information security, service delivery and infrastructure program management. He has published papers on topics such as BYOD, Identity and Access Management and IT Service Management at prestigious forums like ISACA and ITSMF.*

***Suresh Chandra Bose, Ganesh Bose** is a Manager Consulting at Cognizant Business Consulting practice. Suresh is based out of Austin, Texas and has been in the IT Industry for more than 16 years with vast Consulting experience in various industries and executed Strategic initiatives for various Fortune 100 companies in the areas of PMO, PPM, Program Management, TMMI assessment/implementation, Organization Strategy, Test Consulting and CIO/Governance Dashboard/Metrics for various clients across the globe.*

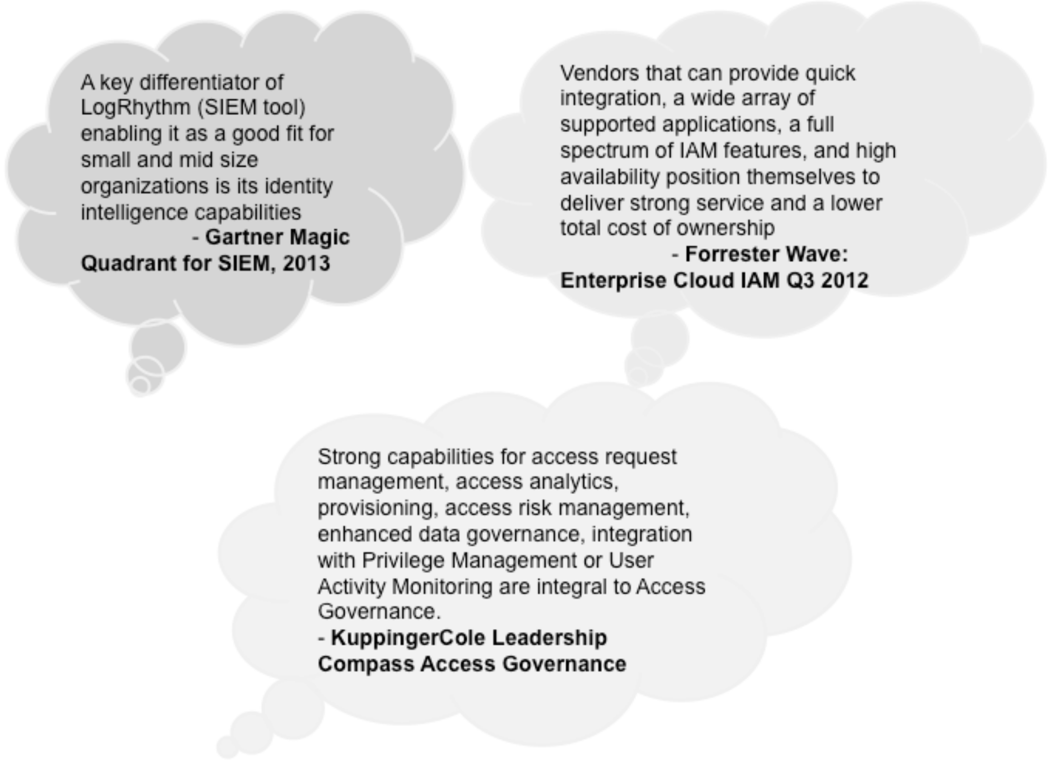
# 1 Introduction and Evolution into Identity Governance

The means of measuring business benefits of Identity and Access Management (IAM) initiatives and the ROI from IAM solution investments have been debated without much consensus in boardrooms virtually since the inception of technology. This has led to hesitation on the part of sponsors to endorse IAM undertakings leading to unrealistic deadlines, insufficient budgets and over worked staff.

During the past few years though, business agility has intensified, fueled by technologies such as cloud computing, remote infrastructures, mobility and BYOD / BYOA. IT has undergone shifts of its own due to disruptive service models in the form of multivendor outsourcing, Software as a Service (SaaS), multi-tenancy and virtual infrastructures. These shifts have increased the fear of the unknown due to organizational, customer, financial and IP data moving outside the organization's security parameter. These changes have also given rise to a plethora of regulations with heightened penalties for noncompliance. All this has contributed to transform IAM from a process for managing access to data and systems to a governance mechanism. Some of the key objectives of IAM in today's environment are:

- Service delivery to the business across hosted, remote, physical and virtual infrastructures
- Secure collaboration with customers, partners and employees
- Technology provision using access models spanning across web, mobile and application programming interfaces
- Role management through role definitions, user groups, identity verification, and authentication
- Compliance with regulatory requirements through personal data security, enhanced access control and privileges management
- Allow line of businesses (LOBs) to simplify access decisions based on trend analysis and save costs by rationalizing system licenses

# 2 Key considerations for success in IAM



A key differentiator of LogRhythm (SIEM tool) enabling it as a good fit for small and mid size organizations is its identity intelligence capabilities
**- Gartner Magic Quadrant for SIEM, 2013**

Vendors that can provide quick integration, a wide array of supported applications, a full spectrum of IAM features, and high availability position themselves to deliver strong service and a lower total cost of ownership
**- Forrester Wave: Enterprise Cloud IAM Q3 2012**

Strong capabilities for access request management, access analytics, provisioning, access risk management, enhanced data governance, integration with Privilege Management or User Activity Monitoring are integral to Access Governance.
**- KuppingerCole Leadership Compass Access Governance**

**Figure 1: Voices In The Industry**

Following are some essentials for a successful implementation of IAM:

**Figure 2: Implementation Considerations**

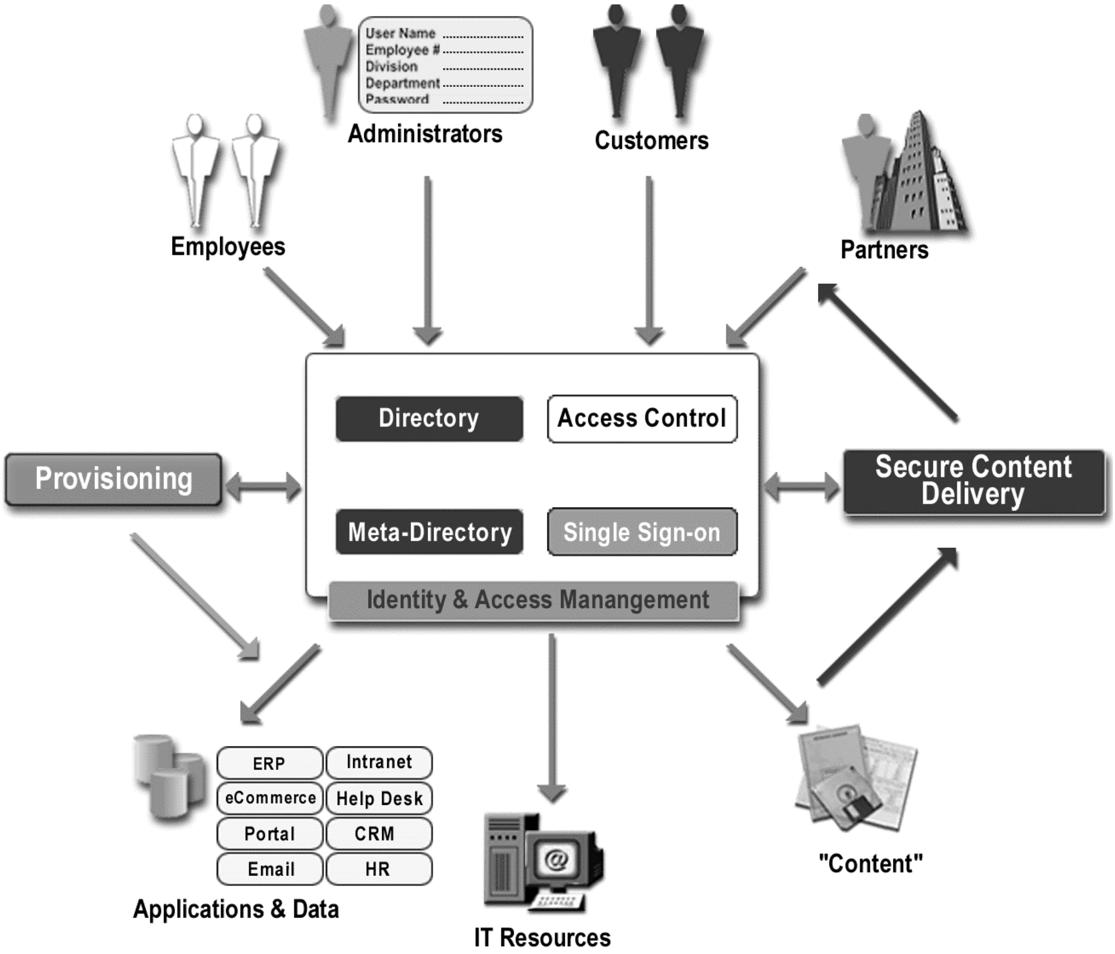


**Figure 3: Critical Success Factors**

## 2.1 Role Policy Management and Provisioning

- Provides real-time enforcement of policy/rule and role based user access to functional and data levels across all applications within an organization
- Provides centralized administration of roles/policies/rules/resources
- Serves as a central repository of role/policies/rule/resources, thus simplifying business intelligence and compliance audit data gathering efforts, and helps to create a more complete view of user access in an organization
- Supports enforcement of Segregation of Duties (SoD) preventing conflict of interest situations

## 2.2 Access Certification and Governance

- Automates discovery, analysis and management of user access rights
- Employs Least Privilege Access Principle (i.e. users are given access to only what is required to perform their job function)
- Facilitates periodic review and re-certification of access by business managers and data owners
- Ensures manual or automated remedial action to rectify access rights exceptions inconsistent with policy or regulatory requirements
- Demonstrates compliance with applicable regulations or business policies
- Enables multi factor authentication such as biometrics for sensitive data or PII (personally identifiable information)

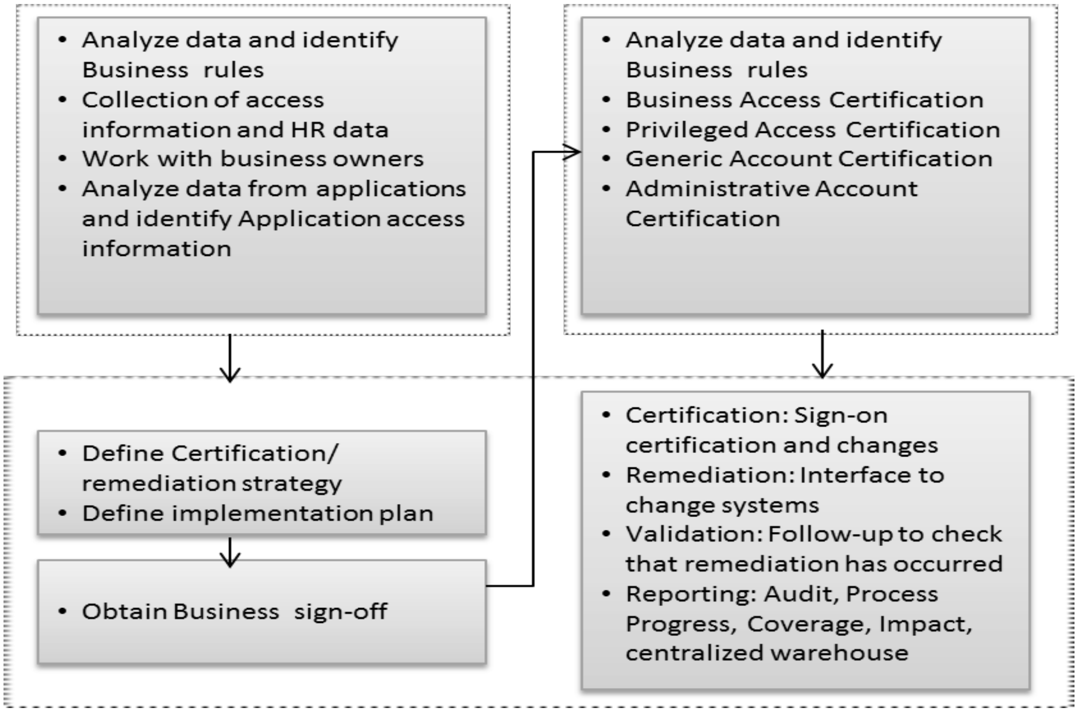

• Analyze data and identify Business rules
• Collection of access information and HR data
• Work with business owners
• Analyze data from applications and identify Application access information
• Analyze data and identify Business rules
• Business Access Certification
• Privileged Access Certification
• Generic Account Certification
• Administrative Account Certification
• Define Certification/ remediation strategy
• Define implementation plan
• Obtain Business sign-off
• Certification: Sign-on certification and changes
• Remediation: Interface to change systems
• Validation: Follow-up to check that remediation has occurred
• Reporting: Audit, Process Progress, Coverage, Impact, centralized warehouse

**Figure 4: Access Governance**

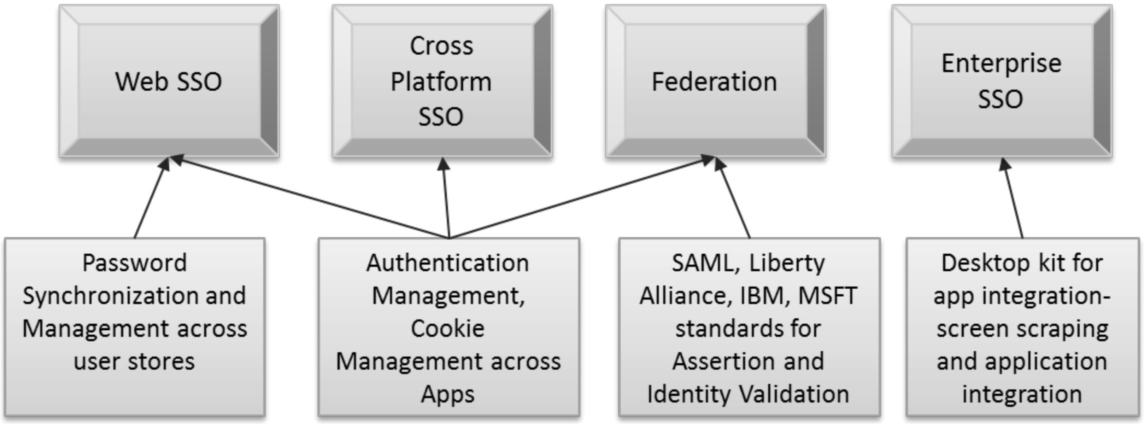
## 2.3 Single Sign On (SSO)

- Employs identity federation for securely sharing digital identities with customers, partners and remote users across platforms
- Captures identities and record audit trails for web services transactions

- Externalizes entitlement logic from applications and achieves centralized security avoiding need for building security into individual applications thereby reducing complexity
- Enables secure delivery of service and cost effective online collaboration

### 2.4 Broad tenets of SSO:

- Web
- Cross Platform
- Federation
- Enterprise

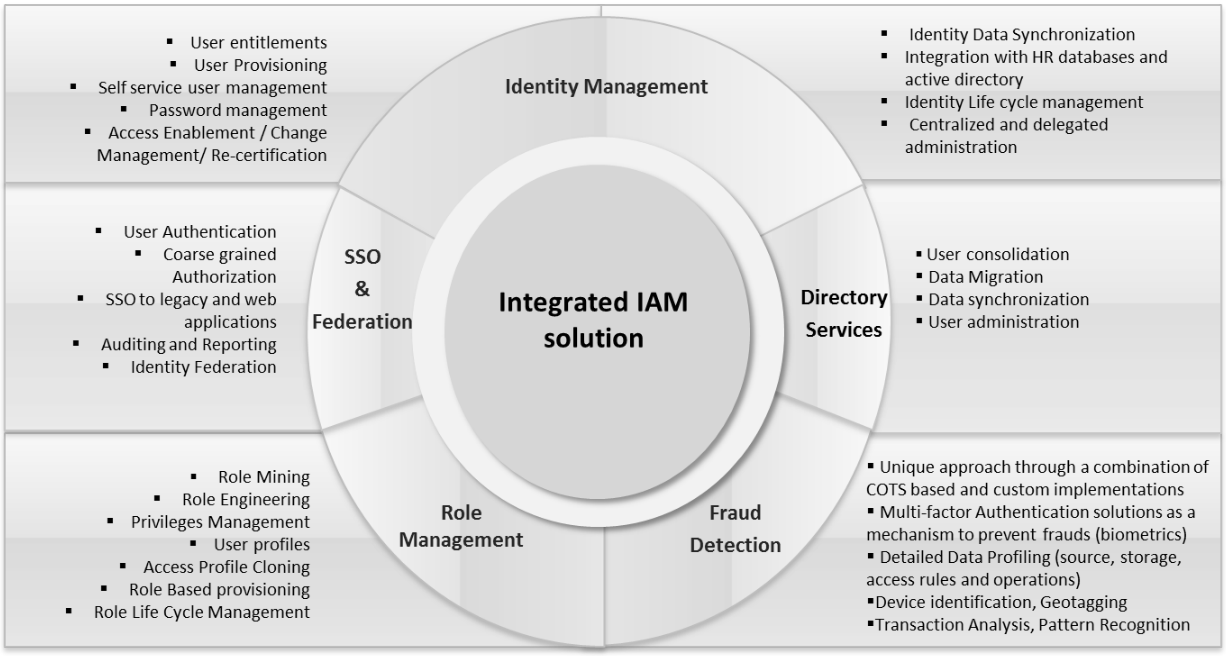


Web SSO
Cross Platform SSO
Federation
Enterprise SSO
Password Synchronization and Management across user stores
Authentication Management, Cookie Management across Apps
SAML, Liberty Alliance, IBM, MSFT standards for Assertion and Identity Validation
Desktop kit for app integration-screen scraping and application integration

**Figure 5: Single Sign On**

# 3 Business case for automation of IAM

Critical success factors for automation



▪ User entitlements
▪ User Provisioning
▪ Self service user management
▪ Password management
▪ Access Enablement / Change Management/ Re-certification
Identity Management
▪ Identity Data Synchronization
▪ Integration with HR databases and active directory
▪ Identity Life cycle management
▪ Centralized and delegated administration
▪ User Authentication
▪ Coarse grained Authorization
▪ SSO to legacy and web applications
▪ Auditing and Reporting
▪ Identity Federation
SSO & Federation
Integrated IAM solution
Directory Services
▪ User consolidation
▪ Data Migration
▪ Data synchronization
▪ User administration
▪ Role Mining
▪ Role Engineering
▪ Privileges Management
▪ User profiles
▪ Access Profile Cloning
▪ Role Based provisioning
▪ Role Life Cycle Management
Role Management
Fraud Detection
▪ Unique approach through a combination of COTS based and custom implementations
▪ Multi-factor Authentication solutions as a mechanism to prevent frauds (biometrics)
▪ Detailed Data Profiling (source, storage, access rules and operations)
▪Device identification, Geotagging
▪Transaction Analysis, Pattern Recognition

**Figure 6: Automation Components**

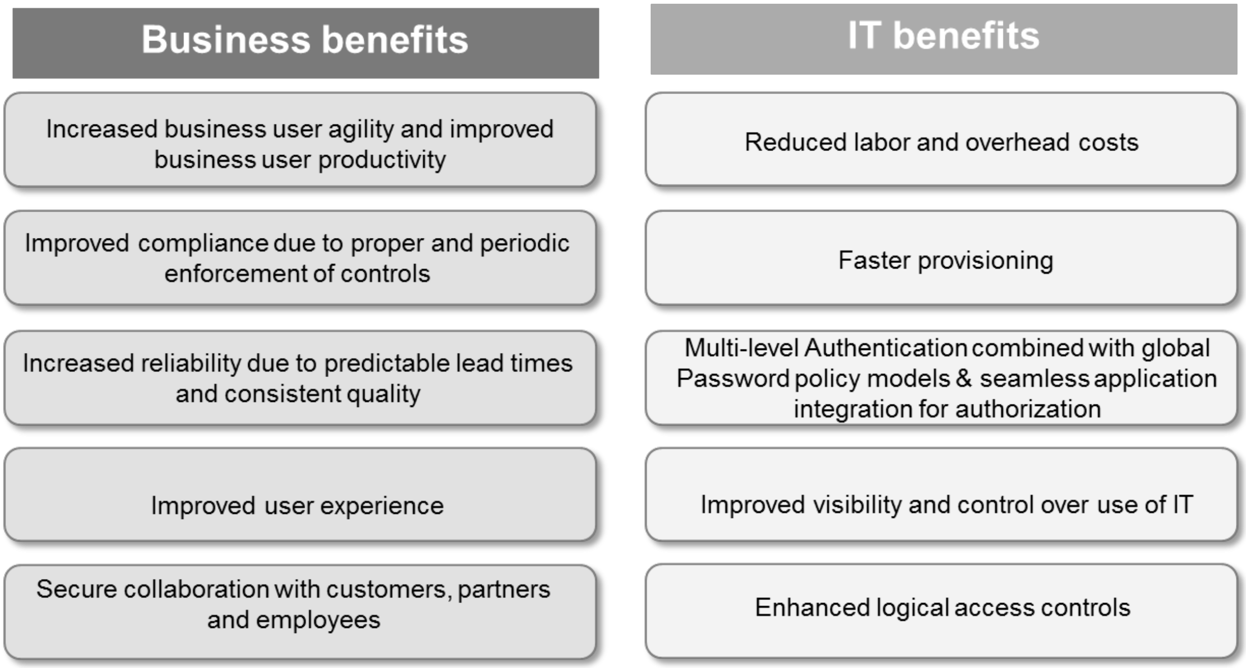

Business benefits
IT benefits
Increased business user agility and improved business user productivity
Reduced labor and overhead costs
Improved compliance due to proper and periodic enforcement of controls
Faster provisioning
Increased reliability due to predictable lead times and consistent quality
Multi-level Authentication combined with global Password policy models & seamless application integration for authorization
Improved user experience
Improved visibility and control over use of IT
Secure collaboration with customers, partners and employees
Enhanced logical access controls

**Figure 7: Business & IT benefits**

# 4 Illustrating and augmenting through measures

While IAM is a complex initiative to identify measures for, analytics and measurements can go a long way in helping the organization achieve their business objectives. Types of metrics that can be measured are:

- Metrics that contribute to achieving customer outcomes and influencing them
- Metrics that impact financial performance and
- Metrics that monitor compliance
- These can be monitored at multiple levels such as strategic, tactical and operational to obtain a top-down 360 degree view

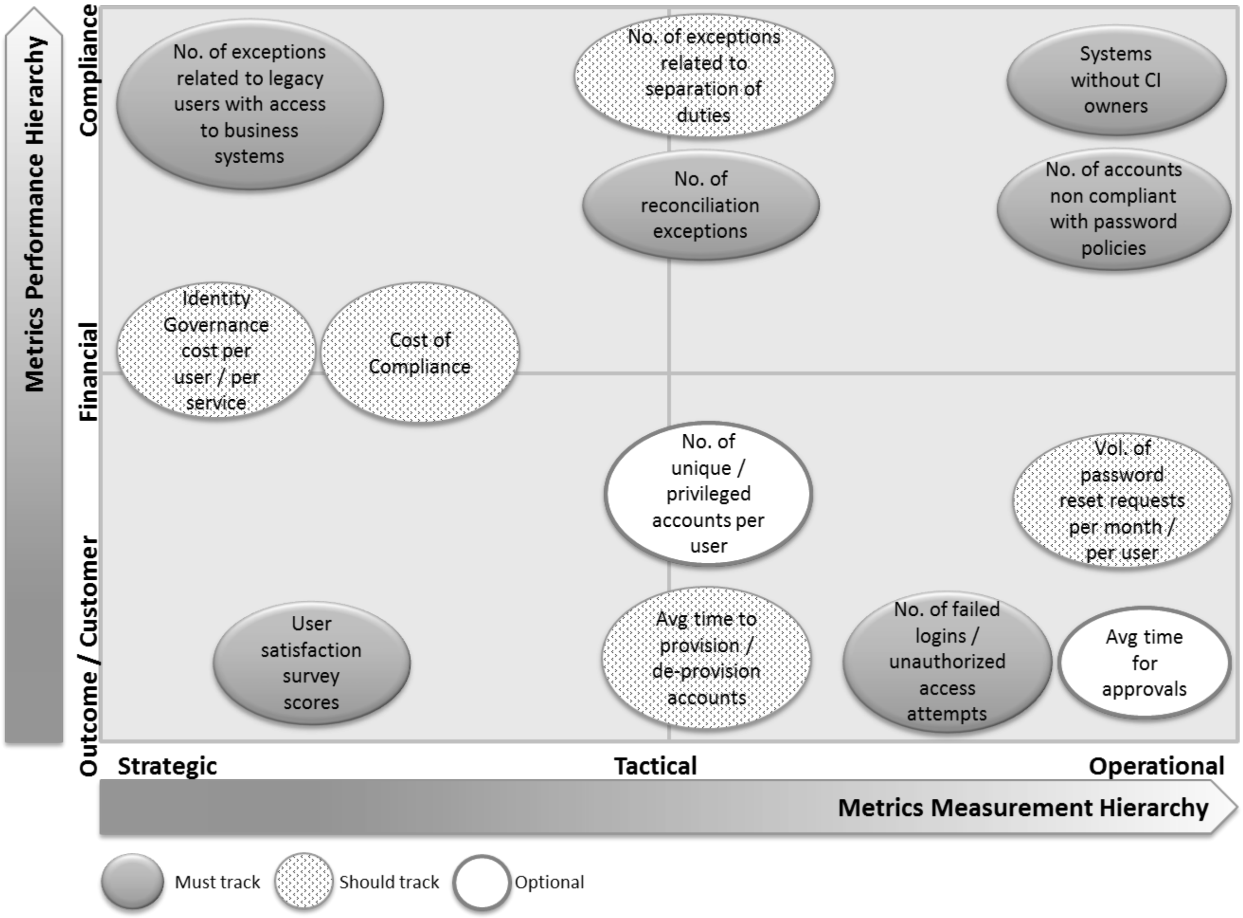

Metrics Performance Hierarchy
Compliance
Financial
Outcome / Customer
No. of exceptions related to legacy users with access to business systems
No. of exceptions related to separation of duties
Systems without CI owners
No. of reconciliation exceptions
No. of accounts non compliant with password policies
Identity Governance cost per user / per service
Cost of Compliance
No. of unique / privileged accounts per user
Vol. of password reset requests per month / per user
User satisfaction survey scores
Avg time to provision / de-provision accounts
No. of failed logins / unauthorized access attempts
Avg time for approvals
Strategic
Tactical
Operational
Metrics Measurement Hierarchy
Must track
Should track
Optional

**Figure 8: Performance Metrics**

Identity intelligence should comprise of analytics on identity related status and trends. Some of the reports that could be generated for analysis are:

- Ambiguities in entitlements, user profiles, accounts and roles mapped to licensing
- Privileged user access and zombie accounts (accounts that remain active after user has switched roles or left the organization)
- DLP monitoring logs on sensitive transactions (financial statements, memos) and ambiguities in identities and entitlement level authentication
- Historical access information on transactions / events monitored
- Threats such as DDoS and APT can be monitored and reported upon and analyzed to enhance security capabilities
- Monitoring logs of privileged user accounts, as they would be vital for forensic analysis
- Access attempts from hackactive regions monitored using geotagging

# 5 Summary

Through the earlier sections we have established why IAM is no longer a debatable option but a definite need for organizations wanting to align themselves better through agile and flexible IT models, save costs, improve performance against compliance standards, secure their information and stay ahead of competitors. An effective IAM strategy coupled with innovative and best in class technologies with the above mentioned capabilities can be a true enabler of business process innovation and help in recognizing the maximum value of investments. In a competitive business environment where business services are increasingly being delivered over collaborative and social platforms, IAM can prove to be a key enabler of differentiation in enterprise risk management, compliance initiatives, customer alignment, relationship management, reduced TCO, increased productivity and improved security.

# 6 Glossary

BYOD - Bring Your Own Device

BYOA - Bring Your Own Application

IAM - Identity and Access Management

CIO - Chief Information Officer

ROI - Return On Investment

SaaS - Software as a Service

LOB - Line of Businesses

SoD - Segregation of Duties

PII - Personally Identifiable Information

SSO - Single Sign On

DLP - Data Loss Prevention

DDoS - Distributed Denial of Service

APT - Advanced Persistent Threats

TCO - Total cost of ownership

IT – Information Technology

IP - intellectual property

# 7 References

Thanjavur Ravindran, Srikanth. 2013. “Solving The Identity and Access Management Conundrum.” ISACA Journal Vol.5, 2013

Kelly M. Kavanagh, Oliver Rochford. “Gartner Magic Quadrant for Security Information and Event Management.” 25 June 2014. ID: G00261641

Maler, Eve. Andras Cser 2012. “The Forrester Wave(TM): Enterprise Cloud Identity and Access Management, Q3 2012", Forrester Research, Inc., July 19 2012

Kuppinger, Martin. 2013. “Kuppinger-Cole Leadership Compass”. Access Governance, February 2013: 70735

# The Path Not Taken: Maximizing the ROI of Increased Decision Coverage

**Laura Bright**

Laura_bright@mcafee.com

## Abstract

Measuring code coverage is a popular way to ensure that software is being adequately tested and gives software teams confidence in the quality of their product. In particular, decision coverage of both automated and manual tests is a good way to ensure that as many paths as possible are covered in testing and the team is not missing possible defects. Management likes to report high decision coverage numbers as an indicator of product quality, and many QA teams set a goal of reaching a target percentage as a way of demonstrating that the code has been thoroughly tested.

But what does high decision coverage mean in terms of overall product quality? Clearly it is good to cover as many decisions as reasonably possible. A good set of functional test cases will cover the "low-hanging fruit", but may leave many uncovered decisions. Some of these uncovered decisions may be difficult or impossible to cover under normal testing conditions, but many can be covered by adding automated functional tests, unit tests, and manual tests. Hard to reach "corner cases" can be covered by setting up special testing environments. Some of these new tests may cover important use cases while others may provide little benefit. The question this paper aims to answer is, how do you determine where to invest your efforts to increase decision coverage?

This paper presents strategies to increase your decision coverage by targeting areas of the code with the highest ROI. Rather than focus on reaching a target percentage, we show how you can set realistic decision coverage goals that will improve your overall software quality. We present specific examples of how this approach has benefitted our team, including:

Identifying automated functional test cases that were overlooked by QA

- ○ Removing dead or obsolete code
- ○ Finding product defects that cause some code to not get executed
- ○ Identifying manual test cases to cover parts of code that are difficult to test through automation
- ○ Determining areas of the code that would benefit from increased unit testing

Through these efforts, our team has increased our percentage decision coverage, but more importantly, has improved overall quality by finding defects that otherwise may have been missed.

## Biography

*Laura Bright is a Software Development Engineer in Test (SDET) at McAfee, with a focus on automated testing. She has spent the past five years developing automation tools and automated test suites for McAfee products. More recently, she has served as a Quality Champion at McAfee and works across teams to share best practices to improve software quality. Previously she has presented "Catch Your Own Bugs: Including all Engineers in the Automation Cycle" at PNSQC 2012.*

# 1 Introduction

Measuring code coverage is a popular way to ensure that software is being adequately tested and gives software teams confidence in the quality of their product. Management likes to report high decision coverage numbers as an indicator of product quality, and many QA teams set a goal of reaching a target percentage as a way of demonstrating that the code has been thoroughly tested.

But what does high coverage mean in terms of overall product quality? Clearly it is good to cover as many code paths as reasonably possible, but how do you know when it is beneficial to add new test cases? A good set of functional test cases will cover the "low-hanging fruit", but may leave many uncovered decisions. Some of these uncovered decisions may be difficult or impossible to cover under normal testing conditions, but many can be covered by adding automated functional tests, unit tests, and manual tests. Hard to reach "corner cases" can be covered by setting up special testing environments. Some of these new tests may cover important use cases while others may provide little benefit.
If your coverage numbers are above your target, does that indicate high quality? If your numbers are below the target, is that a cause for concern? In some cases, the answer to both of these questions is “yes”, but focusing on just one number may oversimplify the problem and cause teams to miss the big picture.

This paper aims to reframe the question of when and how to increase code coverage in terms of product quality. Instead of asking “What should we do to improve our code coverage numbers?”, the question we ask is “What areas of the code need increased coverage to improve the quality of our software?” The goal is not just to increase coverage but to cover overlooked test cases and other reasonable scenarios. We present a case study showing our team’s experience analyzing our code coverage data to identify which areas of the code needed additional testing to meet this goal. We present examples of uncovered areas of the code that required additional tests, as well as areas where we chose not to write tests because these tests were unlikely to improve product quality. The goal is to use code coverage tools and data to improve your testing strategy, rather than focus on tasks with a low return on investment (ROI) just for the sake of increasing your numbers.

Section 2 presents a brief overview of types of code coverage and terminology used in this paper, and discusses related work. Section 3 provides an overview of the team and project structure and describes how we got started on our detailed code coverage analysis and targeted condition/decision coverage improvements. Section 4 presents specific examples encountered while reviewing the coverage data and discusses how to identify areas of the code that should be targeted for improved coverage. Section 5 presents results, and Section 6 concludes.

# 2 Background and Terminology

There are many different types of code coverage including line coverage, function coverage, branch or decision coverage, and condition coverage. See reference [Bullseye] for more information.

- **Line coverage** – percentage of lines of code executed
- **Function coverage** – percentage of functions executed.
- **Decision coverage** – percentage of all branches executed in conditional statements (for example, for a single “if” statement, have tests been executed where the statement evaluates to both true and false)
- **Condition coverage** – percentage of conditions that have evaluated to both true and false. This is different from decision coverage because a single “if” statement may consist of multiple conditions.
- **Condition/decision coverage** – This is the union of the condition coverage and decision coverage. It has the advantage of simplicity without the individual limitations of each of these two metrics [Bullseye].

To collect our code coverage data we used the Bullseye code coverage tool [Bullseye] which reports two of these metrics: function coverage and condition/decision coverage. The condition/decision coverage metric in Bullseye reports the percentage of all possible decisions and conditions that have been executed. Note that Bullseye does not report the condition coverage or decision coverage separately so we do not provide the breakdown of these two metrics in our results. We focus on condition/decision coverage rather than function coverage since the condition/decision coverage is a better measure of how well we are testing all possible paths in the code. Condition/decision coverage is also more difficult to increase than function coverage, since there may be many possible branches and conditions that can be executed in a single function. In the remainder of the paper we use the term coverage to refer to the Bullseye condition/decision coverage metric unless otherwise noted.

In the work presented in this paper, we focus on automated functional test coverage since we found these tests to be the most effective way to cover the uncovered code for our project. However, we do include unit test and manual test coverage examples where appropriate, as these types of tests are also valuable for increasing code coverage and improving product quality. We note that these results are partly due to the nature of our project. The project in this paper is complex anti-malware software that integrates many components. While it is possible to write unit tests for individual components, we found that in many cases adding functional tests had a higher ROI. Unit testing is still an important part of our project, and we recognize that it can be a very useful way to increase code coverage of many software projects.

### 2.1 Previous work

The topic of code coverage has been studied extensively in the literature. [Marick91] provides a good study of achieving different code coverage goals through unit testing, but argues that it is better to achieve as much coverage as possible through black box tests and fill in the gaps with unit tests. The author also notes that some branches are infeasible and discusses the concept of weak mutation coverage, branches that are unlikely to reveal defects. While the specific language, tools, and coverage metrics differ from the examples presented in this paper, the author's arguments are consistent with the main takeaways of the work we present. [Marick99] notes that code coverage metrics should not be a minimum shipping requirement, but instead should be used as a guide to indicate areas where testing can be improved. The author points out that imposing a minimum requirement may encourage testers to add tests that cover the easiest conditions until they reach the goal, thereby potentially missing more important test cases. Imposing a minimum requirement may also encourage testers to add many low ROI test cases [Marick99]. [Manu2010] presents a case study of a team's efforts to increase their block coverage from 91% to 100%, starting with "interesting" blocks such as those that cover key customer scenarios.

## 3 Getting started

This section provides a brief background on our team and project followed by an overview of our efforts to increase code coverage.

### 3.1 Project overview

We present examples and data gathered while testing the McAfee Endpoint Security suite which consists of multiple components designed to protect systems for small and medium businesses. The project follows a scrum model and consists of multiple scrum teams, each working on a distinct component of the Endpoint Security suite. At the end of each sprint, all teams report condition/decision coverage for both automated functional tests and unit tests as part of a larger set of quality metrics. These metrics are reviewed regularly by all stakeholders to help evaluate overall project quality.

The specific project we focus on in this paper is the Threat Prevention module which includes Anti-Malware protection and memory protection. This module includes both on-access and on-demand scanners to scan for viruses as well as buffer overflow protection and access protection. The code is developed in C++. In most cases, automated functional tests are written by QA engineers and unit tests are written by developers, although there has been increasing overlap in the developer/QA roles.

During feature development the size of the underlying code base and the percentage of conditions and decisions covered fluctuates, but most teams typically reported condition/decision coverage numbers in the range of 30-40%. This range was considered acceptable but not ideal, and a goal for all teams was to raise this number to 50% or greater over the course of the project.

As part of an effort to understand our team's numbers and identify areas for improvement, we held team meetings to review uncovered conditions and decisions in the code and identify which conditions and decisions needed to be covered through additional tests. Since reviewing thousands of lines of code is a huge task, we focused on one component per meeting and limited meeting time to one hour. We started with the largest and most complex components since these were among the most commonly used and were likely to have a higher ROI.

All of the developers and QA engineers on the team were expected to attend, and everyone brought valuable expertise to the meeting. Developers provided insight into when uncovered decisions or functions were expected to be covered to help QA engineers write appropriate test cases. In some cases they determined that the uncovered function or decision was obsolete and could be safely removed from the code. QA engineers determined what additional test cases were required to cover these areas. In some cases QA determined that test cases had been inadvertently overlooked or omitted from the automation suite and needed to be added to weekly automation runs.

To keep the meeting within the one-hour time limit, whenever we found an uncovered function or code block that required additional testing or investigation, the meeting moderator would make a note of the code segment and action required and assign follow-up tasks after the meeting. Sample follow up tasks included:

1. Determine if function X is still called anywhere in the code
2. Write test cases and automation scripts to cover condition Y
3. Remove a block of obsolete code

In the next section we provide some more in-depth examples of the types of uncovered decisions found in these meetings. We present both examples of code branches that we added additional tests to cover, as well as branches where we determined that covering them would do little to improve overall product quality.

# 4 Examples

In this section we present pseudocode examples that are representative of the types of uncovered code we found in our review sessions. In general, the uncovered conditions and decisions that have the highest ROI to add coverage for are the ones that are expected to get covered in practice, are straightforward to test either manually or through an automated functional or unit test, and that could potentially expose a previously undetected defect. Note that this does not mean we expect to find defects if we cover the condition, but rather that if there is a defect it would be undetected if we do not test the condition. In the code segments we use the following notation for the decisions, similar to what is provided by the Bullseye tool.

➔ Decision or function is not covered at all<br>
➔ T – path only taken when decision evaluates to true<br>
➔ F – path only taken when decision evaluates to false<br>
➔ t – the individual condition within the decision statement has evaluated to true but not to false<br>
➔ f – the individual condition in the decision statement has evaluated to false but not true<br>
TF – Both true and false decisions have been evaluated<br>
tf – the individual condition has evaluated to both true and false

## 4.1 Uncovered functions

The first example we found in our review were uncovered functions, i.e., functions that were never executed during the test run. Since high function coverage is generally easier to achieve than high condition/decision coverage, these examples offer some "low-hanging fruit", and determining how to handle these functions is a good first step towards improving code coverage.

We found uncovered functions that fell into several categories:

1. **Additional test cases needed to cover the function**. Some of the uncovered functions were types of functions developers could provide guidance for what product functionality needed to be tested to cover the function. In these cases it is usually straightforward to add manual or automated tests for product features that were not commonly used and may have been overlooked by QA.

2. **Wrapper functions**. Some libraries contained multiple APIs for the same functionality, for example wrapper functions that set default values for optional parameters. Generally covering these uncovered APIs was considered lower priority as long as the underlying functionality was being adequately tested, although if the interface was confirmed to be obsolete it was removed.

3. **Dead code**. Some uncovered functions were obsolete APIs or functionality that had been removed from the product. Once developers confirmed that these functions were not ever expected to be executed, they were removed from the code base, which helped increase overall code coverage numbers and more importantly makes the code easier to understand and maintain.

4. **Functions that were expected to be covered**. In some cases, a function was not covered even though we believed it should have been covered by an existing test case. Such cases could either indicate a product defect, or at the very least further investigation to understand why they were not covered and what needed to be changed to cover them.

## 4.2 Dead code

In addition to unused functions, our review sessions identified code that was unreachable in practice. While much of this code was technically reachable through unit tests, it would never be covered by an end user. For example, some parts of the code included features that were not scoped for the current release, either because they were obsolete features dropped from the product or they were "placeholders" for features that were planned for a future release. In both cases, there was little benefit to testing this code for the current release since it covered features that were not supported. We estimate that at least 1-2% of the code in some components was unreachable. See Section 5 for details.

Obsolete code blocks can be safely deleted from the code, while code that is intended for a future release can be commented out. As we show in Section 5, removing and/or commenting out this code significantly improved our team's decision coverage metrics and ensured that our coverage numbers were meaningful when measuring overall quality. Further, it cleaned up the code and allowed the team to focus on testing supported code that would be covered by an end user.

## 4.3 Boundary Tests

Many of the uncovered decisions we encountered were boundary cases that were verifying that variable values were within the acceptable range. Consider the following example in Figure 1:

```
➔ T  if
➔ t (x >= MIN_VALUE &&
➔ t                x <= MAX_VALUE) {
          //do something
   }
➔ else {
          // handle the invalid value
   }
```

*Figure 1: Boundary Test Example*

In Figure 1 the two conditions x >= MIN_VALUE and x <= MAX_VALUE have both evaluated to true but not false, so we have not tested handling invalid values. Such test cases are usually straightforward to automate through black box tests if the values are configurable by the end user, or through unit tests otherwise. Adding these test cases increases decision coverage and verifies that invalid values or errors are properly handled.

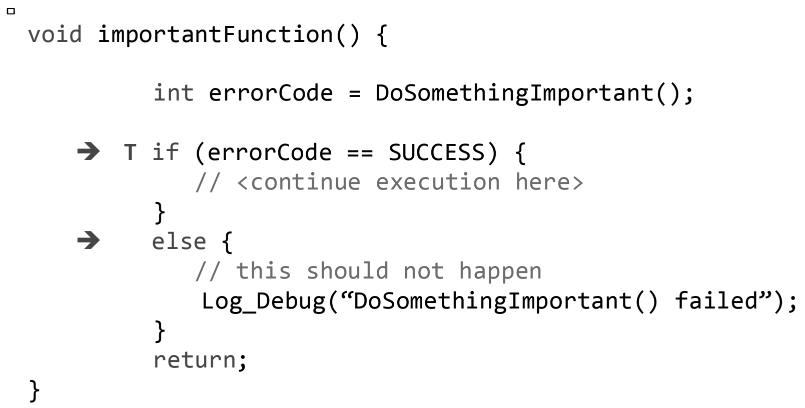
## 4.4 Manual tests

Our review also identified parts of the code that should be covered manually, and gave us the opportunity to evaluate the coverage of our manual test suite. For example, our product needs to detect viruses and malware on removable drives, and provides the option to pause tasks on laptops while running in battery mode. Such test cases cannot be easily automated. Since these test cases are not part of the automation suite, they are not executed as often and may be more prone to defects. Reviewing our coverage data allowed us to easily identify features and scenarios that are best covered by manual tests and ensure that the manual test suite is adequately covering the product.

## 4.5 Error handling

In our experience reviewing uncovered conditions/decisions, there were many error and null pointer checks. Some of these checks may be valid error conditions that could occur in practice, but others are checking for errors that will only occur if there is a bug in the code. In this section we discuss the difference between these two cases. In the former case, adding a test case to cover an uncovered error check may be beneficial, but in the latter case the code will only be covered if a bug exists and there is less benefit to explicitly trying to cover it.

It is good coding style to verify that a pointer is not null before dereferencing, and to verify that a function call succeeded before continuing. All of these checks are important to include in the code, especially during development and testing phases where they may help catch critical defects. However, once the code has been debugged and everything is working as expected, most of these conditions should not get covered in practice. How do you to determine when it is beneficial to write a test case to cover an error condition? Consider the following two examples in Figure 2 and Figure 3:

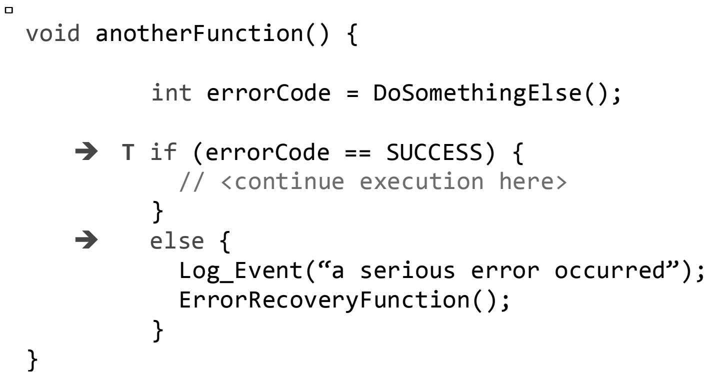


```
void importantFunction() {

         int errorCode = DoSomethingImportant();

    ➔  T if (errorCode == SUCCESS) {
            // <continue execution here>
         }
    ➔    else {
            // this should not happen
            Log_Debug(“DoSomethingImportant() failed”);
         }
         return;
}
```

*Figure 2: Handling an error that is not expected to occur in practice*

It is considered good coding practice to verify error codes returned by function calls, and many static analysis tools will indicate a potential defect if the return value of a function is not checked.

In the example in Figure 2, the function DoSomethingImportant is expected to return success, but the caller of the function validates the returned error code as a formality and to make the code clean and readable. If the call to DoSomethingImportant fails it indicates a bug in that function that needs to be fixed, but there is less benefit to explicitly writing a unit test to cover the uncovered condition. (We note that there is benefit to verifying that the debug log is working correctly, but this particular example covers a failure that is not expected to occur in practice). Adding a specific test case to cover this decision would improve the coverage metric but would be unlikely to help find new defects. Note that all of the decisions within the DoSomethingImportant function should still be covered adequately to ensure that this function is working as expected.



```
void anotherFunction() {

         int errorCode = DoSomethingElse();

    ➔  T if (errorCode == SUCCESS) {
           // <continue execution here>
         }
    ➔    else {
           Log_Event(“a serious error occurred”);
           ErrorRecoveryFunction();
         }
}
```

*Figure 3: Handling an error that may occur in practice*

In the second example in Figure 3, there is explicit error handling code in ErrorRecoveryFunction() to handle a rare catastrophic failure that could potentially occur in practice. The error check is not just a formality for good coding style, but is actually testing a valid error condition. In this case, there would be high value in adding a test case to make sure ErrorRecoveryFunction() is working.

## 4.6 Null Pointer Checks

Another class of error handling decisions where there is often little benefit to covering both True and False is null pointer checks.

It is good coding practice to verify that a pointer is not null before dereferencing it, and doing so improves the overall quality of the code and makes the product more robust. However, as in the case of error handling, there is often little value in explicitly adding a test case to cover both the True and False evaluations of these conditions unless they are failures that might be expected occur in practice. Consider the following examples:

```
          int * integer_array;
          integer_array = new int[MAX_ARRAY_SIZE];

    ➔  T if (integer_array) {
             // <continue execution here>
          }


          Object * myTestObject = new Object();

    ➔  T if (myTestObject) {
               // <continue execution here>
          }
```

*Figure 4: Verifying memory allocation*

In the examples in Figure 4, the memory allocation is expected to succeed in almost all cases unless the system is out of memory. While testing a system with insufficient memory is an important test scenario, explicitly covering every null pointer check in the code is less likely to have a high ROI. A related scenario is a “clean-up” function that deletes an object or array if it exists (Figure 5):

```
void cleanUp (Object * obj) {

    ➔  T if (obj) {
               delete obj;
          }
}
```

*Figure 5: Verifying an object exists before deletion*

As in the previous examples in Figure 4, the null pointer check in Figure 5 is included to make the code more robust and correctly handle cases where cleanup is called before initialization of the object. However, the case where the object is null may not be expected to occur in practice and explicitly testing this condition is a low priority.

Many functions return a reference to an object, and similar rules apply in cases where this reference is not normally null. If a function returns a reference to an array or object and the value is never expected to be null unless there is a catastrophic system error, there is little value in adding a test case to cover the null pointer condition.

We note that the majority of the uncovered conditions we encountered during our review meetings were either return code checks evaluating to error, or pointer checks evaluating to null. This is one reason we determined that trying to achieve near 100% decision coverage would have low ROI for our project.

# 5 Results and Takeaways

This section presents some data from the detailed coverage review meetings. As mentioned in Section 3, we reviewed one component at a time to make the workload manageable, and also to allow us to focus on the most critical parts of the product first. This table presents some key metrics from these review meetings:

| | Percent condition/ decision coverage before review | Percent condition/ decision coverage after review | Test cases added | Unreachable conditions/ decisions removed | Defects filed | Additional conditions/ decisions covered |
|---|---|---|---|---|---|---|
| Component 1 | 46 | 53 | 25 | 121 | 2 | 114 |
| Component 2 | 48 | 50 | 20 | ~20 | 2 | ~25 |

*Table 1: Results*

For component 1, we found approximately 25 new functional test cases to automate and add to our nightly automation suite. Most of these test cases were boundary conditions or uncovered functions for less commonly used product features. We also noted a handful of manual test cases that needed to be executed. During this review we also identified large blocks of code for features that were no longer supported in the product as well as for a feature that was planned for a future release but is not supported in the current release. These blocks included 121 uncovered conditions/decisions. This code was commented out to allow us to focus on higher-priority decisions that were still uncovered.
After removing the uncovered decisions and adding the 25 new test cases to our test suite, coverage for Component 1 increased from 46% to 53%. Most of this increase was due to the dead code removal, but the 25 functional test cases contributed to the increase and also uncovered at least 2 product defects. While 53% is a lower number than many organization aim to achieve, our team determined that most of the remaining uncovered decisions were error checks and null pointer checks that had a low ROI. After this review our team had greater confidence that Component 1 had been adequately tested, and we had increased the coverage above 50% which was considered a realistic and achievable goal for our project.

We later conducted a similar review for Component 2. Component 2 had fewer obsolete/placeholder blocks, but we still identified several functions that were no longer needed. We also identified 20 valid test cases that covered important product functionality and found 2 new defects. We note that during the period this review took place, the code for this component underwent a refactor to reduce unnecessary complexity, which makes it difficult to report the exact numbers for removal of unreachable code and additional conditions/decisions covered.

After the refactoring was completed, our overall condition/decision coverage of the component increased from 50% to 56%. While the refactoring was done independently of the team’s code coverage efforts, an important benefit of this change was eliminating unnecessary conditions and decisions from the code base and helping the team more easily identify the highest priority uncovered conditions/decisions going forward. We plan to continue these efforts across all components of the project.

# 6 Conclusions

Code coverage is an important quality metric that has received considerable attention in the software community. While every software project aims to achieve high code coverage, often less attention is given to how high coverage translates to high quality and how software teams can best use their available resources to improve coverage.

The results presented in this paper present some key findings for our team and product, but your mileage may vary. For example, your code base may have fewer error checks and null pointer checks than the code presented in these examples, so your team may realistically expect to achieve coverage much greater than 50%. Similarly, you may find that most of the null pointer checks or error checks in your code are valid error conditions that need to be covered. The examples presented in this paper should not be used as hard rules, rather they should serve as examples of how you can interpret your own code coverage data to identify the test cases that have the highest ROI for your project. While this paper focused on examples using C++ code and the Bullseye code coverage tool, similar principles should apply to other programming languages and tools.

When reviewing uncovered conditions/decisions in your code coverage data, you should consider the following questions:

- Is this uncovered condition a valid test case?
- Is it straightforward to cover this condition by a manual test, or by an automated functional or unit test?
- Is there a good chance this condition will be covered in practice by an end user?
- Is it a high risk if we don't test this condition?

If you answer yes to all these questions, there is a high ROI to add a test to cover the condition. If your test suite has covered all the conditions and decisions that are likely to occur in practice and the team agrees there is a low ROI to covering the remaining uncovered paths, you can state with greater confidence that you have met your code coverage goals.

# References

[Bullseye] Bullseye Testing Technology "Code Coverage Analysis." http://www.bullseye.com/coverage.html (accessed July 9, 2014).

[Marick91] Marick, Brian. "Experience with the Cost of Different Coverage Goals for Testing" *Pacific Northwest Software Quality Conference, 1991. Available at http://www.exampler.com/testing-com/writings/experience.pdf*

[Marick99] Marick, Brian. "How to Misuse Code Coverage" *Testing Computer Software, 1999. Available at http://www.exampler.com/testing-com/writings/coverage.pdf*

[Manu2010] Manu, C., Amalo, D., Nagpal, P., Tan, R. "The Last 9% Uncovered Blocks of Code – A Case Study" *Pacific Northwest Software Quality Conference, 2010.*

# Reducing the Cost of User Acceptance Testing with Combinatorial Test Design

**Steven Dyson**

steven.dyson@cambiahealth.com

## Abstract

At Cambia Health Solutions, User Acceptance Testing (UAT) is part of our typical testing strategy to ensure that software solutions fit with the workflow and manual processes for efficient business operations. Our UAT is often a manual process making it expensive, time consuming, and heavily reliant on shared business resources. Long UAT cycles have also led to delayed deployment schedules, which left development teams dissatisfied and held up their work efforts.

Two years ago, Cambia began utilizing Combinatorial Test Design (CTD) to analyze the UAT test cases on some of our larger projects. By analyzing our software parameters and values we were able to consistently optimize our test suites by eliminating duplicate test interactions while maintaining test coverage and realize a reduction in the cost and duration of our UAT cycle. This optimization also improved our predictability during the agile software development life cycle.

This paper details how to incorporate the use of CTD into a testing strategy, how to analyze and parameterize UAT test cases, how to setup and use the CTD application Hexawise, and provides an overview of the impact and results CTD has had at Cambia. Our typical optimization resulted in at least a 30% reduction in required testing time, with as much as a 92% reduction in one test suite.

## Biography

*Steve Dyson spent 4 years working in User Acceptance, Integration and System testing at Cambia Health Solution before moving into Software Quality Assurance 2 years ago. Recently, Steve has been building an SQA training program at Cambia to standardized SQA, testing and agile concepts and practices within the company.*

*Steve has a business degree in Accounting from the University of Utah.*

# 1. Combinatorial Test Design (CTD)

## 1.1. What is Combinatorial Test Design?

Software is always evolving and growing in complexity. Research conducted by the National Institute of Standards and Technology (NIST) suggest that many software failures are caused by the interaction of a relatively small number of variables (NIST, "Combinatorial and Pairwise Testing"). Since large applications could have dozens of different parameters that interact amongst each other, the number of unique interactions could be in the thousands (if not millions). Running thousands of tests is costly and time consuming, and Combinatorial Test Design can help solve that problem.

Combinatorial Test Design, or Pairwise Testing, is a test modeling approach that takes the parameters of a software application and combines the interactions between those parameters into an optimized set of test cases based on a defined level of interaction. All parameters and their corresponding values are combined with one another in a test set. The test set produced contains the full Cartesian product of parameter interactions and consists of the smallest number of possible test cases, while ensuring 100% test coverage based on the defined parameters and values.

## 1.2. Benefits and Uses of CTD

CTD is a very useful process when testers need to:

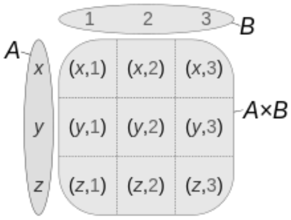
- Create a large test suite from scratch without having to manually write out each test case
- Identify testing gaps in existing test suites
- Eliminate duplications in existing test suites

Test planning can be time consuming, so using the CTD process not only saves us time in our test execution, it also saves time by eliminating the need to manually write extensive test suites.

## 1.3. How CTD Works

Once the parameters and values for the application have been gathered, it is best to organize them in simple tables. When the parameters and values are entered into a CTD tool, the Cartesian product is created. A Cartesian product is a mathematical operation which returns a single set from multiple sets (Cartesian product, Wikipedia).

**Figure 1, Cartesian Product**



| | 1 | 2 | 3 |
|---|---|---|---|
| x | (x,1) | (x,2) | (x,3) |
| y | (y,1) | (y,2) | (y,3) |
| z | (z,1) | (z,2) | (z,3) |

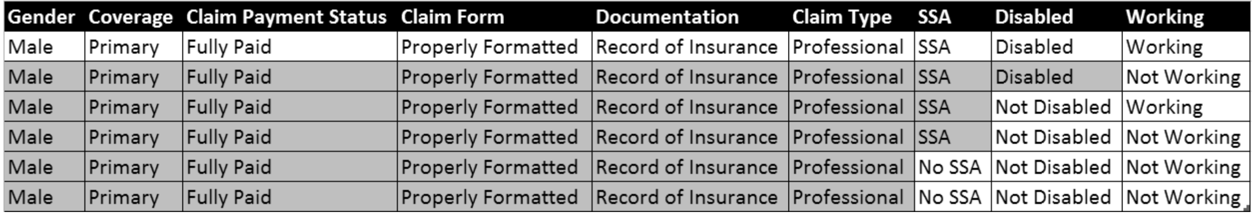
This simple product of set A multiplied by set B, gives us a new set called AB. Set AB contains every interaction between the values in Set A and Set B, and every interaction is present in our new set. Test cases can be structured in the same manner – a value of one parameter being tested with the values of other parameters.

Of course, software applications are not this simple and can be comprised of dozens of parameters. As we add more parameters, and those parameters contain more than just three values, the Cartesian product grows exponentially.

The CTD tool takes that Cartesian product of every possible test interaction, and combines and condenses those interactions into as few test cases as possible. Duplicated interactions are eliminated, and every interaction is present at least once in the new test suite.

In figure 2, as the Cartesian product for this example is written, notice how many values are duplicated throughout the first six test cases. CTD helps eliminate the redundancy in a test suite, by forming efficiently combined test cases.

**Figure 2**



| Gender | Coverage | Claim Payment Status | Claim Form | Documentation | Claim Type | SSA | Disabled | Working |
|---|---|---|---|---|---|---|---|---|
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | SSA | Disabled | Working |
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | SSA | Disabled | Not Working |
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | SSA | Not Disabled | Working |
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | SSA | Not Disabled | Not Working |
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | No SSA | Not Disabled | Not Working |
| Male | Primary | Fully Paid | Properly Formatted | Record of Insurance | Professional | No SSA | Not Disabled | Not Working |

## 1.4. Hexawise

Hexawise is a web-based test design tool specializing in CTD. It allows the user to add or import their test attributes, add special test requirements, create combinatorial tests, and add custom test scripts (Hexawise). This tool will be referenced throughout this paper.

## 1.5. CTD Example

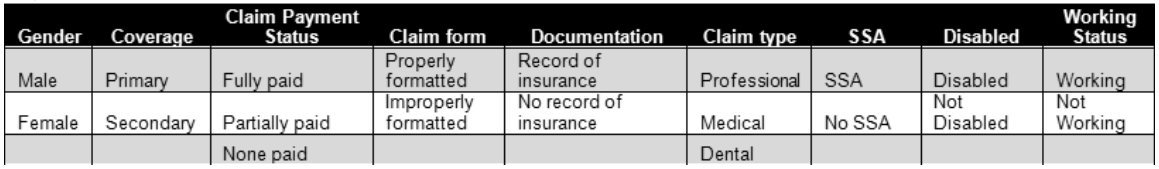
This example is a simplified version of test requirements for Medicare insurance claims in a claims processing system. We have identified the parameters that need to be tested, along with their corresponding values.

**Parameters**

- Gender of the patient
- Whether Medicare is the patient's primary or secondary insurance policy
- The status on payment of the claim
- Whether the Claim form was submitted properly or not
- If we have prior documentation of the patient's insurance history
- The type of claim being processed
- Whether the patient has Social Security (SSA) or not
- Whether the patient is on Disability or not
- Whether the patient is working or not

For the sake of organization, and import capabilities into CTD tools, writing the parameters and values in this format is very important.

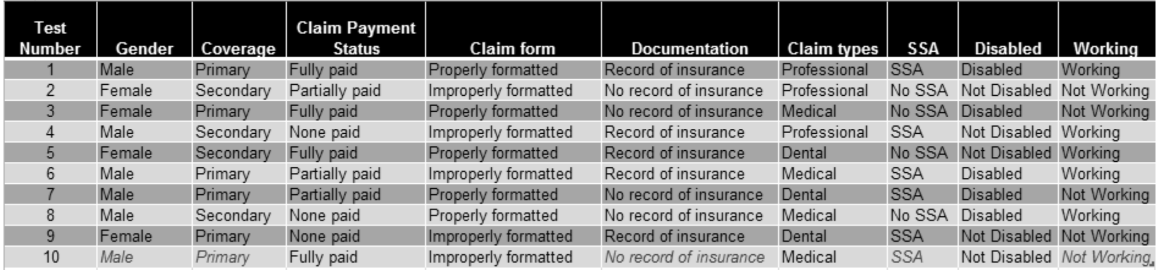
**Figure 3, Parameters and Value Format**



| Gender | Coverage | Claim Payment Status | Claim form | Documentation | Claim type | SSA | Disabled | Working Status |
|---|---|---|---|---|---|---|---|---|
| Male | Primary | Fully paid | Properly formatted | Record of insurance | Professional | SSA | Disabled | Working |
| Female | Secondary | Partially paid | Improperly formatted | No record of insurance | Medical | No SSA | Not Disabled | Not Working |
|  |  | None paid |  |  | Dental |  |  |  |

From this simple example, with each parameter having no more than three values, there are 1,152 possible interactions that can be tested – Male patient tested with Medicare as the primary coverage, Female patient tested with Medicare as the primary coverage, Male patient with a Claim Fully Paid, etc. The Cartesian product of possible interactions grows tremendously as we add more parameters and values to the test plan. Those 1,152 interactions can be combined because each interaction only needs to be tested once. After being run through a CTD tool, all 1,152 interactions can be tested with 10 total cases. Our small test suite provides 100% test coverage of the interaction between any two variables.

**Figure 4**



| Test Number | Gender | Coverage | Claim Payment Status | Claim form | Documentation | Claim types | SSA | Disabled | Working |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Male | Primary | Fully paid | Properly formatted | Record of insurance | Professional | SSA | Disabled | Working |
| 2 | Female | Secondary | Partially paid | Improperly formatted | No record of insurance | Professional | No SSA | Not Disabled | Not Working |
| 3 | Female | Primary | Fully paid | Properly formatted | No record of insurance | Medical | No SSA | Disabled | Not Working |
| 4 | Male | Secondary | None paid | Improperly formatted | Record of insurance | Professional | SSA | Not Disabled | Working |
| 5 | Female | Secondary | Fully paid | Properly formatted | Record of insurance | Dental | No SSA | Not Disabled | Working |
| 6 | Male | Primary | Partially paid | Improperly formatted | Record of insurance | Medical | SSA | Disabled | Working |
| 7 | Male | Primary | Partially paid | Properly formatted | No record of insurance | Dental | SSA | Disabled | Not Working |
| 8 | Male | Secondary | None paid | Properly formatted | No record of insurance | Medical | No SSA | Disabled | Working |
| 9 | Female | Primary | None paid | Improperly formatted | Record of insurance | Dental | SSA | Not Disabled | Not Working |
| 10 | Male | Primary | Fully paid | Improperly formatted | No record of insurance | Medical | SSA | Not Disabled | Not Working |

Every test scenario is unique, and the amount of duplication of parameter interactions is reduced as much as possible. All of the possible interactions have been combined together saving the tester from exhaustively writing every possible test scenario, which can be time consuming and difficult to accomplish.

# 2. CTD Process Steps

## 2.1. Discovery/Test Planning Phase

The discovery phase of the process is when we begin the analysis of the application we need to test. It doesn’t differ much from typical test planning, but the train of thought needs to be geared towards parameter identification, and how parameters interact with each other in a system. These activities may include:

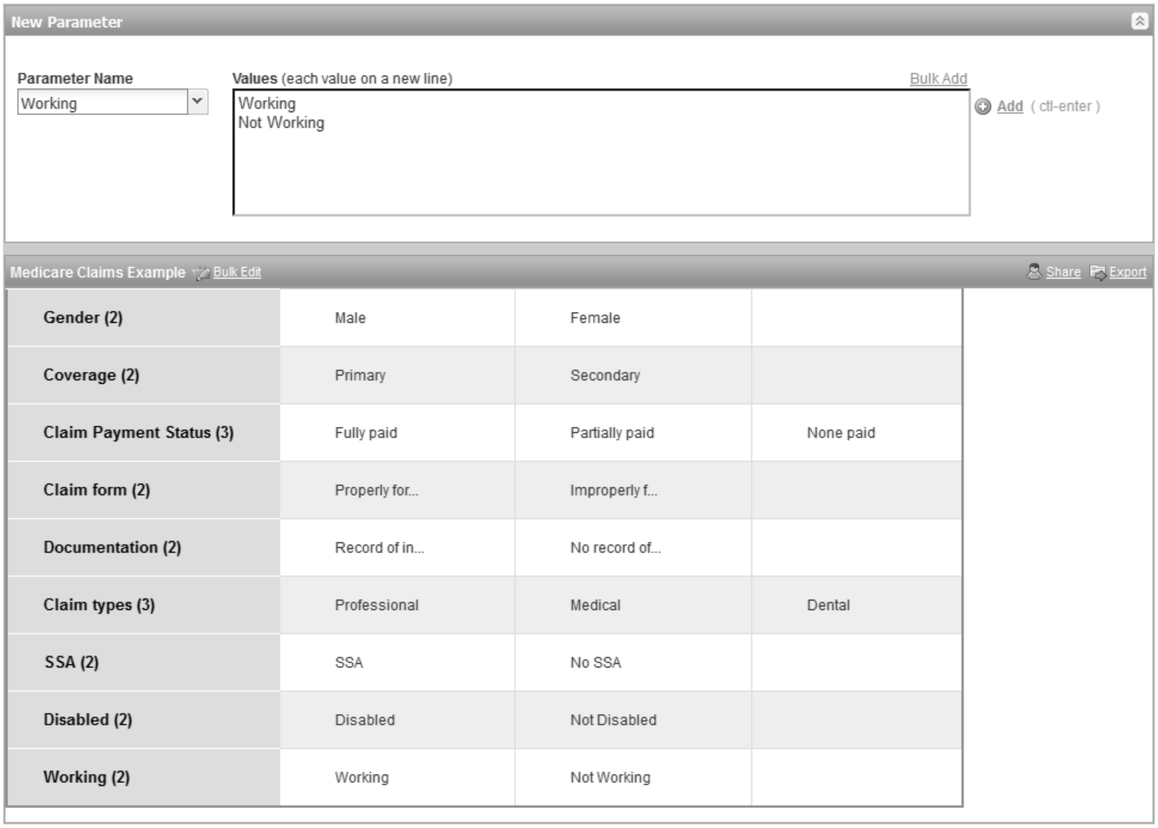
- Discuss with a Subject Matter Expert (SME)
- Review process and system documentation
- Review existing test cases and test suites
- Review historic defect reports

If we are leveraging a set of existing test cases, we start by reviewing the test cases and restructuring them in the parameter and value format. These existing test cases in this format can be imported into Hexawise quickly and efficiently.

If we are creating a new test suite from scratch, we start by reviewing all process documentation and requirements to ensure we are capturing all parameters in the application. Missing a parameter that needs to be tested can compromise the new test suite. This is why working with a SME is very important, because their experience and firsthand knowledge of the application may include details that aren’t present in documentation.

## 2.2. Entering Parameters Into Hexawise

Once we have identified the test scope in the application, we compile the test requirements into the parameter and value format. With Hexawise, these can be manually entered or imported from an Excel file.



New Parameter

Parameter Name: Working

Values (each value on a new line)

Working
Not Working

Medicare Claims Example

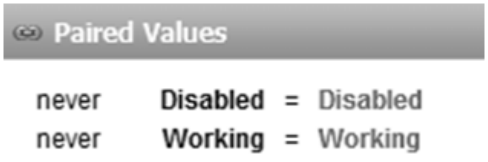
| | | | |
|---|---|---|---|
| Gender (2) | Male | Female | |
| Coverage (2) | Primary | Secondary | |
| Claim Payment Status (3) | Fully paid | Partially paid | None paid |
| Claim form (2) | Properly for... | Improperly f... | |
| Documentation (2) | Record of in... | No record of... | |
| Claim types (3) | Professional | Medical | Dental |
| SSA (2) | SSA | No SSA | |
| Disabled (2) | Disabled | Not Disabled | |
| Working (2) | Working | Not Working | |

## 2.3. Restrictions

There will often be interactions in a test plan that we have knowledge can't exist, or wouldn't exist in a test or production environment. We don't want to see these bad interactions muddying up our test plan, so we can restrict them. Restrictions are the interactions that should not be present in the test suite.

If it is known that the value of *Disabled* would never interact with a *Working* value, as seen in Figure 5, that combination can be marked as an Invalid Pair in Hexawise as seen in Figure 6.

**Figure 6**



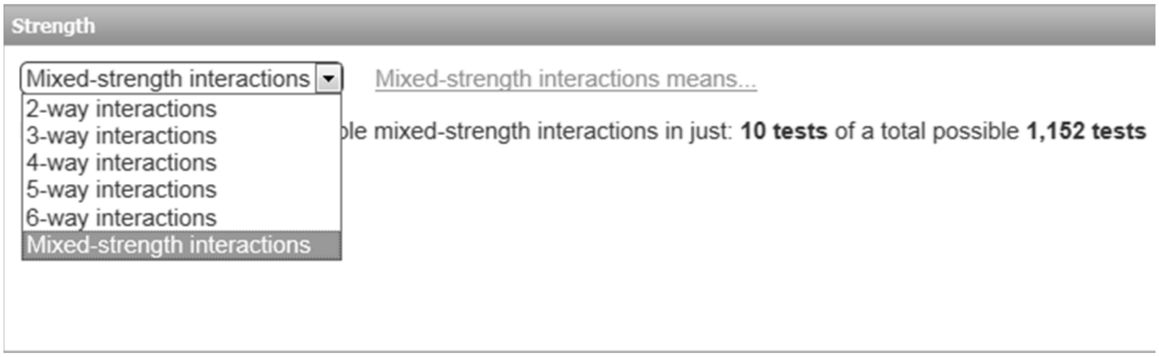
Paired Values

never **Disabled** = Disabled
never **Working** = Working

This interaction will then not show up in any of our test cases. We can add as many restrictions as necessary to ensure our test suite contains only good test cases.

## 2.4. Interaction Levels

Interaction levels in CTD and the Hexawise tool can be defined, based on whether we want value interactions to be matched in pairs of 2 (pairwise testing), sets of 3, etc. As Interactions levels increase the test suite will become more thorough, but the size of our test suite will also increase.



Strength

Mixed-strength interactions ▾ Mixed-strength interactions means...

ble mixed-strength interactions in just: **10 tests** of a total possible **1,152 tests**

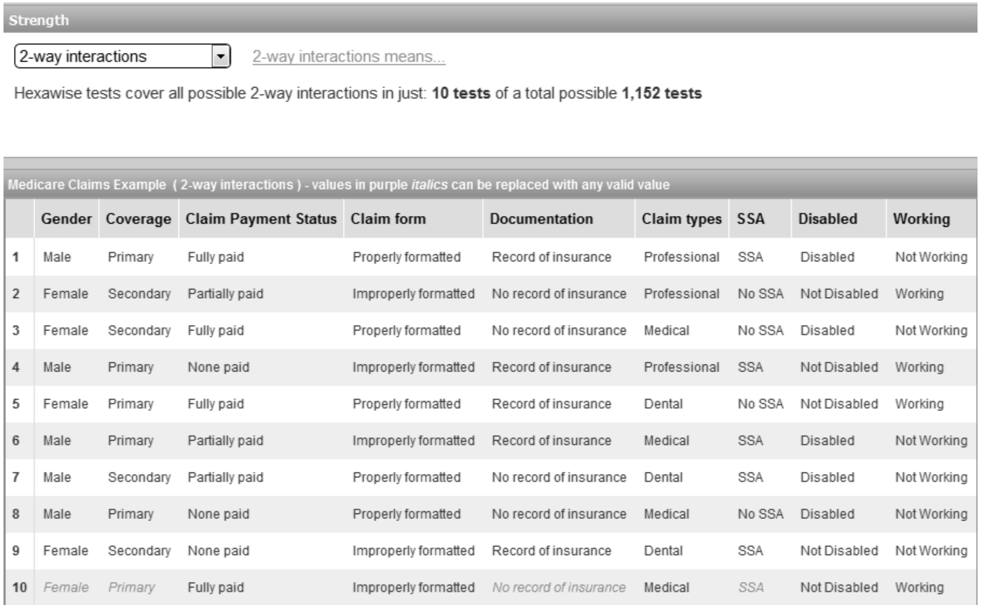
Pairwise testing is more than sufficient for the majority of software testing, but if we're testing software for the International Space Station, we may want to increase the interaction level. Pairwise testing ensures every pair of values is present in the test suite. Not every set of 3 values, 4values, etc. will be present, however.

If an application has stringent requirements, in which 3 or more attributes have specific interactions that need to be tested, it is recommended to increase the interaction strength. This will increase the number of test cases in the test suite to accommodate those requirements.

## 2.5. Executing Tests with Hexawise

Creating test cases in Hexawise is quick and simple. After we have defined our inputs and any restrictions for bad value pairs, we are ready to create tests. With the click of a button, Hexawise runs all of the value combinations together, excluding any restrictions we have defined, and generates the test suite. As mentioned previously, we can test all interactions with only 10 test cases. Notice that our restriction of the bad pair *Disabled* and *Working* does not appear in our test set.

**Figure 8**



Strength

2-way interactions ▾ 2-way interactions means...

Hexawise tests cover all possible 2-way interactions in just: **10 tests** of a total possible **1,152 tests**

Medicare Claims Example ( 2-way interactions ) - values in purple *italics* can be replaced with any valid value

| | Gender | Coverage | Claim Payment Status | Claim form | Documentation | Claim types | SSA | Disabled | Working |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Male | Primary | Fully paid | Properly formatted | Record of insurance | Professional | SSA | Disabled | Not Working |
| 2 | Female | Secondary | Partially paid | Improperly formatted | No record of insurance | Professional | No SSA | Not Disabled | Working |
| 3 | Female | Secondary | Fully paid | Properly formatted | No record of insurance | Medical | No SSA | Disabled | Not Working |
| 4 | Male | Primary | None paid | Improperly formatted | Record of insurance | Professional | SSA | Not Disabled | Working |
| 5 | Female | Primary | Fully paid | Properly formatted | Record of insurance | Dental | No SSA | Not Disabled | Working |
| 6 | Male | Primary | Partially paid | Improperly formatted | Record of insurance | Medical | SSA | Disabled | Not Working |
| 7 | Male | Secondary | Partially paid | Properly formatted | No record of insurance | Dental | SSA | Disabled | Not Working |
| 8 | Male | Primary | None paid | Properly formatted | No record of insurance | Medical | No SSA | Disabled | Not Working |
| 9 | Female | Secondary | None paid | Improperly formatted | Record of insurance | Dental | SSA | Not Disabled | Not Working |
| 10 | *Female* | *Primary* | Fully paid | Improperly formatted | *No record of insurance* | Medical | *SSA* | Not Disabled | Working |

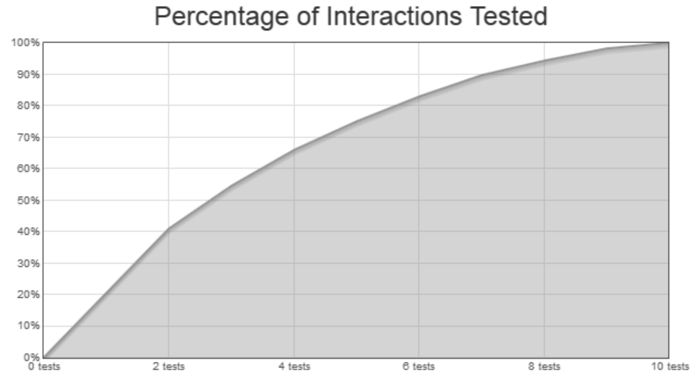
These tests can now be exported to Excel or HP Quality Center. Hexawise provides the user the ability to add auto-scripts if there is a need more specific detail in the test set.

### 2.6. Test Case Coverage Analysis with Hexawise

One note about a test suite created by CTD is that the numerical order of test cases is important. In our test example, we have test cases 1-10. As we progress through the order of test cases, the number of unique interactions decreases as Hexawise exhausts all possible combinations. This means that if testing time is limited and not all tests can be executed, it is important to test through the numerical order to ensure the highest test coverage possible.

As seen in Figure 9, the Percentage of Interactions Tested indicates that test case 1 and 2 represent 40% of all possible interactions. That is a significant portion of our testable values. As we progress through the test cases, the number of unique interactions tested decreases.

**Figure 9**



In the event limited testing time is available, this feature of Hexawise is valuable in that it aids the testers in defining an acceptable level of test coverage, and tells them how many tests need to be executed.

# 3. Cambia's User Acceptance Testing

In 2012, Cambia Health Solutions hired a consulting firm to help identify solutions for finding process efficiencies in our projects in preparation for the federally mandated upgrade to ICD-10 (which is the reclassification of all diagnosis codes used in medical billing). The firm suggested utilizing the CTD method in order to optimize our more complex UAT regression test suites. We began with a pilot to see if the process would work well for us. After adopting the process we optimized many of our existing test suites, and created brand new test suites for new applications.

### 3.1. Project 1 – CTD Pilot: Policy Benefit Validation

For our pilot run for CTD, a large regression suite was chosen. The regression suite covered insurance policy benefit validation, and was used to test every insurance policy we offer. When a new group signs up for our insurance products, they often have multiple policies. Our regression suite was designed to test a single policy, and the suite contained 667 test cases. On average, it took our user acceptance testers around 40 hours to complete the testing for a new group's entire set of policies. This was far too labor intensive, and we needed to find a way to optimize our test suite.

We met with a SME to identify all of the parameters, and their respective values that needed to be tested. We also identified any restrictions, or interactions between parameters that we did not want present in the test suite.

Seven different parameters were identified:

- Product
- Type of Service
- Provider Category
- Diagnosis

- Procedure
- Benefit Event

Each parameter had various values assigned. For Example, Provider Category contained three values: In Network, Participating, and Out of Network; while the Benefit Event parameter contained values such as deductible, coinsurance, copayment, and yearly visit maximum. The Type of Service parameter contained over 70 different benefits that needed to be tested, which was the driving force of the size of this test suite.

**Results**

We ran this test suite through the CTD tool and saw great results. We discovered that our initial test suite of 667 test cases was missing 23 test scenarios, which brought us to a total of 690 cases. The test suite that was produced from the CTD process reduced our total number of test cases to 230 – a 67% reduction in the test suite size and testing time. This meant that a tester could now test three groups in the same time it took them to test a single group.

The savings were tremendous, so we made the decision from our pilot to begin utilizing the CTD process for as many UAT test suites as possible.

## 3.2. Project 2 – Other Party Liability Test Suite Enhancement

CTD doesn't always result in a lower number of test cases, as we found in one of our project examples. I consulted the subject matter expert (SME) on a project team that tested the software which processed insurance claims with members who had a primary and secondary insurance in the event of an accident, also known as Other Party Liability (OPL). The other insurance was typically an auto insurance carrier. She had recently taken over as the user acceptance tester and was leveraging a preexisting test suite that wasn't very thorough.

We took the existing test suite containing 27 test cases, broke it down into the parameter format, and ran the inputs through the CTD tool.

Seven different parameters were identified for the new test suite:

- Group Type
- Provider Type
- OPL Condition
- Additional Claim Needed
- Accident Date
- Second Claim Accident Date
- Body Part Match
- Diagnosis

**Results**

We quickly found out that our test coverage was terrible, and there were several attributes that weren't being tested. We discovered that our current test suite only had about 30% test coverage, excluding the attributes that were missing.

Our newly expanded test suite now contained 112 test cases, but we had 100% test coverage based on the attributes defined by the new SME. After running the new test suite, we identified two high severity defects that had been in our production environment for months.

## 3.3. Employee Web Services Project

By far our best result with using CTD was on a web service testing project for an employee insurance website. The tester wanted to test the interaction of fields on web searches, to validate that error codes appeared due to certain combinations or if fields were left blank. The tester had basically mapped out a lot of the Cartesian product of all possible value interactions, so CTD was a perfect fit. There were a total of 228 test cases in the original test suite.

Eight different parameters were identified:

- From Date Search
- To Date Search
- Invoice ID
- Group ID
- Receipt ID
- Matching Invoice
- Matching Receipt
- Matching Group

**Results**

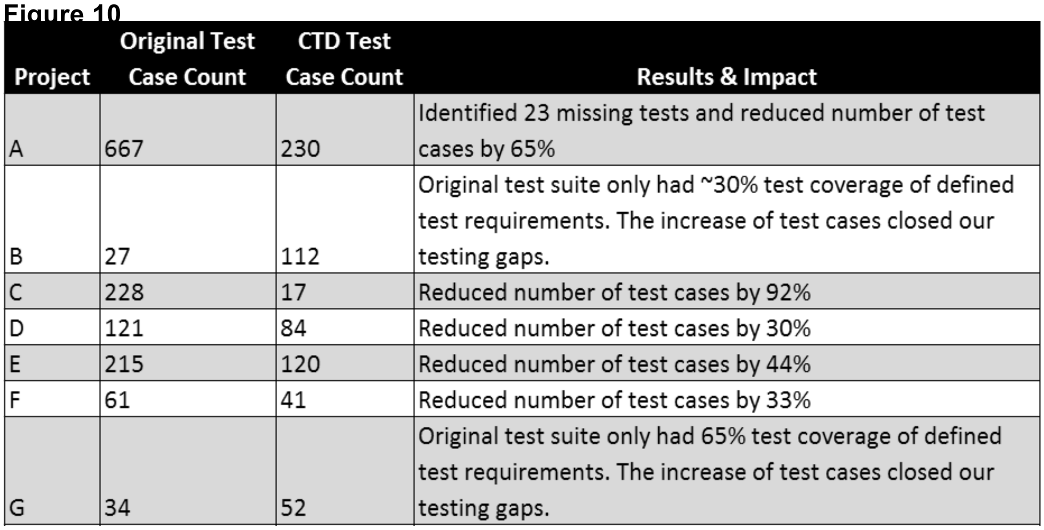
Since the original test suite was basically the Cartesian product, and CTD's algorithm condenses the Cartesian product, the reduction in test cases was remarkable. Our test suite created by CTD had 100% test coverage based on our defined attributes and values with only 17 test cases, which was a 92% reduction in the number of test cases that needed to be run. The duplication of test interactions in the original test suite contributed to the 228 test cases, so by combining those duplication interactions, we proved that the test suite required the tester to test the same things over and over again.

# 4. Analysis

Cambia has had a lot of success thus far with CTD and we are hoping to increase adoption within our organization on our various agile development teams. This test planning method didn't work for us in every situation and we spoke to many teams in which CTD just wasn't a good fit. Some of those reasons include a lack of variability in attributes, their test attributes needed to match database rows eliminating the value gained from test combinations, and applications that couldn't be broken into an attribute and value format.

Here are the results from seven projects in which we were able to utilize CTD at Cambia to improve existing UAT test suites (including those discussed in Section 3).

**Figure 10**



| Project | Original Test Case Count | CTD Test Case Count | Results & Impact |
|---|---|---|---|
| A | 667 | 230 | Identified 23 missing tests and reduced number of test cases by 65% |
| B | 27 | 112 | Original test suite only had ~30% test coverage of defined test requirements. The increase of test cases closed our testing gaps. |
| C | 228 | 17 | Reduced number of test cases by 92% |
| D | 121 | 84 | Reduced number of test cases by 30% |
| E | 215 | 120 | Reduced number of test cases by 44% |
| F | 61 | 41 | Reduced number of test cases by 33% |
| G | 34 | 52 | Original test suite only had 65% test coverage of defined test requirements. The increase of test cases closed our testing gaps. |

Any time we had a reduction in the number of test cases, we saw at least a 30% reduction, and any time we had an increase in the number of test cases, we had a drastic improvement in our test coverage.

# 5. Conclusion

Combinatorial Test Design is a great test planning method and should be considered in any testing strategy when the software under test is in a parameterized structure. There are several benefits that can be realized with CTD including time savings in test planning and test execution, along with optimizing existing test suites.

The method is not going to meet the entire test planning needs of a team, but it can serve as a valuable supplemental process for systems with a high number of attribute and value interactions. Cambia has been able to realize real financial, and time, savings with this method and we're looking to expand into other areas of our organization.

# References

Web Sites:

National Institute of Standards and Technology (NIST). "Combinatorial and Pairwise Testing." - http://csrc.nist.gov/groups/SNS/acts/index.html (accessed July 10th, 2014).

Hexawise. http://www.hexawise.com (accessed June 14th, 2014).

Wikipedia. "Cartesian product". http://en.wikipedia.org/wiki/Cartesian_product (accessed June 14th, 2014).