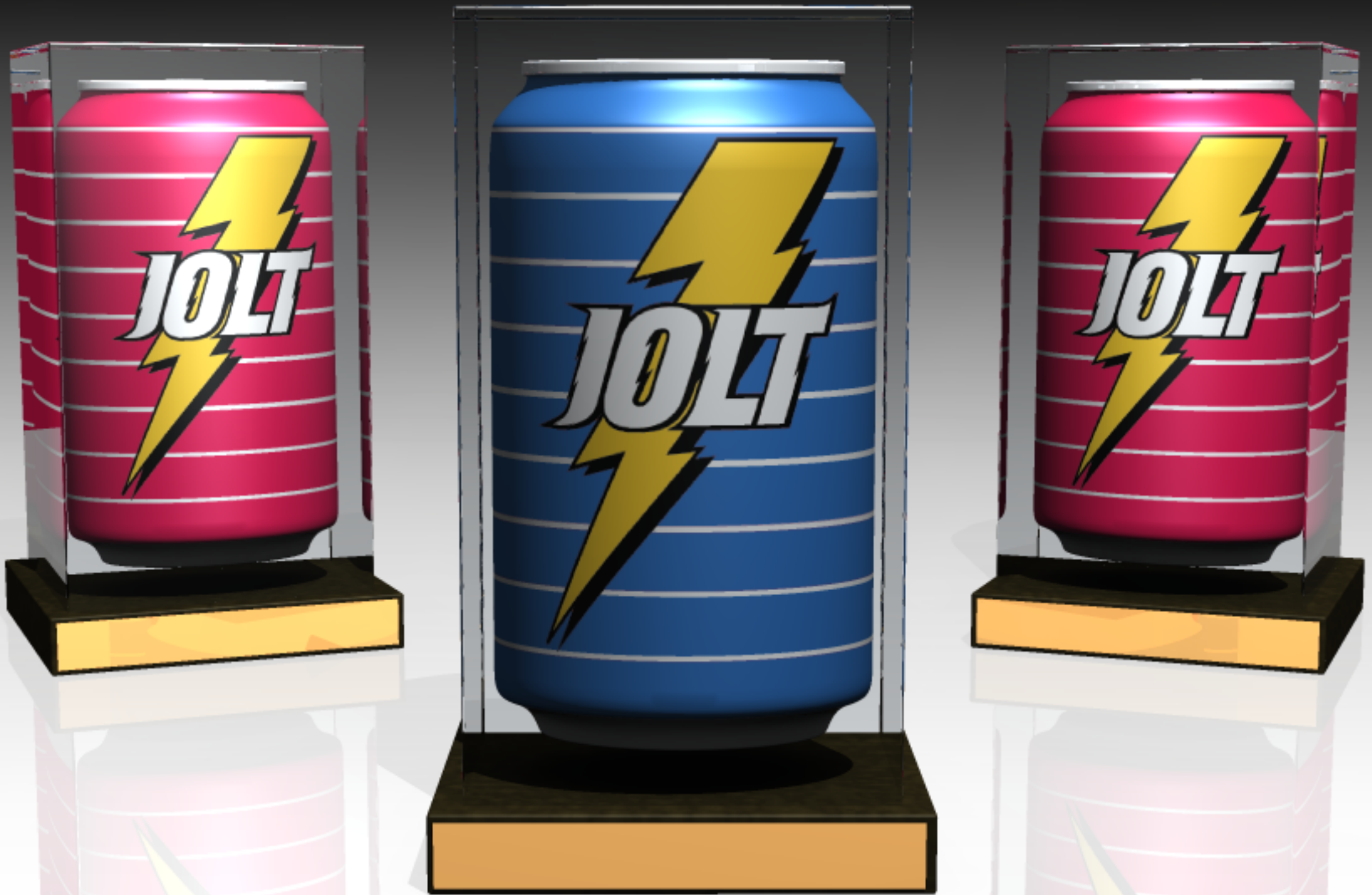


# Jolt Product Excellence Award

## 2006 + 2008 + 2007



**The Leading Agile  
Lifecycle Management Solution**



Sign up for **FREE** Community Edition

Rally's award-winning Agile life cycle  
management tool for a single team!

June 2008

\$9.95 [www.StickyMinds.com](http://www.StickyMinds.com)

# BETTER SOFTWARE

**GIMME A "D"**  
A further look at  
the D language

**IT'S ALL IN  
YOUR HEAD**  
The illusion of  
risk management

The Print Companion to **StickyMinds.com**

## AGILE MODEL-DRIVEN DEVELOPMENT

SCALING AGILE TO MEET  
THE NEEDS OF  
REAL-WORLD PROJECTS



**FIND THE BUG INSIDE & WIN  
AN IPOD SHUFFLE™!**

# SERIOUS SOURCECODE ANALYSIS

Klocwork helps software developers create better code.

When developing mission-critical embedded software, developers must quickly and accurately identify, assess and fix critical software bugs and security vulnerabilities right at their desktop before they impact anyone else.

Klocwork's leading static source code analysis tools provide powerful, collaborative analysis of your C, C++ and Java code at the earliest point in the software development process – before code check-in – when detected issues are easiest and less costly to fix, freeing developers to do what they do best – innovate.

Take the first step towards better code. Visit [www.klocwork.com/freetrialsignup](http://www.klocwork.com/freetrialsignup) for a free product trial.

**Klocwork**  
[www.klocwork.com](http://www.klocwork.com)

Visit the Klocwork booth at the Better Software Conference and Expo 2008 (June 9–12) and attend our session Source Code Analysis: The Basics and Beyond with Gwyn Fisher, CTO on June 12 at 10:15am.







Take the

# handcuffs off

quality assurance

**Empirix gives you the freedom to test *your way*.**

Tired of being held captive by proprietary scripting? Empirix offers a suite of testing solutions that allow you to take your QA initiatives wherever you like.

**Download our white paper, *Lowering Switching Costs for Load Testing Software*, and let Empirix set you free: [www.empirix.com/freedom](http://www.empirix.com/freedom).**

Introducing QAZone by Empirix— a new online community dedicated to fostering communication and collaboration among QA and IT professionals. Register for a free account: <http://qazone.empirix.com>

**EMPIRIX**  
Be Confident.



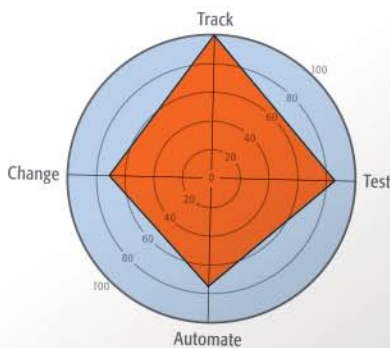
# Seapine Software

## Seapine Software Quality-Ready Assessment



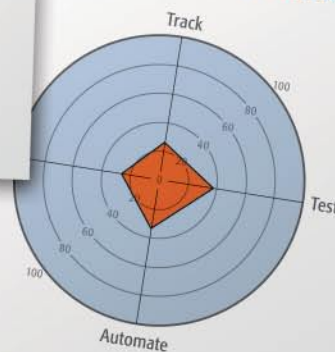
- Not prepared for an audit
- Little or no automation

## Seapine Software Quality-Ready Assessment



- Audit-ready development artifacts
- Strong testing methodology and automation

## Seapine Software Quality-Ready Assessment



- Lacking in most areas of quality
- Probably an "It's good enough" organization

# What's Your Quality-Ready Score?

You need the right tools to build a quality-ready advantage. Take the Seapine Software Quality-Ready Assessment to measure your development and QA teams' ability to create a sustainable competitive advantage through integrated ALM and learn how to make your organization's productivity soar.

Find out your score at [www.seapine.com/qra](http://www.seapine.com/qra)



Find the Bug  
and win an iPod Shuffle™!  
Search through the digital  
edition to find the flying  
bug. Click it to be entered for a  
chance to win an iPod Shuffle™.

## Cover Story

### AGILE MODEL-DRIVEN DEVELOPMENT 22

Despite what you might have heard, modeling is an important part of agile software development. Find out how agile model-driven development fits into the overall agile development lifecycle. *by Scott W. Ambler and Celso Gonzalez*

## Features



### THE MYTH OF RISK MANAGEMENT 30

Risk management is an illusion. We must recognize that software projects are inherently risky and admit to ourselves that it's not the known problems that are going to cause our projects to fail. *by Pete McBreen*

### STOP THE INSANITY! 36

We've all heard Einstein's definition of insanity, and it definitely holds true in software development. We are going to make mistakes in product development, but root-cause analysis can help us understand those mistakes and be proactive in not repeating them. *by Ed Weller*

## Columns & Departments

### In Every Issue

Mark Your Calendar 4

Contributors 8

eLightenment 12

Product Announcements 44

10 Things You Might  
Not Know About ... 46

Ad Index 48

**Better Software magazine**—The print companion to StickyMinds.com brings you the hands-on, knowledge-building information you need to run smarter projects and deliver better products that win in the marketplace and positively affect the bottom line. **Subscribe today to get ten issues.**

Visit [www.BetterSoftware.com](http://www.BetterSoftware.com)  
or call 800.450.7854.

### TECHNICALLY SPEAKING 11

**The Mission Is the Message** • *by Lee Copeland*

A mission statement is supposed to guide and inspire the members of an organization. Is your statement sending the right message?

### CODE CRAFT 14

**A "D" in Programming, Part 2** • *by Chuck Allison*

In Part 2 of his pitch for the D programming language, Chuck explores some powerful features of the language using an example from a previous Code Craft.

### TEST CONNECTION 18

**Know Where Your Wheels Are** • *by Michael Bolton*

Drawing from his experiences while learning to drive, Michael applies those lessons to teaching new testers some valuable skills.

### MANAGEMENT CHRONICLES 20

**Advice for the New Leader** • *by Michele Sliger*

Learn how to guide your team to success by stepping back and letting team members solve their own problems and learn from their mistakes.

### THE LAST WORD 47

**How to Fail Less and Enjoy More** • *by Frédéric Boulanger*

The shiniest software application in the world, shipped on time and under budget, is a failure if it doesn't make someone's job easier.

**StickyMinds.com** We invite you to visit StickyMinds.com, the online companion to *Better Software* magazine. StickyMinds.com covers the same pertinent topics as the magazine, putting the power of information at the click of your mouse. Weekly columns, headline-making bugs, hundreds of technical papers, an online tools guide, discussion boards, and so much more make StickyMinds.com your site for 24/7 brainfood to help you build better software.



## MARK YOUR CALENDAR

### TRAINING WEEKS

[www.sqetraining.com](http://www.sqetraining.com)

#### Testing

**September 8–12, 2008**

New York/New Jersey Area

**September 15–19, 2008**

Washington, DC

#### Agile Software Development

**September 8–12, 2008**

Washington, DC

**September 22–26, 2008**

Denver, CO

### SOFTWARE TESTING CERTIFICATION

[www.sqetraining.com/stf](http://www.sqetraining.com/stf)

**August 26–28, 2008**

Boston, MA

**September 8–10, 2008**

New York/New Jersey Area

**September 9–11, 2008**

Minneapolis, MN and Salt Lake City, UT

**September 15–17, 2008**

Washington, DC

### CONFERENCES

#### STARWEST2008

##### Software Testing Analysis & Review

[www.sqe.com/starwest](http://www.sqe.com/starwest)

**September 29–October 3, 2008**

Disneyland Hotel

Anaheim, CA

#### Agile Development Practices 2008

[www.sqe.com/agiledevpractices](http://www.sqe.com/agiledevpractices)

**November 10–13, 2008**

Shingle Creek Resort

Orlando, FL

# BETTER SOFTWARE

Publisher

**Wayne Middleton**

Vice President of Publishing

**Holly N. Bourquin**

#### Editorial

Editor

**Heather Shanholtzer**

Managing Technical Editor

**Lee Copeland**

Technical Editors

**Chuck Allison**

**Jonathan Kohl**

**Antony Marciano**

Editor, StickyMinds.com

**Francesca Matteu**

Managing Editor, Multimedia

**Joseph McAllister**

Production Coordinator

**Cheryl M. Burke**

#### Design

Creative Director

**Catherine J. Clinger**

#### Advertising

Senior Advertising Sales Manager

**Shae Young**

Advertising Sales Manager

**Joe Anderson**

Production Coordinator

**April Evans**

#### Circulation and Marketing

Marketing Coordinator

**Megan Brown**

Circulation Coordinator

**Jamie Green-Gago**

A PUBLICATION OF SOFTWARE QUALITY ENGINEERING



#### CONTACT US

Editors: [editors@bettersoftware.com](mailto:editors@bettersoftware.com)

Subscriber Services: [info@bettersoftware.com](mailto:info@bettersoftware.com)

Phone: 904.278.0524, 888.268.8770

Fax: 904.278.4380

Address:

*Better Software* magazine

Software Quality Engineering, Inc.

330 Corporate Way, Suite 300

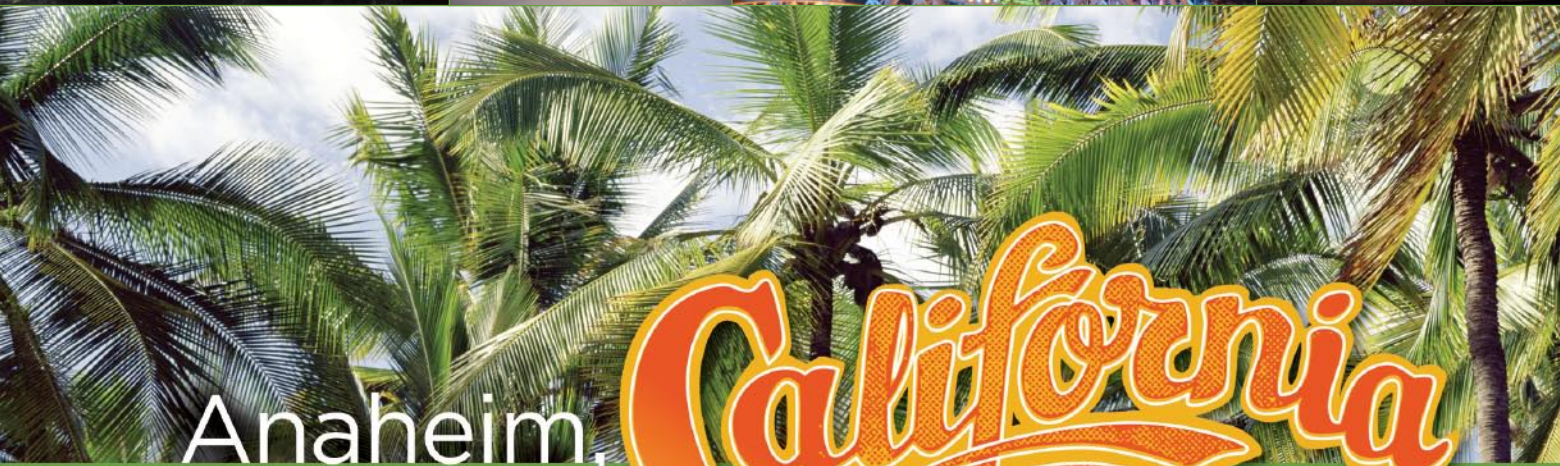
Orange Park, FL 32073



# SOFTWARE TESTING ANALYSIS & REVIEW

*The Greatest Software Testing Conference on Earth*

Sept. 29–Oct. 3, 2008 • The Disneyland® Hotel



Anaheim,

California

**99.7% of 2007 Attendees  
Recommend STARWEST  
to Others in the Industry**

[www.sqe.com/starwest](http://www.sqe.com/starwest)  
REGISTER EARLY AND SAVE \$200!



[www.sqe.com](http://www.sqe.com)







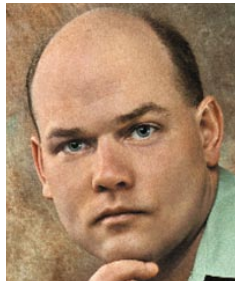
\_We're all back to normal size again. Except Gil's brain. Which is still a bit on the tiny side.

Download the Systems Development e-Kit at:  
**IBM.COM/TAKEBACKCONTROL/SYSTEMS**





**CHUCK ALLISON** developed software for more than twenty years before becoming a professor of computer science at Utah Valley University. He is a technical editor for *Better Software* magazine and founding and current editor of the online journal *The C++ Source*. He spent most of the 1990s as an active member of the C++ Standards Committee and is author of two C++ books, including *Thinking In C++, Volume 2: Practical Programming*, with Bruce Eckel. His company, Fresh Sources, Inc., gives onsite training in C++, Python, and design patterns. His current top technical interest is the resurgence of functional programming. Whenever he finds a little down time he plays classical guitar or bikes the country roads of central Utah. Contact Chuck at [chuck@freshsources.com](mailto:chuck@freshsources.com).



**SCOTT W. AMBLER** is the Practice Leader Agile Development with IBM Rational. He is author of several books, including *Agile Modeling*, *Refactoring Databases*, and *The Elements of UML 2.0 Style*. Scott works with IBM customers worldwide to adopt agile software development techniques at scale.



**MICHAEL BOLTON** lives in Toronto and teaches heuristics and exploratory testing in Canada, the United States, and other countries. He is co-author, with James Bach, of *Rapid Software Testing* and a regular contributor to *Better Software* magazine. Contact Michael at [mb@developsense.com](mailto:mb@developsense.com).



**FRÉDÉRIC BOULANGER** is the founder and president of Macadamian Technologies, a software creation partner that specializes in taking next generation products from napkin sketch to market-ready. Under Frédéric's leadership, Macadamian has grown from four founders to a team of more than 130 and is known as one of the most respected software development firms in the industry.



**LEE COPELAND** has more than thirty years of experience in the field of software development and testing. He has worked as a programmer, development director, process improvement leader, and consultant. Based on his experience, Lee has developed and taught a number of training courses focusing on software testing and development issues. Lee is the managing technical editor for *Better Software* magazine, a regular columnist for StickyMinds.com, and the author of *A Practitioner's Guide to Software Test Design*. Contact Lee at [lcopeland@sqa.com](mailto:lcopeland@sqa.com).



# Driving Business Success with Software Reuse

## Leveraging an SCCM Solution for Effective Component Based Development

### Introduction

As enterprise software systems continue to grow in complexity, IT is struggling to find better ways to meet the increasing business demands. With pressure to create higher quality software more quickly and at less cost, enterprise IT must find ways to streamline development. One way that organizations are meeting this challenge is by implementing ways to drive a successful software reuse initiative. The upside to this approach is considerable: devising an effective software reuse strategy enables software development teams to construct enterprise software systems through assembly of reusable parts, and Component Based Development (CBD) serves as a critical enabler of realizing higher degrees of reusability. While CBD is not new, immature technologies and short-term approaches have hindered previous efforts and have soured some on the CBD journey. But that is changing, and many are looking at CBD with fresh eyes.

Today, proven technologies and tools exist that allow teams to develop, share, and manage the evolution of their software assets. With many organizations embarking on large-scale Service Oriented Architecture (SOA) initiatives, Component Based Development (CBD) stands to play a critical role in realizing higher degrees of reuse across enterprise services and applications. In fact, CBD is one of the few technologies to have successfully bridged the gap between commercial and open source software development. At present, the open source community thrives on componentization as a major means of achieving high degrees of reusability. Realizing maximum value and return on investment from your enterprise software development initiatives in today's diverse software ecosystem demands an effective reuse strategy that features CBD as its principle driver.

### Component Based Development and SOA

In many ways, Service Orientation is an evolution of the primary tenets of Component Based Development, and a stated SOA goal is to fulfill mission critical business processes through an orchestration of reusable services. But SOA also represents a significant step forward, in that it offers the possibility of helping IT and business achieve greater alignment through services that expose discreet business functions. While SOA has the potential to increase alignment between IT and business, the services development faces many of the same challenges that software development teams have dealt with for decades. One critical dilemma is preventing problems that arise from the inability to manage change and evolutionary growth. An effective software reuse strategy, achieved through componentization provides a significant benefit in realizing your SOA goals.

Let's illustrate this benefit with a simple example. Figure 1 shows a sample Use Case diagram for an insurance company Claim Management System (CMS). A Customer Service Representative enters the claim upon receiving a notice of loss from the customer. After the claim has been entered into the system, workflow rules dictate to which Claim Adjustor the claim is routed. After receiving the claim, the adjustor can evaluate the loss, enter the claim into their electronic claim file, and eventually process the claim to settlement.

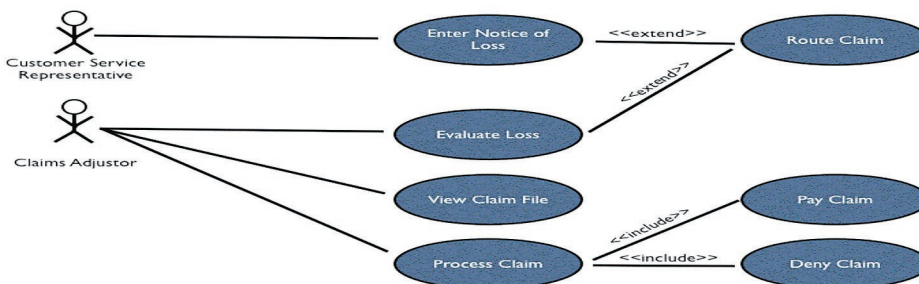


Figure 1 - Use Case Diagram for a Claims Processing System



Figure 2 shows a component diagram for the CMS illustrating two separate Web applications and a number of services and components. Each service fulfills a core business function, and is composed of one or more reusable components that can be deployed and reused across multiple applications and services. The Claim Severity Component is an example of a component deployed and used by both the workflow and adjudication service to help score the severity of the claim and ensure that the appropriate rules are applied during adjudication. In this example, component reuse helps speed the development of applications and services through a combination of custom development and component assembly.

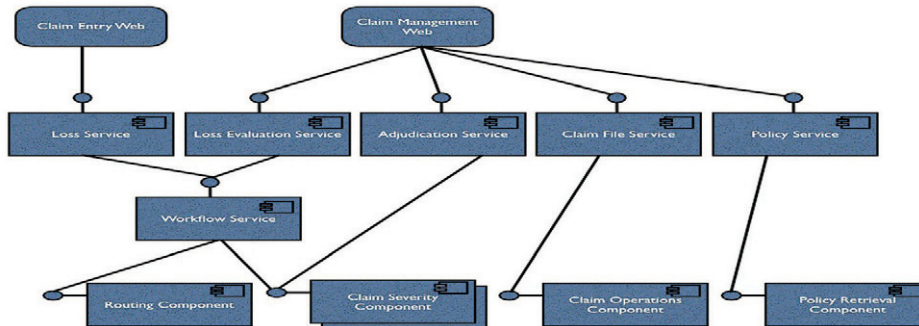


Figure 2 - Component Diagram Illustrating the CMS Services and Components

Managing the development of these components is crucial to realizing the advantages of CBD and SOA. Yet managing this development is also difficult, not only because of complex technical challenges, but also due to the essential complexity surrounding software development. Often not considered a significant part of a CBD strategy, a robust Software Configuration and Change Management (SCCM) solution is a fundamental aspect to consider for project success. Telelogic Synergy™ and Change™ is a product suite for change and Configuration Management that can help ease your pains surrounding enterprise CBD and SOA initiatives. Before delving into how the Synergy and Change contribute to CBD, let's first examine the benefits of CBD followed by some common impediments.

## Benefits of Software Reuse

Software reuse has been considered the “holy grail” of software development for decades. The business value realized through successful reuse initiatives can offer substantial rewards, which explains why so many organizations are pursuing this goal. Below are a few of the significant advantages teams stand to realize through a successful CBD initiative for achieving reuse:

- **Reduced time-to-market:** CBD emphasizes assembly of applications from pre-built parts. Robust components have already undergone rigid testing cycles. Because components are modularized and independent units, they can often be developed in parallel. Given the proper collaboration and testing tools, teams can work effectively even when dispersed across the globe.
- **Higher Quality:** Components are discreet fine grained units of behavior, and automated tests can be written to ensure that each component undergoes a strict test lifecycle. As components are field tested as they are consumed by an increasing number of applications, developers gain increased trust in CBD.
- **Decreased cost:** Increased component modularity helps isolate software functions, providing an efficient mechanism to upgrade a component without impacting an entire application. Although the short-term cost of CBD is typically greater than developing applications in silos, as your component reuse increases, the reduced time-to-market and higher quality benefits offer substantial long-term savings.
- **Increased Organizational Agility:** The ability of IT to respond quickly to business needs is imperative in today's dynamic business climate. As the component inventory of your organization grows, teams are able to enhance application functionality by leveraging pre-built software components. Additionally, because change requests of the business stakeholders are isolated to discreet units of functionality, change can be made in isolation and released to all component consumers.

For the complete “Driving Business Success with Software Reuse” White Paper, please visit [www.telelogic.com/softwarereuse](http://www.telelogic.com/softwarereuse).

### Global Headquarters

P.O. Box 4128, SE-203 12  
Malmö, Sweden  
P: + 46 40 650 00 00  
F: + 46 40 650 65 55

### Americas Headquarters

9401 Jeronimo Road  
Irvine, CA 92618 USA  
P: + 1 949 830 8022  
F: + 1 949 830 8023

Offices across Europe, America, Asia  
and Australia. Distributors worldwide.

[info@telelogic.com](mailto:info@telelogic.com)  
[www.telelogic.com](http://www.telelogic.com)



**CELSON GONZALEZ** is one of the Rational World Wide Architecture Management leaders. His role is to provide expertise to IBM customers and internal resources in domains ranging from business modeling to J2EE development—including requirements management, architecture, and design. Lately Celso has been focusing on accelerating development using patterns-based engineering.



**TIM LISTER** is a principal of The Atlantic Systems Guild ([www.systemsguild.com](http://www.systemsguild.com)). Tim is a well-known expert and lecturer on software management techniques. He co-authored the popular book *Peopleware: Productive Projects and Teams*.



**PETE MCBREEN** is the author of *Software Craftsmanship* and *Questioning Extreme Programming*. He is an independent consultant who actually enjoys writing and delivering software. Despite spending a lot of time writing, teaching, and mentoring, he does hands-on coding on a live project every year. Pete specializes in finding creative solutions to problems that software developers face. After many years of working on formal and informal process improvement initiatives, he took a sideways look at the problem and realized, "Software development is meant to be fun. If it isn't, the process is wrong." Pete lives in Cochrane, Alberta, Canada with no plans to move back to a big city.



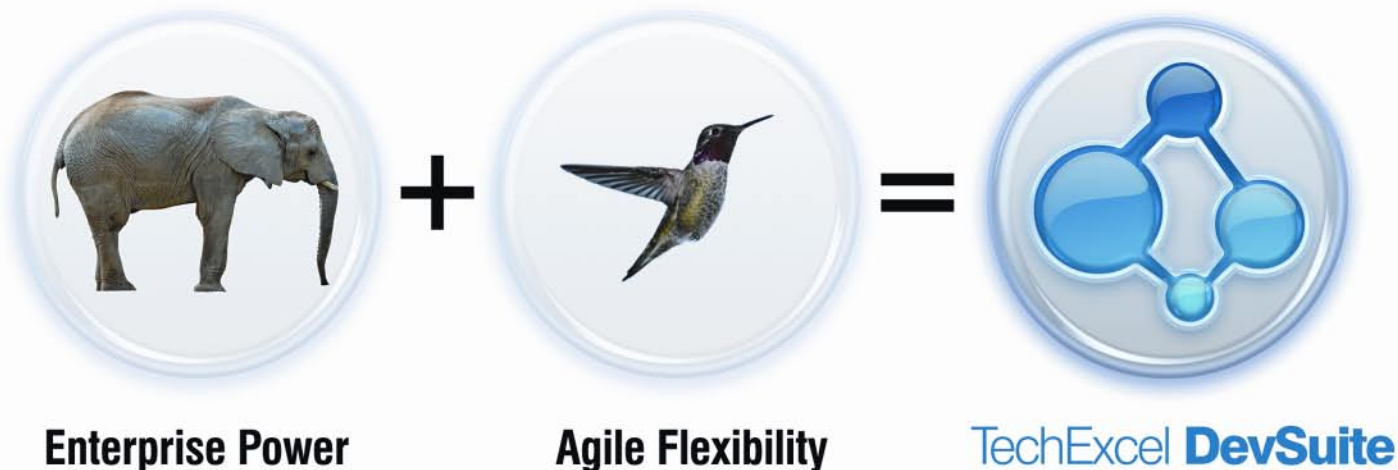
**MICHELE SLIGER** has extensive experience in agile software development, having worked in both XP and Scrum teams before becoming a consultant. As a self-described "bridge builder," her passion lies in helping those in traditional software development environments cross the bridge to agility. Her book *The Software Project Manager's Bridge to Agility* (with co-author Stacia Broderick) will focus on that topic, helping PMI-trained project managers make the transition. A Certified Scrum Trainer (CST) and a certified Project Management Professional (PMP), Michele can be reached at [michele@sligerconsulting.com](mailto:michele@sligerconsulting.com).



**ED WELLER** is an SEI certified High Maturity Appraiser for CMMI appraisals, with nearly forty years of experience in hardware and software engineering. Ed is the principal of Integrated Productivity Solutions, a consulting firm that is focused on providing solutions to companies seeking to improve their development productivity. A regular columnist on StickyMinds.com, Ed can be contacted at [edwardwellerIII@msn.com](mailto:edwardwellerIII@msn.com).







**Now projects of any size can be agile.**



# DevSuite

The implementation platform for scalable, agile development.

- Out of the box configurations for Scrum, iterative development and CMMI
- Quantify requirements and design with specifications
- Specifications drive development and testing
- A fully integrated suite of best-of-breed products:



## DevSpec

Requirements management



## DevPlan

Project planning



## DevTrack

Implementation and issue tracking



## DevTest

QA test management

Learn how to make agile scalable at [www.techexcel.com/scalableagile](http://www.techexcel.com/scalableagile)

**TechExcel**

[www.techexcel.com](http://www.techexcel.com) | 1-800-439-7782

# The Mission Is the Message

by Lee Copeland

Mission statements—those succinct statements of purpose that organizations create to guide and inspire their members and to inform the public of their intent—typically define the organization's purpose, the business it is in, its clients, its responsibilities to those clients, and its main objectives.

Examples of mission statements include:

- “Google’s mission is to organize the world’s information and make it universally accessible and useful.”—Google
- “Establish Starbucks as the premier purveyor of the finest coffee in the world while maintaining our uncompromising principles as we grow.”—Starbucks
- “To be the best pizza for every pizza occasion.”—Pizza Hut
- “To Refresh the World ... in body, mind, and spirit; To Inspire Moments of Optimism ... through our brands and our actions; To Create Value and Make a Difference ... everywhere we engage.”—Coca Cola
- “To bring inspiration and innovation to every athlete in the world. If you have a body, you are an athlete.”—Nike

Lofty and inspirational ideals. However, recently I read this “mission” statement of a major software vendor (hereafter referred to as MSV). I have listed excerpts (CAPITALIZATION is theirs):

MSV warrants that the Product will perform substantially in accordance with the accompanying materials for a period of ninety days from the date of receipt.

“A first step to building better software is to commit to building better software, not hiding behind legal mumbo-jumbo.”

AS TO ANY DEFECTS DISCOVERED AFTER THE NINETY (90) DAY PERIOD, THERE IS NO WARRANTY OR CONDITION OF ANY KIND.

Any supplements or updates to the Product, including without limitation, any (if any) service packs or hot fixes provided to you after the expiration of the ninety day Limited Warranty period are not covered by any warranty or condition, express, implied or statutory.

**YOUR EXCLUSIVE REMEDY.** MSV’s entire liability and your exclusive remedy shall be, at MSV’s option from time to time exercised subject to applicable law, (a) return of the price paid (if any) for the Product, or (b) repair or replacement of the Product, that does not meet this Limited Warranty and that is returned to MSV with a copy of your receipt. You will receive the remedy elected by MSV without charge, except that you are responsible for any expenses you may incur (e.g., cost of shipping the Product to MSV).

Except for any refund elected by MSV, **YOU ARE NOT ENTITLED TO ANY DAMAGES, INCLUDING BUT NOT LIMITED TO CONSEQUENTIAL DAMAGES**, if the Product does not meet MSV’s Limited Warranty.

This Limited Warranty is void if failure of the Product has resulted from accident, abuse, misapplication, abnormal use or a virus.

Since I don’t speak legalese, my son the attorney translated:

This software may do about what we claim for ninety days. After

that, you’re on your own. If it works, great. If it doesn’t, you’re sadly out of luck. If it doesn’t work, and we fix it, and it still doesn’t work, tough luck. We can choose to fix it or give you your money back—guess which we’ll choose. If something really bad happens because you use this software, tough. Money lost, property destroyed, lives ruined—tough.

So, if one purpose of a mission statement is to guide and inspire the members of an organization, what does this statement inspire them to do?

A first step to building better software is to commit to building better software, not hiding behind legal mumbo-jumbo. How about this as a software development “mission statement”:

We’ve done our best to build this software so that it will be useful to you, and we stand behind our work. As software for the “shrink-wrap” market, it may not do precisely what you need. It is your responsibility to evaluate it for your purposes before committing to it; it is our responsibility to implement it against its specifications. We’ll make those specifications available to you if you desire. If our software does not meet its specifications, we’ll use our best efforts to repair it as quickly as possible. And besides, we don’t know how you could “abuse” software so don’t worry about that—we’ll do our best, allowing you to do yours.

Are you ready to adopt this kind of mission statement in your organization? Are you truly ready to commit to building better software? **{end}**



## EDITOR'S PICK



**Heather Shanholtzer**  
Editor, *Better Software*  
magazine  
hshanholtzer@sqa.com

While working on a recent issue of *Better Software* magazine, I had an interesting exchange with Cathy, our art director. I had asked her to use an art element to fill a large white space at the end of an article, and she sent me some options to review.

"I like the distressed crab," she wrote in the accompanying email.

I opened each attachment to view my choices. The first was a clip art crab—nothing special, just a crab. The second image was a cartoon-like crab with a ticked-off look on its face—a bit too cutesy for my taste. The third image was a red, scuffed-up crab silhouette. It was pretty cool and would fit well with the layout of the article where it would be placed.

I emailed Cathy and said, "The distressed crab is cute, but I think the roughed-up one better matches the rest of the art in that article."

A few minutes later I was in her office and Cathy asked me to show her which crab I liked best. I pointed to my favorite, and she smiled. "That's my favorite, too."

"I thought you liked the angry crab?" I responded.

"No," she said. "I told you I liked the *distressed* crab."

And that's when it hit me. Editors and art directors don't always speak the same language. To someone who works with words distressed can mean a lot of things, but the first things that pop into my head are worried, angry, stressed out, in trouble, etc.

To someone who works with images, distressed is a treatment or a style applied to a picture or a font that causes it to appear damaged or scuffed. That's a big difference.

There can be a similar disconnect in the way managers communicate with the programmers on their teams. Trying to understand how a programmer's outlook differs from your own is the first step to creating an environment that works well for the whole team.

So this month's Editor's Pick is "The Proper Care and Feeding of Programmers" by Mike Cohn. Get insight into the innate characteristics that cause programmers to approach problems in ways that leave their managers scratching their heads. Discover what behaviors are likely to cause conflicts and what you can do to work with those traits, instead of against them.

Read "The Proper Care and Feeding of Programmers" at [www.StickyMinds.com/editorspick10-5](http://www.StickyMinds.com/editorspick10-5).

## Jolt Product Excellence Award 2006 + 2008 + 2007



## The Leading Agile Lifecycle Management Solution

Sign Up Free: [www.rallydev.com/bsm](http://www.rallydev.com/bsm)



Scaling Software Agility

See us at the



June 9-12, Las Vegas

## POWERPASS POINTER

### The Case of the Missing Fingerprint

BY JENNITTA ANDREA

"The Case of the Missing Fingerprint" is the tale of a planning spreadsheet and its effect on three different projects. Learn the impact a single decision can have on a project—and pick up some helpful tools like fingerprint graphs and project timelines along the way.

Keep reading at [www.stickyminds.com/powerpass10-5](http://www.stickyminds.com/powerpass10-5).

From a development

Quotables

perspective, we want to prevent

errors due to this root cause from recurring

in future products or development

PAGE  
37

activities. This means we must look

for the underlying situation that allowed us—or

even encouraged us—

to make the error.

You raise an excellent point about business executives learning how to better pick the brains of their techies. Asking just a few good questions could immeasurably help those who are attempting to persuade, both by drawing pertinent information out of them and by giving them insight about what kinds of information they should be prepared to present—if not this time, then next time around.

NAOMI KARTEN RESPONDING TO A COMMENT POSTED ON HER ARTICLE "THE ART OF PERSUADING MANAGEMENT."

[www.stickyminds.com/quotables10-5a](http://www.stickyminds.com/quotables10-5a)

It is the proliferation of just these types of inefficiencies that got me into development. I want to do my part to eradicate them. But as a tester, I find I am often too late.

I suggest we ask the business owner some questions. When do you want to identify these problems? After the system has been designed, coded, unit-tested, validated, accepted, and deployed to the field? Or before system development has begun? Do you want to put more resources into finding these problems during validation or would you prefer to put that effort into finding solutions to these problems during the analysis? I agree that it is better to unearth problems before they get into the field. But I also think that it is more likely that the problems will be addressed if they are identified from the outset.

STICKYMINDS.COM MEMBER KEN TAYLOR COMMENTING ON MICHAEL BOLTON'S "LEARNING THE HARDWARE LESSONS"

[www.stickyminds.com/quotables10-5b](http://www.stickyminds.com/quotables10-5b)

Negotiation is a process that searches for a way to satisfy people who have competing needs. While some people frame negotiation as a power struggle or competition, it is often helpful to reframe negotiation as a problem-solving exercise that seeks the best available solution.

PAYSON HALL, "PROJECT NEGOTIATIONS AND THE IRON TRIANGLE."

[www.stickyminds.com/quotables10-5c](http://www.stickyminds.com/quotables10-5c)

S U C C E E D I N G   W I T H   A G I L E <sup>SM</sup>

## DON'T BE SHY about transitioning to agile.

Face and overcome the potential dangers. Master agile software development techniques and deliver valuable, functional software in a world of uncertainty and change.

Join **Mike Cohn**, author of *User Stories Applied* and *Agile Estimating and Planning*, in courses exploring the principles of agile software development: people over processes, working software over documentation, collaboration over negotiation, and flexibility over rigidity.

For more information, visit  
[www.mountaingoatsoftware.com/better](http://www.mountaingoatsoftware.com/better)



### Upcoming Courses

#### Austin

May 8-9

Certified ScrumMaster for Video Game Development\*

\*with Clinton Keith

#### Boulder

May 12-13

Certified Scrum Product Owner\*

\*with Ken Schwaber

#### Washington, DC

June 3-4

Certified ScrumMaster

June 5

Agile Estimating and Planning

#### La Jolla

July 29-30

Certified ScrumMaster

July 31

Agile Estimating and Planning

#### San Jose

October 13

Effective User Stories

October 14-15

Certified ScrumMaster

October 16

Agile Estimating and Planning



# A “D” in Programming, Part 2

by Chuck Allison

As I write this, my last official pitch for the D programming language, I notice that D is at position twelve and climbing in Tiobe’s ranking of the twenty most popular languages for February 2008 [1]. Earlier in March, the first book on D, *Learn to Tango with D*, hit the shelves [2]. In its preface, D’s designer, Walter Bright, said:

“Amazingly, there is no language that enables precise control over execution while offering modern and proven constructs that improve productivity and reduce bugs ... Often programming teams will resort to a hybrid approach, where they will mix Python and C++, trying to get the productivity of Python and the performance of C++. The frequency of this approach indicates that there is a large unmet need in the programming language department. D intends to fill that need.”

Like C++, D supports down-to-the-metal programming when you need it, and it compiles to fast-and-lean, native executables. It also supports generic programming and templates in all their glory. The standard C library is available directly from D code. Like Python, D supports modules and packages, garbage collection, functions that behave as first-class entities, a clean (though C-like) syntax, and flexible, built-in data structures. Like Java, D has inner classes. Like C#, D gives you delegates, but in a more flexible way. D is a multi-paradigm language that may be just what you need most of the time.

For many of you, I suppose the software engineering features of D would be of most interest, but in this article I’d like to bring to “closure” (pun intended) a running example from previous Code Craft articles as I explore some powerful features of the D language.

## Nested Functions and Closures

Much of what we enjoy today about objects was accomplished in earlier days through other means. To hide data in C, for example, you just declared a file-scope variable to be *static*. The file was the container, the “object,” if you will, that held data values not directly accessible to users. See listing 1.

```
/* file1.c */
static int theData = 7;    // Private data
int getData() { return theData; }
```

Listing 1

This approach breaks down when you need multiple instances. File I/O is too cumbersome and expensive to use as a model for objects that contain hidden data. Object-oriented languages have direct support for in-memory object creation at runtime, of course, as in listing 2.

How did we ever survive without classes?

There were many ways, but I’d like to mention one that is still important: *nested functions*. Nested functions are not



ISTOCKPHOTO

```
// MyClass.java
class MyClass {
    private int theData;
    public int getData() { return theData; }
}
```

Listing 2

permitted in many modern languages but were *de rigueur* in languages like Lisp, Algol, PL/I, Pascal, and Ada. C dropped nested functions for simplicity, but Python and D have brought them back because sometimes they are superior to using objects.

To illustrate, let me return to an example I’ve used in recent Code Craft articles: function composition. In the January 2008 Code Craft, I presented the generic C++ function composer shown in listing 3.

A *Composer* is an object that takes a list of single-valued functions (or entities callable as such) and calls them in turn in nested fashion so you end up with `f1(f2(...fn(x)...) )`. The private data here is the sequence of functions, indicated by the pair of iterators, *beg* and *end*. Since objects aren’t the only way to hide data, let’s look at the nested-function solution in D shown in Listing 4.

This rendition of `compose` is a function template, evidenced by the `(T)` following its name, and it accepts a dynamic array of functions, each of which takes a single `T` argument and returns a `T` value. It returns a *delegate* that is callable as a single-valued function of type `T`. The nested function `doit` iterates through the list of functions, `funs`, in reverse order, applying each function and accumulating the result as it goes. Let’s examine this more closely.

Notice that `compose` has only two statements: a definition of the nested function `doit` and a statement that returns a

```
template<class Iter>
class Composer {
private:
    typedef typename iterator_traits<Iter>::value_type Fun;
    typedef typename Fun::result_type T;
    typedef reverse_iterator<Iter> RevIter;
    RevIter beg, end;
    static T apply(Tsofar, Fun f) {
        return f(ssofar);
    }
public:
    Composer(Iter b, Iter e) : beg(RevIter(e)), end(RevIter(b)) {}
    T operator()(T x) {
        return accumulate(beg, end, x, apply); // Function Applicator
    }
};
```

Listing 3

```
T delegate(T) compose(T) (T function(T) [] funs) {
    T doit(T n) {
        T result = n;
        foreach_reverse (f; funs)
            result = f(result);
        return result;
    }
    return &doit;
}
```

Listing 4

pointer to the function `doit`. The function `doit` itself uses the array `funs`, which is defined in `compose`'s parameter list. It is common to say that `funs` is in the “calling environment” of `doit`.

So what happens when a pointer to `doit` is returned from a call to `compose`? In order for `funs` to persist for use by subsequent calls to `doit`, it has to somehow be saved and connected to `doit`. Consider how `compose` is used in listing 5.

The function keyword has two related uses in D: to declare a function pointer and to define an anonymous *function literal* (like “lambda” expressions in functional programming languages). The first usage occurs in the definition of `compose` in the first line of listing 4 and also in the first line inside `main` in listing 5. In each case, `funs` is declared a dynamic array of single-valued function pointers. In the second and third lines of

```
import std.stdio;

void main() {
    int function(int)[] funs;
    funs ~= function int(int x){return x*x;};
    funs ~= function int(int x){return x+1;};
    auto c = compose(funs);
    writeln(c(3)); // 16
}
```

Listing 5

`main` in listing 5, two unnamed functions are created and stored in `funs`.

The variable `c` holds the delegate returned by the call to `compose`. So what is a delegate in D? It is the very mechanism that allows `funs` to persist and be used by the instance of `doit` returned by a call to `compose`. A delegate is a *pair* that contains a pointer to a function (which could be a class method) and the calling context of the function. That calling context could be an activation (stack frame) of an enclosing function, as `compose` is for `doit`, or it could be an object or class used as context for a method. So that its calling context—the activation of the call to `compose`—is preserved for `doit` to do its work, `doit` must be re-

turned as a delegate. That context is preserved even after `compose` has terminated. Such a persistent calling context is called a *closure*. The closure for `compose` is moved from the runtime stack to the garbage-collected heap so it persists as long as the variable `c` in listing 5 does.

There is no need to create a class to solve this problem, and the nested-function approach is simpler anyway.

## Whither Function Objects?

C++ popularized the use of function objects—sometimes called *functors*—with the introduction of STL. A function object is nothing more than an instance of a class that overloads the function-call operator. To illustrate, listing 6 creates a C++ function object, `gtN`, which determines whether its argument is greater than a previously stored value.

Similar code can be written in D using the `opCall` special function, but the nested-function version in listing 7 is simpler.

Once again we have an outer function that returns a nested function that has access to the outer calling context. It appears that nested functions and closures can replace function objects

```
template<class T>
class gtN {
    T n;
public:
    gtN(T x) : n(x) {}
    bool operator()(T m) {
        return m > n;
    }
};

int main() {
    gtN<int> g5(5);
    cout<< g5(1) <<endl; // false
    cout<< g5(6) <<endl; // true
}
```

Listing 6



altogether in D. These classic language constructs are as useful today as ever. **{end}**

#### REFERENCES:

- 1] [www.tiobe.com/tiobe\\_index/index.htm](http://www.tiobe.com/tiobe_index/index.htm)
- 2] Bell, Kris; Igesund, Lars Ivar; Kelly, Sean; and Parker, Michael. *Learn to Tango with D*. Apress, 2007.

```
bool delegate(T) gtn(T) (T n) {
    bool doit(T m) {
        return m > n;
    }
    return &doit;
}

void main() {
    auto g5 = gtn(5);
    writeln(g5(1)); // false
    writeln(g5(6)); // true
}
```

Listing 7



**Have you ever wished for a language with the power and ease of a scripting language and the efficiency of C? Have you given D a try?**

Follow the link on the [StickyMinds.com](http://StickyMinds.com) homepage to join the conversation.

## To load test your website, you could type this:

```
Definitions
! Standard Defines
Include "RESPONSE_CODES.INC" Include "GLOBAL_VARIABLES.INC"
CHARACTER*512 USER_AGENT Integer USE_PAGE_TIMERS CHARACTER*
CHARACTER*1024 cookie_2_0 CHARACTER*1024 cookie_2_1 Timer T_
Code
!Read in the default browser user agent field
Entry[USER_AGENT,USE_PAGE_TIMER Start Timer T_OBFUSCATED
PRIMARY GET URI "http://yahoo.chHTTP/1.1" ON 1 &
HEADER DEFAULT_HEADERS &
,WITH {"Accept: image/gif, image/xbitmap, image/jpeg, image/p
"application/x-shockwave-flash, application/msword, */*", &
```

**or this:**

**[www.webperformanceinc.com](http://www.webperformanceinc.com)**



Why code every test case by hand, when our unique software detects and automatically configures the test cases for you – quickly and accurately, then gives you superior reports that are easy to understand? With Web Performance automatic load testing, the time and money you save could increase productivity as much as 500 percent.

For more information about how you can increase performance and productivity using Web Performance automated load testing, visit [www.webperformanceinc.com](http://www.webperformanceinc.com)

## STOP THE FREAK, KILL THE CREEP, BRING ORDER TO YOUR ALM TECHNIQUE

## THE COMPLETE ALM SOLUTION ON TIME, ON BUDGET, ON THE MARK

Without oversight, software projects can creep out of control and cause team to freak. But with Software Planner, projects stay on course. Track project plans, requirements, test cases, and defects via the web. Share documents, hold discussions, and sync with MS Outlook®. Visit [SoftwarePlanner.com](http://SoftwarePlanner.com) for a 2-week trial.



[www.softwareplanner.com](http://www.softwareplanner.com)



Codonomicon whitepaper:

## How to integrate FUZZING and security testing into SDLC

**1**

Introduction to fuzzing

By Heikki Kortti, Codenomicon

**2**

End user perspective to fuzzing

By Jon Oltsik, Enterprise Strategy Group

**3**

Security Test Product of the Year 2008

By Juan Rosales, research analyst, Frost and Sullivan

CODENOMICON Ltd. | [info@codenomicon.com](mailto:info@codenomicon.com) | [www.codenomicon.com](http://www.codenomicon.com)

Tutkijantie 4E | FIN-90570 OULU | FINLAND | +358 424 7431  
101 Metro Drive | Suite 660 | San Jose, CA 95110 | UNITED STATES | +1 408 392 9000



## 1 Introduction to fuzzing

By Heikki Kortti, Codenomicon

Fuzzing or fuzz testing means sending malformed or invalid inputs to a software, device or system in order to find critical flaws and vulnerabilities. During the past 10 years, fuzzing has become increasingly popular as a low-cost but highly effective way of hardening implementations against external attacks.

Fuzzing enables software testers, developers and auditors to easily find defects that can be triggered by malformed inputs via external interfaces. This means that fuzzing is able to cover the most exposed and critical attack surfaces in a system relatively well, and identify many common errors and potential vulnerabilities quickly and cost-effectively.

Fuzzing is especially useful in analyzing black-box systems, as it does not require any access to source code. Having access to information such as source code, design or implementation specifications, debugging or profiling hooks, logging output, or details on the state of the system under test or its operational environment will help in root cause analysis of any problems that are found, but none of this is strictly necessary.

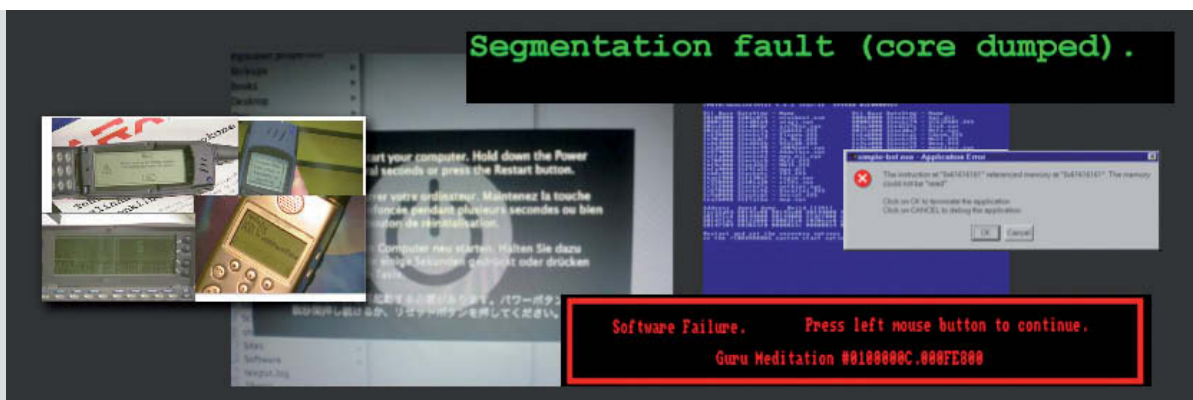
Fuzzing is often compared to code auditing and other white-box testing methods. While code auditing is another highly valuable technique in a software tester's or developer's toolbox, code auditing and fuzzing are really complementary to each other. Fuzzing focuses on finding some critical defects quickly, and the found errors are usually very real. With fuzzing, there are no false positives.

Some fuzzers are implemented as fuzzing frameworks, which means that they provide an end-user with a platform for creating fuzz tests. Fuzzing frameworks typically require a considerable investment in time and resources to develop tests for a new interface. If the framework does not offer ready-made test data for common structures and elements, efficient testing also requires considerable expertise in designing inputs that are able to trigger faults in the tested interface.

In contrast to fuzzing frameworks, another category of fuzzers consists of test suite-based fuzzers. These package a set of tests that have been pregenerated with fuzzing methods into a test suite or test tool that can be used in actual testing. Usually this type of fuzzing requires only minimal work from the end-user (tester), as the interface model as well as test case definitions have already been created beforehand. The tester needs to configure only a few basic settings to start running the tests, and does not even need to fully understand the specifications or other details of the tested protocol or file format.

Another dimension for comparing fuzzers stems from whether they are model-based or not. Compared with a static, non-stateful fuzzer which may not be able to simulate any protocol deeper than an initial packet, a fully model-based fuzzer is able to test an interface more completely and thoroughly, usually proving much more effective in discovering flaws in practice. Tests executed by a fully model-based fuzzer are usually able to penetrate much deeper within the system under test, exercising the packet parsing and input handling routines extremely thoroughly, and reaching all the way into the state machine and even output generation routines.

Security needs to be integrated into every step of the software development life-cycle (SDLC). Neither fuzzing nor code auditing is able to provably find all possible bugs and defects in a tested system or program. As a rule of thumb, the effectiveness of fuzzing is based on how thoroughly it covers the input space of the tested interface (input space coverage), and how good are the representative malicious and malformed inputs for testing each element or structure within the tested interface definition (quality of generated inputs). Fuzzers which use templates or network captures as the model will reach very low input space coverage. Fuzzers which populate their tests with random or semi-random data will have bad quality of inputs and can have significant number of meaningless tests.



For further information on fuzzing, check out: <http://www.codenomicon.com/products/buzz-on-fuzzing.shtml>

## 2 End user perspective to fuzzing

By Jon Oltsik, Enterprise Strategy Group

Fuzzing has proven to be particularly effective when testing for software exposure and system failure risks because it:

- Has broad application. Since black box fuzzing is independent of individual development projects, it can be applied in numerous use cases. For example, fuzzing can be used to analyze the behavior of file formats, networking and application protocols, APIs, etc.
- Provides “wide and deep” coverage. Leading fuzzing tools support a wide range of protocols from Layer 2 through Layer 7 of the OSI stack. This in itself is valuable, but many offer extensive test suites to exercise all aspects of these protocols. Many users have found problems in secondary esoteric protocols that impact overall software security, performance, and availability. With this “wide and deep” capability, black box fuzzing tools can test each and every protocol, not just the obvious ones.
- Eliminates the need for source code and protocol knowledge. Most test engineers would agree that more testing is always better, but many organizations don’t have the resources, time, or skills to develop hundreds of new scripts to test software on their own. Fuzzing tools eliminate this restriction by aggregating testing expertise and a multitude of test routines into a turnkey solution. With this type of coverage, software engineers can spend their time testing code rather than studying protocols or developing their own test scripts.

The following findings are based on interviews conducted with users of commercial fuzzing tools.

The key driver in fuzzing is the capability to extend testing into new domains, namely security testing, and through a turnkey solution:

- “There is so much going on in the operating system in terms of protocol support and we wanted to make sure that we understood how these protocols impacted the security and robustness of our software. Black box testing gave us a much broader use case.” (Software Company)
- “Originally, we found some unstable behavior with some protocols when using Nessus for testing purposes. We adopted [fuzzing] tools soon afterward, because we wanted to exercise more protocols through a thorough set of test cases. [Fuzzing] tools certainly fit this requirement. (Telecommunications Equipment Company)
- “We needed protocol specific tests and didn’t have the resources to develop our own test for CIFS, RTP, SSL and lots of others. [The commercial fuzzing tool] gave us support for every protocol we needed.” (Software Company).

What criteria do the customers use when choosing the fuzzing product? Vendor reputation and the test coverage came up high on the list:

- “Some of our team members had worked with Codenomicon before and suggested that we try it. I can’t tell you how important these types of personal references are when you are about to purchase and use something you have little experience with.” (Software Company).
- “The most important thing is this: Codenomicon’s protocol depth and state is the best in the market. The documentation is also awesome; you can see every test case.” (Telecommunications Equipment Company)
- “We really thought we would use open source tools, but the more protocols we decided to test, the more work we had to do. Codenomicon supports almost every protocol we wanted to test, so it made economic sense to purchase DEFENSICS rather than spend dedicated time and resources toward customizing open source.” (Software Company)

People also like using software based solutions compared to appliance based tools. Software will be easier to use corporate-wide, and results in more users and better results in a shorter time frame.

- “We like the idea of a software-based system. We can easily modify the system with new operating systems, more network cards, or more complex configurations. We are also able to select preferred hardware vendors that have established good SLAs.” (Telecommunications Equipment Company)
- “Since Codenomicon is a software-based system, we were able to purchase DEFENSICS using our operating rather than capital budget. This made the decision easy.” (Telecommunications Equipment Company).

Finally, the ultimate purpose of fuzzing is to find flaws. The value of fuzzing is in the proactive elimination of security critical issues in software.

- “Based upon our models, we know that if customers find software bugs, it cost about \$32,000 to fix. If we catch a bug during our software verification phase, it costs between \$4,000 and \$8,000 to fix. If we find software bugs in development, it only costs around \$600 to \$800 to fix them. Obviously, our goal is to find bugs as early as we can, and Codenomicon is helping us do this.” (Telecommunications Equipment Company)

In summary, black box testing is a simple matter of doing more with less. In this case, commercial fuzzing tools provide more test suites, protocol support, and fuzzing capabilities while decreasing the need for deep protocol knowledge, test suite development, and source code expertise.

**“Why did we adopt [fuzzing] tools?  
That’s easy – our customers told us to. We are finding this  
true of more and more of our carrier customers.”**

*- Telecommunications Equipment Company*

For more customer comments on fuzzing, check out: <http://www.codenomicon.com/resources/whitepapers/2008-customer-view.shtml>

## 3 Security Test Product of the Year 2008

By Juan Rosales, research analyst,  
Frost and Sullivan

The 2008 Frost & Sullivan Award for Product of the Year in the World security test market is presented to Codenomicon Ltd. (Codenomicon) in recognition of DEFENSICS 3.0, the most recent version of the company's flagship software-based test platform targeted at developers, service providers, and enterprises who deal with a variety of interfaces in the network equipment they make or utilize in their networks. Providing pre-emptive security and robustness testing for Internet, wireless and digital media systems, DEFENSICS 3.0 covers over 140 network protocols, digital media formats, and wireless interfaces, allowing for fast test execution. Extremely user-friendly, DEFENSICS 3.0 features a new and improved graphical user interface (GUI) that enables technicians to control multiple test runs from a single interface. The new platform also allows for easy migration from version 2.0, with each important test tool receiving a full update. Furthermore, DEFENSICS 3.0 has improved upon the test coverage capabilities of its preceding versions, allowing the end users to alter their test cases based on time and priority while producing custom tests. As a result, DEFENSICS 3.0 is poised to be a premier solution for robustness testing in security test applications.

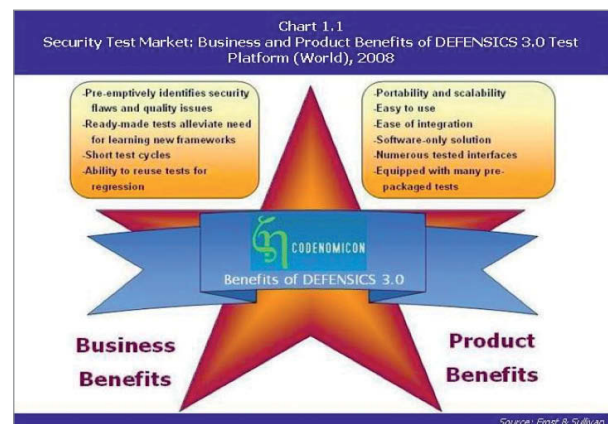
Headquartered in Oulu, Finland, Codenomicon markets its testing software and services directly and through international partners. Codenomicon's customers include Alcatel-Lucent, AT&T, Cisco Systems, F5 Networks, Nordea, Nortel, Microsoft and Siemens AG among many others. The company is privately held with investments from Equitec Partners and Prime Technology Ventures. Codenomicon, whose main objective is to ensure the security and robustness of any application or service implementation, has been recognized by the industry for its innovations in systematic blackbox negative testing. Development and security personnel in lab or staged environments utilize Codenomicon's DEFENSICS platform to fortify quality and security assurance- quickly, easily and reliably. The test software features a systematic blackbox and negative test methodology uniquely capable of revealing undesired behavior and issues in protocol implementations.

Codenomicon teams its Protocol Modeling Engine and Attack Simulation Engine with protocol support that covers network, wireless, and digital media. Thousands of pre-built, highly targeted, and well-documented test cases allow users to see results as soon as the platform is connected to the target system – accelerating time-to-value. Codenomicon's test tools benefit all end-users in the software, networking, service provider and defense industry. Developers can cut development and maintenance costs by catching bugs early in the software development lifecycle, while operators can test and compare software from different vendors and fix any outstanding bugs. Enterprises and independent labs can test the robustness of software to provide insights for purchase decisions or risk analysis.

Among DEFENSICS 3.0's most important business-related benefits is its ability to enable vendors, carriers, and enterprises to identify and fix security flaws and quality issues pre-emptively. This process, which is performed via intelligent negative testing and maintained RFC coverage, ensures that such flaws are eliminated before they can be discovered by third parties. Also, the inclusion of automated, ready-made tests removes the need for end users to learn new frameworks or to design tests from scratch. DEFENSICS 3.0 comes equipped with thousands of pre-defined, fully configurable test cases that are optimized to efficiently discover irregular responses, slower system reaction, or terminated processes or system crashes. By knowing only the test target protocol interfaces, DEFENSICS 3.0 users can readily start testing and experiencing immediate results. The most recent platform version also offers shorter test cycles than its predecessors. Furthermore, identified flaws are repeatable and traceable, and tests can be reused for regression purposes. Users have fully-integrated documentation, the exact test case construct, and input context to determine the main cause of any identified defects.

DEFENSICS 3.0, which is a highly portable and scalable solution, allows for easy integration to satisfy end user needs. The platform runs on various operating systems and nominal hardware, including laptops. This software-only solution provides engineering and security professionals the flexibility to immediately test any system or device in field or lab settings. The software supports remote users, multiple sites, multiple protocols, external audits, and third-party license management systems. By making the system accessible to different teams and users, organizations can increase usage and optimize resources while reducing expert staff utilization, as well as extra travel and preparation costs. Much like with DEFENSICS 2.0, the current platform version covers many different interfaces and formats, enabling the testing of systems from link-level communications all the way up to the application protocol. However, DEFENSICS 3.0 is able to cover a variety of new types of interfaces and usage scenarios, due to advances in testing technology.

The Frost & Sullivan Award for Product of the Year is presented each year to the company that has demonstrated excellence in new product development and launch within its industry. The recipient company has shown innovation by launching a broad line of emerging products and technologies.



For more information on the Frost & Sullivan Security Test Product of the Year 2008 award, see:  
<http://www.codenomicon.com/resources/whitepapers/2008-product-of-the-year.shtml>





[www.sqe.com/agiledevpractices](http://www.sqe.com/agiledevpractices)

**November 10-13, 2008  
Orlando, FL  
Shingle Creek Resort**

**Meet agile experts first hand as they share their insight and wisdom about the best practices and techniques in the agile industry.**

**For additional information visit:**

**[www.sqe.com/agiledevpractices](http://www.sqe.com/agiledevpractices)**

**Great Speakers.  
Great Content.  
Great Location.**

As agile software development moves beyond the early-adopter stage and becomes accepted and used throughout the industry, we at Software Quality Engineering want to be a part of this revolution. Whether you're investigating agile concepts or committed to agile practices in your organization, this conference has valuable information for you.

## **Covering These Hot Agile Topics**

Agile Methods and Processes  
Transitioning to Agile  
Agile Leadership Principles  
Managing Agile Projects with SCRUM  
Estimating in Agile Projects  
Lean Software Development  
Organizational Models for Agile  
Agile Release Planning  
User Stories and Use Cases  
Agile Design Approaches  
Pair Programming  
Agile Teams  
Measuring Agility  
Continuous Integration  
Test Driven Development  
Tester Roles in Agile  
Distributed Agile Development  
Collaboration Methods  
Personal Skills Development  
Behavior Driven Development  
Becoming a Change Agent  
Scaling Agile Projects  
Hiring and Developing Agile Teams  
And much more...

**To Register Call 888.268.8770  
or 904.278.0524 or Visit  
[www.sqe.com/agiledevpractices](http://www.sqe.com/agiledevpractices)**

# Know Where Your Wheels Are

by Michael Bolton

“Run over that Coke can up there with your right front tire.” Thirty years on, I still hear my dad saying that. *Bump*—but only one—as my right front tire ran over the can. “Good,” he said. “You always want to know where your wheels are.”

Those words came back to me the other day in a discussion about how to train testers. An acquaintance said she wanted to teach new testers by telling them to use scripted approaches. That didn’t seem to me like a way to build skill; following a script doesn’t necessarily engage the cognitive processes of the learner. I thought of the skills associated with driving a car, what good teachers had taught me, and how I had learned.

*There’s way more to it than just knowing the written rules.* When you drive a car, the rules of the road are important, but we’re not always being watched and other people aren’t always following the rules. The laws that nature enforces are more important from moment to moment than the rules of the road: A car travelling twice as fast as yours takes four times the distance to stop, and if there’s another car coming straight at you, you’ll have to deal with it right away, whether the other driver is right or wrong. Good training and practice equip us with skills to deal with unexpected, time-critical situations.

*Books and classes can help.* When it was time for me to learn to drive, I got a copy of the driver’s handbook and my parents sent me to driver education classes. The classes taught me heuristics: Aim high while steering—don’t just look at the hood of the car, instead look several lengths ahead; cooperate with other drivers and make sure they see you; don’t follow too closely and leave enough room to get yourself out of trouble; get the big picture—don’t just look straight ahead, check your mirrors frequently, and note what’s going



ISTOCKPHOTO

on around you; move smoothly—when you’re off course, use light corrections in steering and speed so that you address the problem without going out of control. Hearing about those heuristics in advance, and recognizing new ones as I went on, accelerated my learning.

*Preparatory procedures are valuable.* Both the driving instructor and my dad emphasized several checklist items to get ready to drive. They told me to get into the habit of approaching the car from different directions and performing a quick inspection of the tires and the outside of the vehicle. Then they focused on getting me comfortable in my seat, checking that my seatbelt was fastened, adjusting the middle and side mirrors, and making sure the car was well ventilated. Just before turning the key, they had me check the gearshift lever and the parking brake, and immediately afterward, they had me survey the instrument panel—gas, oil pressure, warning lights. Dad taught me to back up close to a wall to check my headlights and taillights by looking at their reflections. No one gave me a sequence in which to check each detail, but they had me narrate what I was doing until they were sure I had

absorbed the checklist. Some things are too important to be left to a script that someone else wrote.

*Practice, coaching, and mentoring are essential.* The classroom work had been valuable; it helped give background and structure to my learning. But the real lessons happened as I drove with a coach—the driving instructor, one of my parents, or another experienced driver—beside me. We started in safe places, such as shopping mall parking lots, dead-end streets, and quiet neighborhoods. My mentors gave me easy tasks to perform, instructing me explicitly, observing me carefully, and telling me what to look and listen for based on what I was actually doing in the moment. They directed my attention to what the car was trying to tell me and praised me for good work. The best and fastest learning happened with me literally in the driver’s seat.

*Good mentors raise the bar.* As I developed my skills, my coaches put me into increasingly more challenging situations—busier roads, worse weather, freeways, and so forth. My dad peppered me with questions every few moments to develop my cognitive skills and to make sure I could juggle a bunch of tasks

simultaneously. “What color is the car behind us? How many people were in that police car coming the other way? What’s the name of the street on the right? You have to process a lot of information as you’re driving.” These exercises helped me shift my focus rapidly between the big picture and fine details. I was put in situations where it was safe to make harmless mistakes, and then my mentors suggested alternative approaches and had me practice them. When I did something potentially dangerous, they did their best not to overreact. My own reaction to a scary situation was usually more than enough to make the learning stick.

*Freedom and responsibility are closely linked.* After I got my license, my parents let me use the car on my own—short trips at first and then longer excursions. They gave me freedom but they also required responsibility. I had to be able to tell them where I was going, how I planned to get there, and when I expected to be back. If they saw anything risky in my plans, they would make suggestions like taking another route or avoiding rush hour. When I did get back, they asked me specific but reasonable questions about things that I had done and seen. My parents guided me but they didn’t try to control me. They realized skill doesn’t really develop when someone else takes control over your process. As I demonstrated responsibility, they granted me more freedom, but as long as I was using one of their cars, I had to remain accountable.


*Our task teaches us.* One of the biggest sources of feedback about my driving was the car itself. It engaged my senses. The visual information from the speedometer and the tachometer reinforced the audible information that I got from the sound of the engine. If I pushed on the brakes too suddenly, the car would pitch forward. If I turned too hard, it would rock to one side. Very occasionally, a more serious mistake would trigger the sound of a horn or a dirty look from other drivers. When my Dad was driving one day and noticed that a wheel had gone out of balance, he pulled over to let me drive so I could feel the vibration of the steering wheel. He also reminded me to keep my eyes moving

around the car, to listen for new sounds, and to be aware of new smells. His goal was to remind me that there were often clues available when there was a problem with my car or with my driving.

Learning for drivers, just as for testers, is ongoing, and I still make mistakes every now and then. A few years ago in Santa Barbara, I tried to drive through a flooded underpass where the water was a good deal deeper than I had expected. Suffice it to say that I didn’t die, and I won’t make that mistake again.

When we test, we need to know the written rules for our product—the documented requirements—but we also need to recognize, deal with, and report important things that the written stuff doesn’t cover. Books and classroom learning can prepare us, but the learning is richer, deeper, and more adaptable when it’s heuristically based and experiential. The software that we’re testing and the people around us often provide us with valuable feedback about what to test or what might be wrong with our testing process, and even experienced people make mistakes every once in a while. Coaching, mentoring, and, above all, practice are vital to developing expertise, and as we gain skill, we can take on and account for increasingly challenging work.

There are still some bad drivers out there. Sometimes the problem is that they’re completely unsupervised; more often, they simply haven’t developed sufficient skill. Wouldn’t things be even worse if the unskilled people were driving under the control of someone else’s script? {end}



**How do you learn the complex cognitive skills of testing? Of other disciplines?**

▼

Follow the link on the [StickyMinds.com](http://StickyMinds.com) homepage to join the conversation.



## ARE YOU IN SEARCH OF QUALITY? SO ARE WE.

Quality Assurance Analysts  
Quality Assurance  
Software Engineers

Apply online at  
[www.blackbaud.com/QAcareers](http://www.blackbaud.com/QAcareers)  
or contact Stephanie McDonald,  
senior technical recruiter, at  
**843.654.3547** or  
[stephanie.mcdonald@blackbaud.com](mailto:stephanie.mcdonald@blackbaud.com).

RELOCATION ASSISTANCE  
IS AVAILABLE

## Quality of Work

Work at the high-tech industry leader in software and services for nonprofits.

Use your talents to make a difference for our customers, who make a difference in the world.

Be an integral part of our product development team and help bring the best possible products to our customers.

## Quality of Life

Enjoy the history, culture, and charm of Charleston, South Carolina. Receive great benefits and a competitive salary.

Get started today!

**Blackbaud®**



# Advice for the New Leader

by Michele Sliger

It was time to go. Jim had packed up his things and was surprised to see how many boxes he'd filled. One little cubicle could really accumulate a lot of stuff after two years! It was a good thing he was only moving across the quad.

Jim was sad to leave, but he was looking forward to his new position as the manager of the R&D product group. It was a newly created department, and there were going to be a lot of challenges ahead. Jim was excited about having the autonomy to build great teams and foster innovation—something he'd been a part of in his current position as a team leader under Alicia, the manager of the Gemini product line.

Jim thought about the path he'd traveled and just how he had acquired the skills necessary to be accepted for this new position. It was more than his solid knowledge of the existing product lines and more than his knowledge of the industry as a whole. Jim realized that he truly owed a debt of gratitude to Alicia, who had helped him become a better team leader. Alicia had mentored and guided him through good times and bad, always asking just the right questions to help him look at things in a different way. She never told him what to do, but gave him the opportunity to experiment and learn about what worked and what didn't. She never berated him for his failures, but instead would smile and ask what he'd learned as a result and what he would do differently next time.

Wondering what else he could learn, Jim hopped up and headed for Alicia's office, hoping to catch her before she left for the day. She was a staunch believer in working at a sustainable pace, which meant that most folks in her department were out the door by 5:00 p.m. It was just after 5:00, so Jim hurried down the hall.

He was in luck—Alicia was still in her office. "Hi, Jim! Did you stop to say goodbye? I figured you'd be here until eight tonight, trying to pack up that rat's nest of a cubicle!"



"Ha ha. I'm all packed and ready. And you know this isn't goodbye. Heck, you'll probably see more of me now that we're peers and have to go to all those management meetings together. I actually stopped in for one last piece of advice. Do you have a minute, or were you heading out the door?"

"I'm done with my work for today, but I'm not in any hurry to leave. What can I help you with?" Alicia sat down in one of the chairs in front of her desk and invited Jim to do the same.

"First I wanted to thank you for helping me grow as a leader over these past two years. I think that's why I got this new position. Now you can help me by telling me: How on earth did you do it?"

Alicia's eyes went wide and she burst into laughter. "You're kidding, right? Jim, you were there, too, you know! And I'm pretty sure—or at least I *was*—that you are an observant fellow."

Jim smiled and nodded. "Yeah, I know. And believe me, I got the basics." Jim started rattling off all the things he'd learned from Alicia. "Focus on the goal and the intent of the vision. Never tell, but instead ask questions, A LOT of questions. Listen, really listen, without trying to think faster than others and solve their problems before they have a chance to. Be patient. Remind people that failure is not a bad thing—it's an opportunity to learn and get better.

Don't forget that this is just a job and what's important is often outside these walls: friends, family, quality of life." Jim paused. "Did I forget anything?"

"That sounds good," Alicia responded. "So, what's your real question?"

Jim paused and thought. Alicia waited patiently and did not try to fill the silence with answers. Jim realized this was another good technique she'd used with him before, helping him to look past the surface question to pinpoint the real concern. He was ready to try again.

"You don't solve problems, you allow us to. Isn't that hard? I'm worried that I will want to get in there and help create solutions with the rest of the team. How do I take a step back? How do I know *when* to step back?"

Now it was Alicia's turn to ponder. "I think in the beginning, you may be more involved than is typical because it's a new department with a new vision and a new staff. But you're going to be so busy, Jim, that you won't have time to be with the teams much. You have to build this department, and there will be a lot of administration and organization problems to solve. You have to focus on those issues so your team can focus on the work at hand. I never assume that I can do the job better than the folks I've hired. So it's back to asking the right question, which in this case is asking the teams, 'How can I help?'"

“That’s good advice. I think I’ll make a sign for my desk that reminds me. Something like ‘Remember what YOUR job is and don’t try to do theirs!’ And you’re right, I will have so much to do, just in setting the direction and vision for the teams.” Jim paused, thinking that being a visionary and a guide was going to keep him busy indeed. “I have one more question, then I’ll leave you alone so you can get home. How do you know what the right questions are to ask?”

“Oh, that’s easy! I don’t!” Alicia smiled and continued, “It’s not like I have a list of questions that are part of a checklist. It’s just genuine curiosity. I know that I’ve hired the best people to solve these problems and I trust them to do their jobs. I don’t have to worry about the answers—they do. So that frees me up mentally to just listen. And I truly want to know what they are grappling with and how I can help.

“I think that the most important thing is to care about your staff. Really care about them as people and never, ever, treat them like ‘assets.’ They’ll know, Jim, if you try to fake that. So if your team members aren’t coming and talking to you, then get up and go and talk to them. Just check in, find out how they are doing as individuals. Build relationships so they can feel comfortable coming to you with their concerns. And I don’t think you have to worry. I’ve seen you with your team here, and I’ve heard them all speak highly of you.”

“Thanks, Alicia. You make it sound so easy! I appreciate your insights. I hope that one day I’ll be in the same position, with one of my staff coming to thank me for helping him grow and take on more responsibility.” Jim smiled, stood up, and extended his hand to Alicia.

Alicia shook his hand. “That’s the best part, Jim! That’s when you’ll know that you’re doing your job right and helping others in the process!” {end}



**What challenges have you encountered when “stepping back” as a leader?**

Follow the link on the [StickyMinds.com](http://StickyMinds.com) homepage to join the conversation.

## STORY LINES

- **Let your team members do what they were hired to do! Don’t try to solve problems for them.**
- **Be a guide for others by listening, asking questions, and reminding them of the vision, intent, or goal.**
- **Be prepared to support your knowledge workers when they fail and help them see that this is just another opportunity to learn and grow and improve.**

# — Are we there yet?

Well, it depends who you ask...



With **TargetProcess**  
you will always get  
the most accurate answers  
on the state of your:

- Projects
- Resources
- Customer needs



**Full support of Scrum, XP and other Agile processes**






# **AGILE MODEL-DRIVEN DEVELOPMENT**

**SCALING AGILE TO MEET  
THE NEEDS OF  
REAL-WORLD PROJECTS**

**BY SCOTT W. AMBLER  
AND CELSO GONZALEZ**







**D**espite what you may have heard, modeling is an important part of agile software development. Sadly, it doesn't get a lot of attention, even though it's a fundamental technique for scaling agile to meet the needs of the real-world situations in which project teams regularly find themselves. It pays to think before you act, and modeling enables you to think through the critical, high-level issues that other techniques struggle to address.

This article overviews agile model-driven development (AMDD), showing how modeling fits into the overall agile software development lifecycle. We also share industry statistics regarding modeling on agile projects and discuss three tooling strategies that you may want to consider. We walk through an example of modeling on the development of a business application and end with a discussion of the trade-offs of agile modeling.



## AGILE MODEL-DRIVEN DEVELOPMENT

The Agile Modeling (AM) method, introduced in late 2001, describes a collection of principles and practices for effective modeling and documentation on agile projects [1]. The AMDD lifecycle was introduced in 2004 [2], and newer updates have been published online since then. Figure 1 depicts the current AMDD lifecycle, showing that there are three fundamental modeling points throughout the development process:

- *At the beginning of the project.* Someone is always going to ask you how much you think the project will cost, how long you think it will take, what you think you're going to build, and how you think you're going to do it. To answer these questions you're going to need to do some initial requirements and architectural envisioning. This is done during the first iteration of the project, sometimes referred to as iteration 0 in methods without specific project phases, as the warm up iteration in the Eclipse Way, and as the Inception phase in the Unified Process. At this point in the project the goal is to understand the scope and technical strategy, not to document them in detail.

- *At the beginning of each iteration.* Agile teams are self-organizing by nature, with detailed project planning done at the beginning of each iteration. To accurately estimate the effort needed to implement a requirement, the team often will discuss its strategy for doing so, frequently using an inclusive modeling tool such as a whiteboard or paper. (Yes, sketching on whiteboards, writing user stories on index cards, or using sticky notes to lay out a screen or report are all modeling activities.)
- *Just in time (JIT) throughout the iteration.* Developers, even when pair programming, will “model storm” on a JIT basis to think through an issue pertaining to what they're about to build. These model-storming efforts are often short—usually fewer than ten minutes in length—and are focused on a specific issue.

Figure 1 also shows how AMDD and test-driven development (TDD) work together. TDD is a detailed specification technique and a confirmatory testing technique and is proving to be incredibly effective. But TDD isn't sufficient for all of your specification and testing needs; you also need to think through high-level issues and test beyond the confirmatory level. To scale TDD you need something like AMDD to think through the high- and medium-level requirements and design issues and independent, investigative testing efforts [3].



## AGILISTS MODEL, BUT THEY DON'T TALK ABOUT IT

There seems to be a fear in the agile community that if we use terms such as “model” or “document” that suddenly the “evil bureaucrats” will dig their claws into our projects and force us to write detailed, big requirements specifications or to take a big-design-up-front approach. This fear isn't completely unfounded, as there are a lot of traditional modelers out there who still believe in those concepts—regardless of the mounting evidence [4, 5] against those strategies.

The strange thing is that agilists are, in fact, modeling on a regular basis,

even though they're not directly talking about it. For example, a lot of agilists like to talk about treating requirements like a prioritized stack that is created with initial requirements modeling at the beginning of the project. A recent survey [6] asked several questions pertaining to modeling in an effort to discover what is actually going on out there. The survey found that 77.7 percent of agile teams are doing initial requirements modeling and 77.2 percent do initial architecture modeling. It also found that 92.7 percent do whiteboard modeling, 65.9 percent do paper-based modeling (index cards, sticky notes, etc.), and 47 percent

are using some sort of drawing tool or modeling tool.



## USE THE SIMPLEST TOOLS

One of the practices of AM is to *use the simplest tools*. Figure 2 shows three modeling styles from the point of view of tool usage—any given project team will use one or more of these approaches throughout its lifecycle. A common theme is that regardless of your approach, you are likely at some point to use inclusive modeling tools such as whiteboards and paper to facilitate communication with both your stakeholders

and with other team members. There will always be some sort of “skilled transformation” of the information identified by your inclusive modeling efforts to capture it in your tools, which are used to create working software. Furthermore, depending on tool availability and the skill sets of your team members, you might use more complex, software-based modeling tools.



## AGILE MODELING IN PRACTICE

In this example, we chose to use the “Close to the Metal” strategy, as it best illustrates the advantages of a modeling tool without requiring as much customization as the “Agile MDA” strategy. We are developing an auction application for a company that will allow its employees, who are spread out around the world, to buy logo merchandise from its intranet as well as from the Internet. The project team is dispersed across several locations. The main team is in Toronto, including the team lead, three Web developers, and the product owner. Four developers with auction expertise are located in Bangalore, and business stakeholders are located throughout the world.

We start our modeling with some initial requirements envisioning done in collaboration with the product owner and the business stakeholders. One of the most efficient approaches to describe requirements is to use a scenario-based approach with user stories, use cases, or usage scenarios. We'll be creating use cases.

From our working sessions we drew the following initial use case diagram of figure 3 (simplified for this article), where the “Place a Bid,” “Bill Buyer,” and “Ship Item” use cases have been identified as the most critical. This diagram and the underlying use cases will evolve throughout the project. Remember, the goal is to understand the scope at this point in the project; detailing the requirements will come out during the project iterations. While identifying the initial set of use cases, we also identified a starting set of non-functional, quality-of-service requirements, such as security (the application will be accessed from the Internet) and performance (the employees are all around the world).

The architecture envisioning oc-

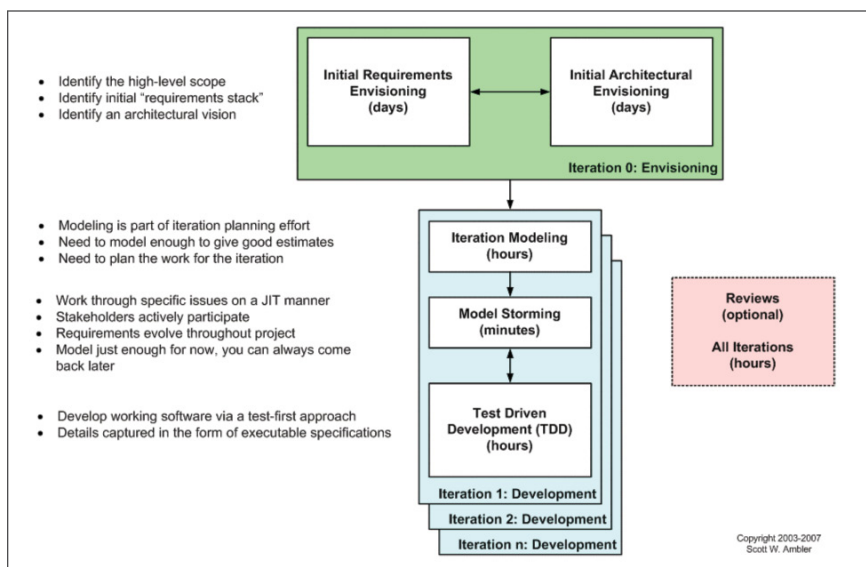


Figure 1: The lifecycle of agile model-driven development

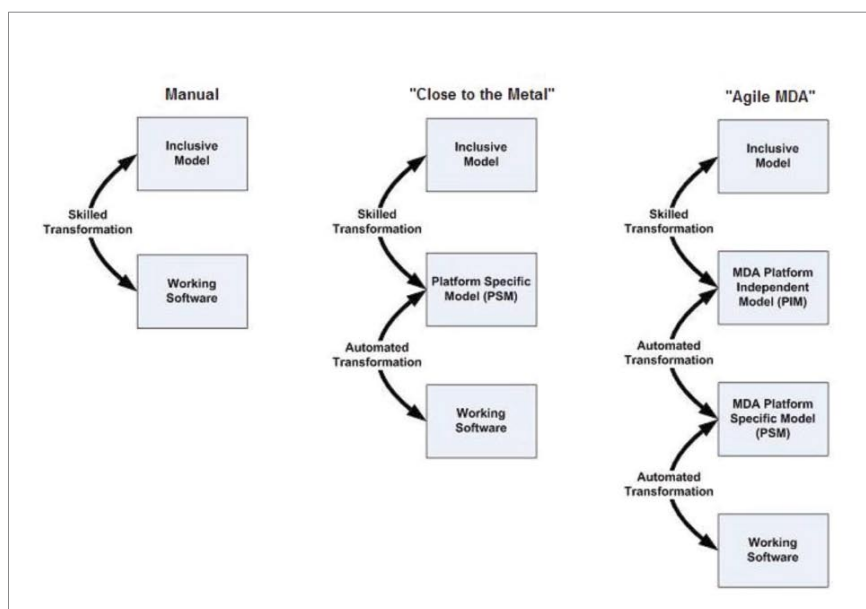


Figure 2: Three tool strategies for AMDD

curs in parallel to the requirements envisioning—an architecture is never elaborated in a vacuum, but instead addresses both functional and non-functional requirements. Some agile teams like to use whiteboards for this kind of activity, but a complementary strategy is to use a modeling tool on a laptop combined with a projector, so everybody can follow the progress in real time. We have found that using a software-based modeling tool—in this case we’re using Rational Software Architect (RSA) for the visual models depicted in this article—instead of a whiteboard has some advantages. First, it avoids the overhead of having someone copy the whiteboard at the end of the session—this is particularly important if you don’t have dedicated whiteboard space or if for security reasons you can’t leave your architecture sketches up. Second, if you require technical documentation, perhaps due to regulatory compliance issues, you just print the diagram or generate a report from your working model. Third, the software-based tool facilitates the integration of remote participants into the design sessions, as you can share your model in real time over the network, which is critical for our developers in Bangalore.

Identifying your technical strategy is an important part of architecture envisioning, and a simple but critical view is a technology diagram, as seen in figure 4, which provides an overview of the architecture layers and the technologies that will be used. Another diagram that we develop describes the deployment architecture we will be using. As on the majority of projects we have some technology constraints to which we need to adapt. Our main constraint is to reuse two existing Derby database schemas: one developed to store employee information (HR) and the other developed to manage the items’ stock availability (Inventory), as shown in the deployment architecture diagram in figure 5.

For business applications, initial high-level domain modeling often is a valuable part of your envisioning activities because it identifies the major business entities and their relationships. Our approach consists of looking at the existing model (if there is one) to reuse any existing domain objects and update the model while implementing the use cases

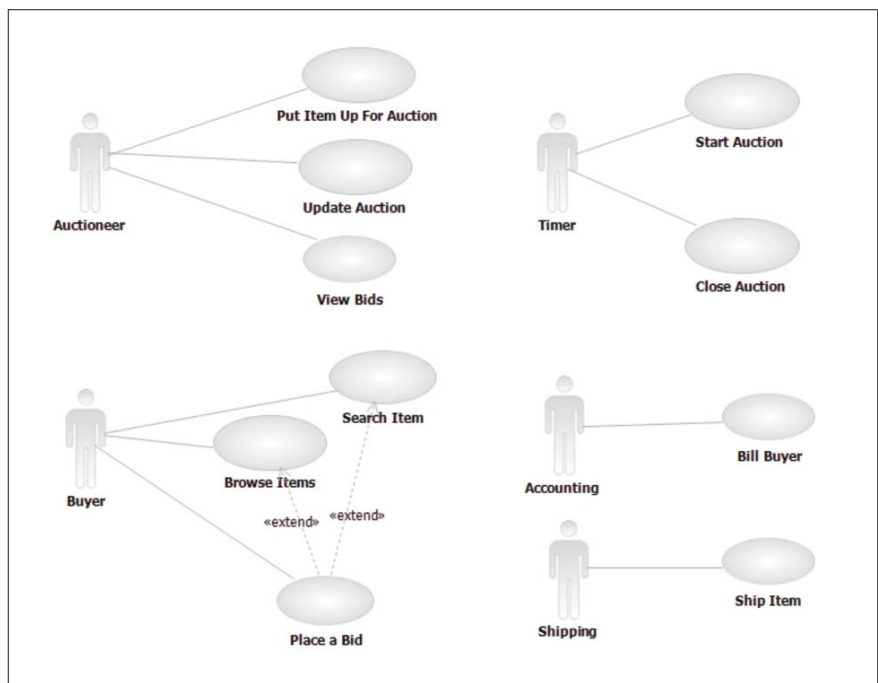


Figure 3: Initial use cases

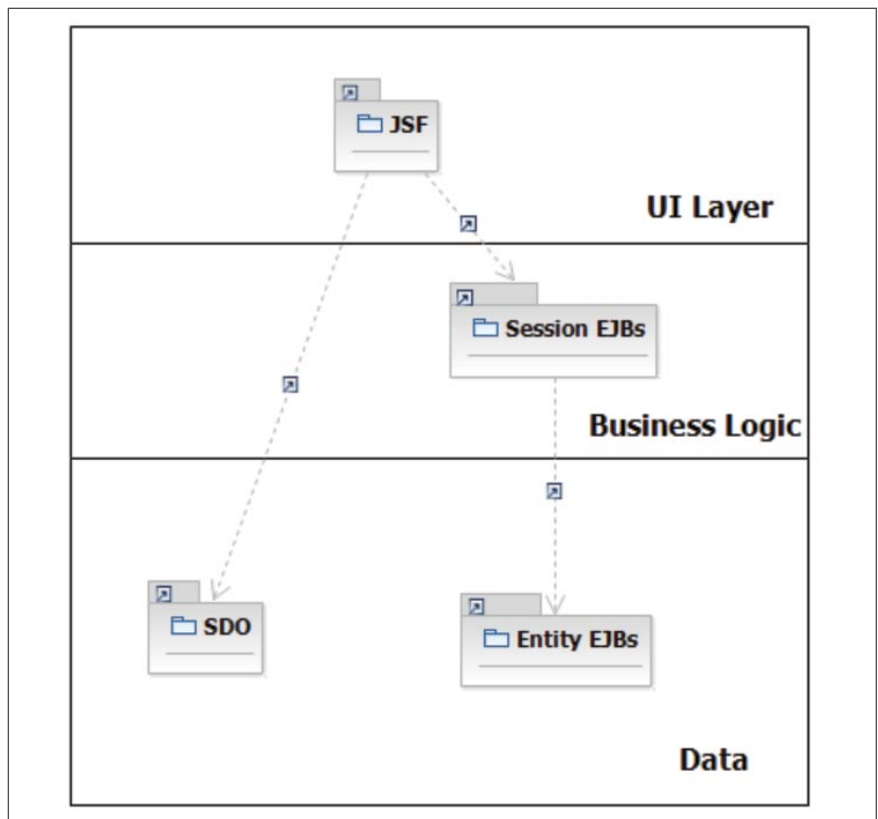


Figure 4: Auction technology diagram

rather than doing too much up-front domain modeling. This reduces overall development cost and time to market, and improves data quality by avoiding the creation of yet another silo database. For this project there was no existing domain model, but we can start

one by reverse engineering the HR and Inventory databases. These databases are the source of record for that data, so they’re where we need to start to keep the overall data quality within our organization as high as possible. You can either look at this model using Entity/Re-



lationship notation, as shown in figure 6, or as a UML domain model, as shown in figure 7, depending on the skills present on your team.

All these models provide us with critical views for our initial architecture and, as everything else on an agile project, they can be modified if necessary to adapt to changes discovered along the project. The advantage of doing this initial modeling is that it provides a starting point from which to work, reducing the chance of major refactoring later in the project. A small investment in up-front thinking helps scale agile techniques for the complex situations in which many project teams find themselves.

We can now start our first iteration and work on the “Place a Bid” and “Bill Buyer” use cases. Because some of the team members and stakeholders are remote, the product owner will start detailing the use cases by modeling a bit ahead of the team [7]. Figure 8 shows the level of detail to which the product owner would likely go, just enough so the developers understand the gist of the requirement and can then ask him detailed questions during the iteration. The more detailed requirement specification would be captured in the form of customer tests as part of our TDD activities. Remember, with AMDD you explore high-level concepts and with TDD you specify the details.

Let’s focus on implementing “Place a Bid,” so we’ll need to concern ourselves only with that part of the model for now. Figure 9 shows the new Bid object as well as its relationships to existing objects.

The **Bid** class contains the current bid, the bid increment, the new resulting bid, and the currency used along with a reference to the product that was purchased and the employee who purchased it. While creating the bid, the **price** attribute of the **Item** class would be updated to store the new bid amount. We can use forward-engineering automation to generate an updated physical data model as shown in figure 10, and, comparing it to the original one, generate an SQL script to update the database, as shown in figure 11.

The initial reaction that many agilists have to the model in figure 10 is that we’ve over-engineered the system, particularly when you consider the detail sur-

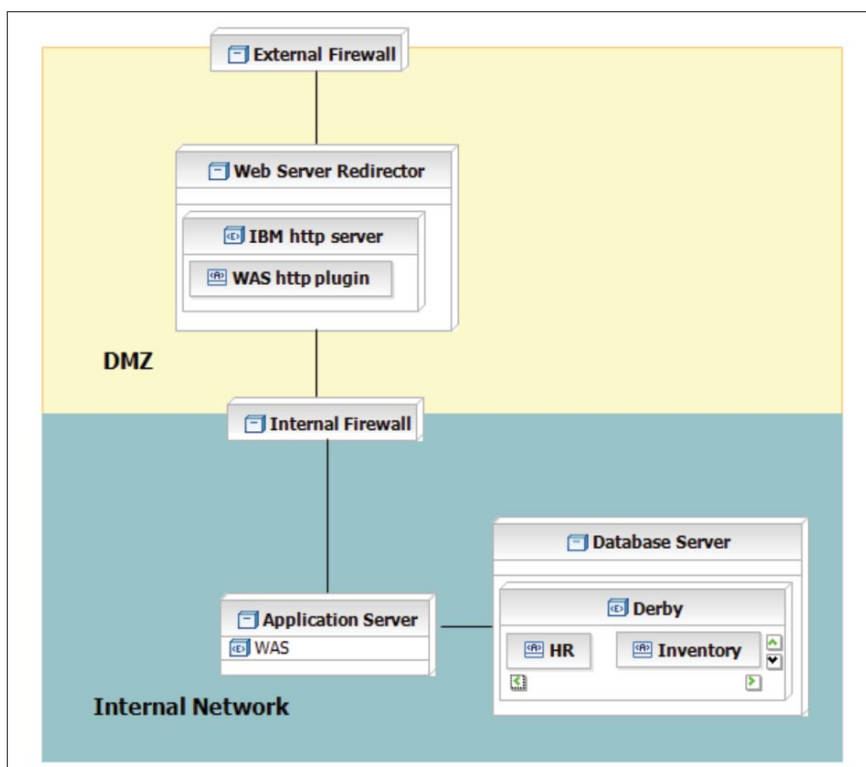


Figure 5: Auction deployment diagram

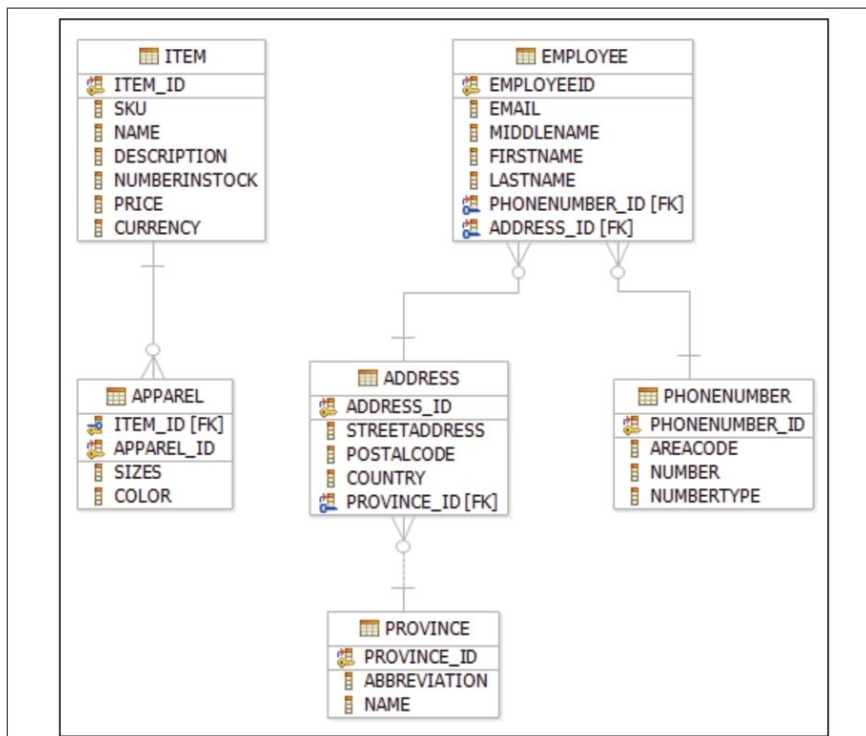


Figure 6: Initial data model

rounding phone numbers and addresses. *Au contraire, mon ami!* Remember, our system takes advantage of the existing HR and Inventory databases, so the simplest design is to leverage the existing assets instead of building a silo system from scratch.

Now that we have our data stabilized, let’s have a look at the other layers. Displaying the items only takes a simple SDO call to retrieve the items from the database and display them in a JSF page. This is something that our IDE JSF wizard will generate for us, so we

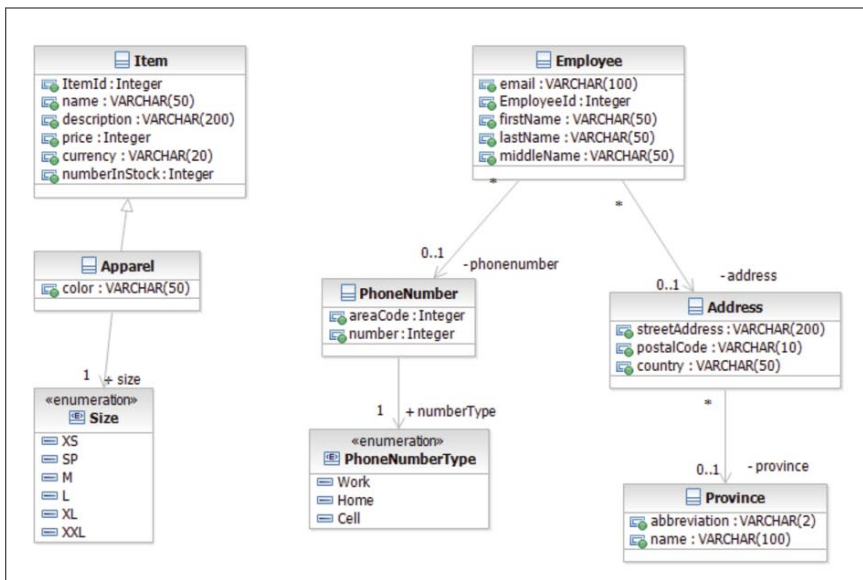


Figure 7: Initial UML domain model

## Use Case: Place a Bid

**Actor:** Employee

**Precondition:** The Employee is logged into the Auction application.

### Main Flow

1. The system displays the available items.
2. The user selects the item he likes to bid on.
3. The system displays the bid form.
4. The user enters the bid increment and submits the bid.
5. The system displays the bid confirmation and the new bid.
6. The use case finishes.

Figure 8: Initial version of the “Place a Bid” use case

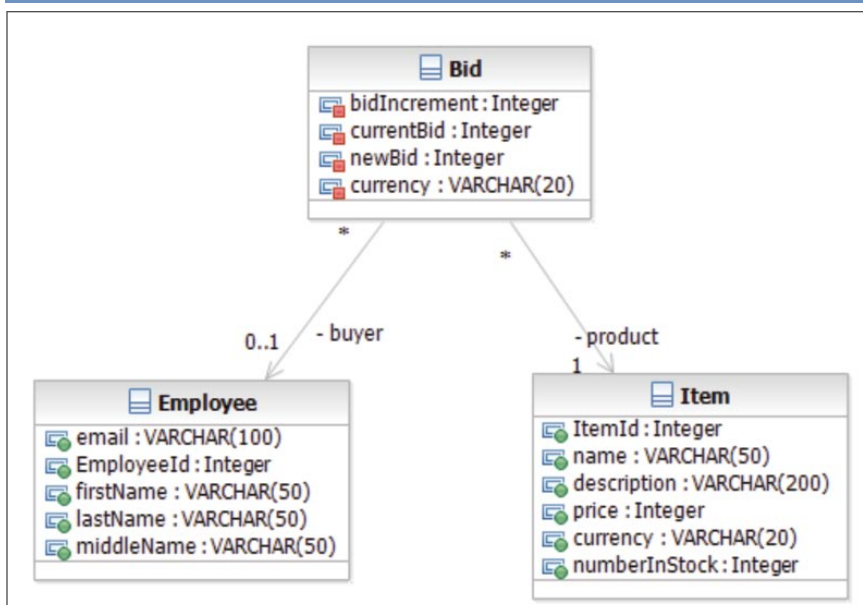


Figure 9: Updated domain model

don't need to model it. Remember, use the simplest tool for the task at hand. On the other side, the bid would be implemented using EJBs, so it would be worthwhile to model it. As for the use cases, just because you decide to model doesn't mean that you need to model every detail. Modeling is a technique to help you design your solution, and you are the only one who knows how much detail you need. This is also true in the context of code generation. The purpose of code generation is not to force you to detail everything, but to leverage the modeling you have done to seed your code and to avoid redoing all this work manually. Then, you can flesh out that generated code via a TDD approach as you saw in figure 1. You will need to decide as a team how much code seeding you wish to do via modeling versus how much hand coding via TDD—different situations require a different mix.

As seen in figure 12, for our application we model two EJBs with their attributes and methods. We will use different stereotypes that will generate the EJB's code during the UML-to-EJB transformation. We also decided to define the attribute types, as well as the `placeBidOnItem` signature, as it is not too much overhead and will provide better generated code. As we said previously, each time we place a bid we need to set the price of the item to the value of the new bid. To do that, we need an entity EJB to access the Item table. We are lucky that such an entity bean has been developed by another project. We just have to drag it into our diagram to be able to create a dependency between the `bidSB` session bean and the visualized `ItemCMP` entity bean. Visualization allows us to work with the code using UML visual diagrams without creating distinct UML elements; you directly access the code through a UML rendering engine. Figure 13 shows the visualization of the `Bid`-related classes.

As soon as the code is seeded from your model, you can start implementing using TDD until you validate through testing that the “Place a Bid” use case is successfully implemented. You then can start a new iteration and select a set of use cases to implement following the same approach. It is also worth mentioning that software-based design tools allow you to reapply a transformation

while preserving the code added to the previously generated classes, allowing this kind of iterative development.



## MODELING GAIN WITHOUT THE BUREAUCRATIC PAIN

There are several advantages to taking an AMDD approach to agile software development:

- *AMDD helps scale agile.* We often find ourselves in situations where the team is distributed geographically and organizationally, the team is large, the application is complex, you must comply with industry regulations, your organization has an IT governance program in place, your system integrates with others, or you're working with not-so-agile people in other groups. As we scale agile approaches to more complex situations, the need for realistic approaches to modeling and documentation becomes more apparent.
- *Documentation support for distributed development.* Capturing requirements on index cards and architectures on whiteboards works really well when you've got a small, co-located team. When your team is distributed, either in space or in time, then there is greater need for written documentation.
- *Models help persist information over the long term.* Someone will have to maintain, operate, and support the system that you're building today. The implication is that you're going to need to write some documentation to aid these people in doing so.
- *AMDD scales TDD.* TDD is great for specifying details in a JIT manner; AMDD provides the techniques to think things through at a high level, thereby scaling the specification part of TDD.
- *Good modeling tools, in skilled hands, increase productivity.* Good modeling tools will automate a lot of the grunt work of software development, thereby raising the abstraction level and your overall productivity levels.

Nothing is perfect and, as you'd ex-

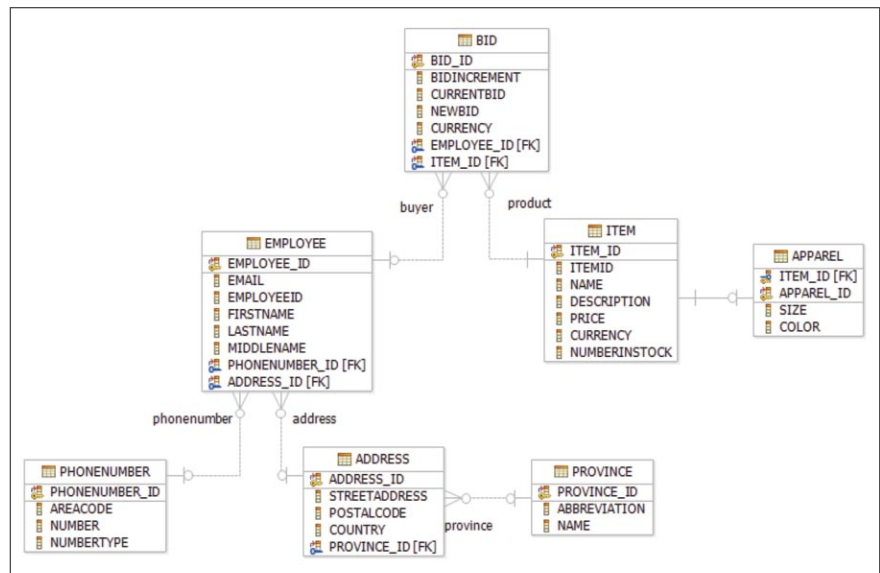


Figure 10: Updated physical data model

```
ALTER TABLE "Auction"."BID" DROP CONSTRAINT "BID_EMPLOYEE_FK";
ALTER TABLE "Auction"."BID" DROP CONSTRAINT "BID_ITEM_FK";
ALTER TABLE "Auction"."BID" DROP CONSTRAINT "BID_PK";
DROP TABLE "Auction"."BID";

CREATE TABLE "Auction"."BID" (
    "BIDINCREMENT" INTEGER ,
    "CURRENTBID" INTEGER ,
    "NEWBID" INTEGER ,
    "CURRENCY" VARCHAR(20) ,
    "BID_ID" INTEGER NOT NULL
        GENERATED BY DEFAULT AS IDENTITY
        (START WITH 1 , INCREMENT BY 1),
    "EMPLOYEE_ID" INTEGER ,
    "ITEM_ID" INTEGER NOT NULL
);

ALTER TABLE "Auction"."BID" ADD CONSTRAINT "BID_PK"
    PRIMARY KEY (BID_ID);

ALTER TABLE "Auction"."BID" ADD CONSTRAINT "BID_EMPLOYEE_FK"
    FOREIGN KEY (EMPLOYEE_ID)
    REFERENCES "Auction"."EMPLOYEE" (EMPLOYEE_ID);

ALTER TABLE "Auction"."BID" ADD CONSTRAINT "BID_ITEM_FK"
    FOREIGN KEY (ITEM_ID)
    REFERENCES "Auction"."ITEM" (ITEM_ID);
```

Figure 11: Database update script

pect, there are several disadvantages to AMDD:

- *You need to adopt modeling tools.* We've both worked in organizations where it was difficult to get a whiteboard installed in a meeting room or a team work area, let alone get the funding for

a tool.

- *Developers need modeling skills.* Although whiteboard sketching and paper-based modeling is a great start, there are significant benefits to using software-based modeling tools. Simply purchasing and installing such tools



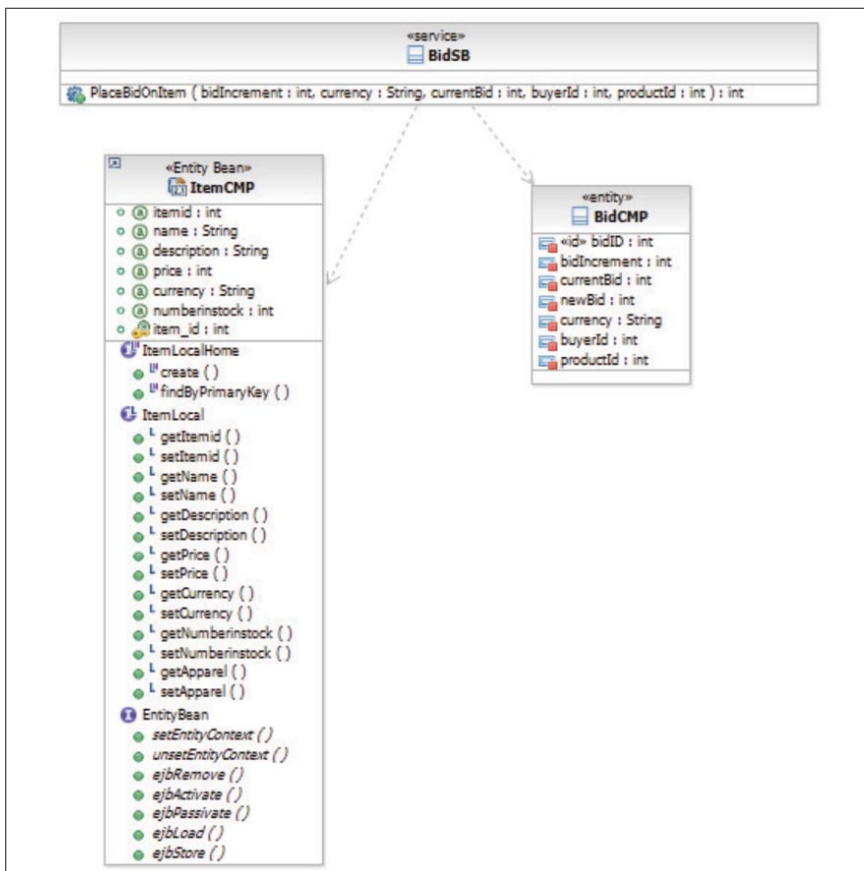


Figure 12: Bid EJBs model

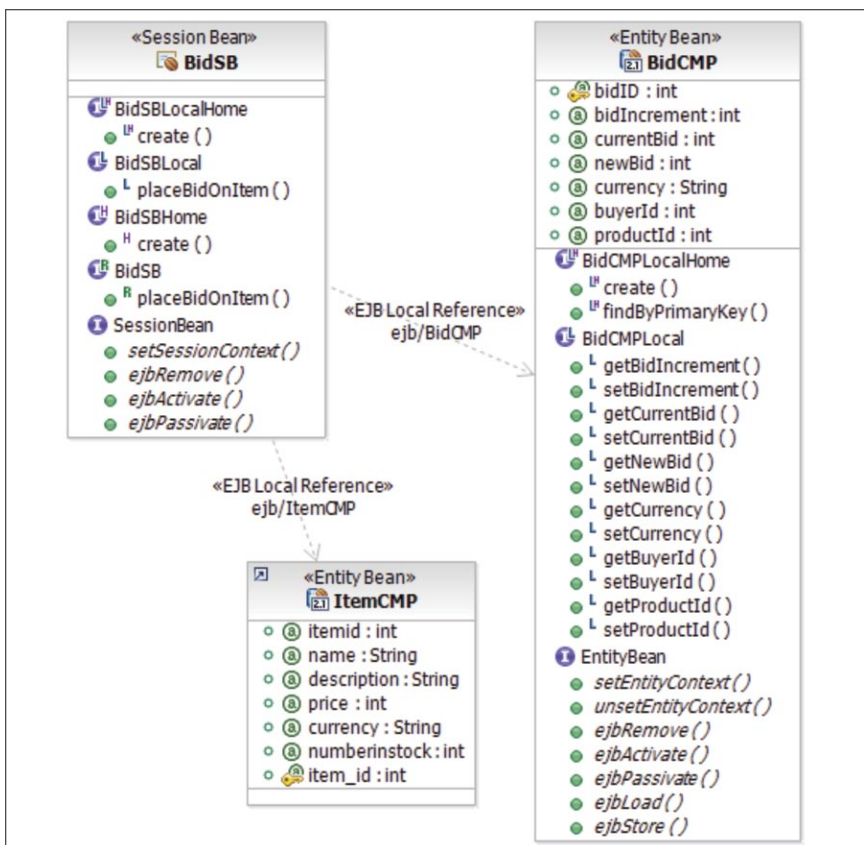


Figure 13: Generated EJB classes visualization

isn't sufficient—you also will need to invest in training and mentoring.

- *Good modeling tools are expensive.* But, when used properly by skilled people, they provide a lot of benefit.
- *Many modelers struggle with agile development.* There are people out there with great software modeling skills, but they're often convinced that they need to model everything up front in detail before development can begin. These people can be valuable members of agile teams, but they must be willing to adopt agile ways of working, to share their skills with others, and to pick up new skills along the way.

Modeling is an important part of agile software development. But it is possible to gain the benefits of modeling without the pain of the needless documentation that often is associated with modeling on traditional projects. We've shown you how modeling fits into the agile lifecycle in a lean and streamlined manner, which improves overall team productivity. Don't let the rhetoric of some agile developers dissuade you from taking advantage of agile modeling techniques and good tooling. {end}

#### REFERENCES:

- 1] Ambler, S.W. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: Wiley Publishing, 2002.
- 2] Ambler, S.W. *The Object Primer: Agile Model-Driven Development with UML 2.0*. New York: Cambridge University Press, 2004.
- 3] Ambler, S.W. *Scaling Test-Driven Development (TDD)*. Dr. Dobb's Journal, February 2008. [www.ddj.com/architect/205207998](http://www.ddj.com/architect/205207998)
- 4] Ambler, S.W. *Answering the "Where is the Proof that Agile Works" Question*. 2007. [www.agilemodeling.com/essays/proof.htm](http://www.agilemodeling.com/essays/proof.htm)
- 5] Ambler, S.W. *Examining the Big Requirements Up Front (BRUF) Approach*. 2007. [www.agilemodeling.com/essays/examiningBRUF.htm](http://www.agilemodeling.com/essays/examiningBRUF.htm)
- 6] Dr. Dobb's Journal. *Agile Adoption Rate Survey: March 2007*. [www.ambysoft.com/surveys/agileMarch2007.html](http://www.ambysoft.com/surveys/agileMarch2007.html)
- 7] Ambler, S.W. *The "Model a Bit Ahead" Pattern*. 2005. [www.agilemodeling.com/essays/modelAhead.htm](http://www.agilemodeling.com/essays/modelAhead.htm)

The

*Myth* of

Risk Manag

Insure Your Project Against the Unknown and the Unknowable  
by Pete McBreen





ement

**R**isk management is an illusion. Risks cannot be managed in software development projects because they are either unmentionable, uncontrollable, unquantifiable, or unknown. Projects run into problems because they are hit by something that everyone knew about but was unable to talk about, something totally unexpected, something that was rated as highly unlikely, or something that was made worse by the “mitigation strategy.”

If risks were truly manageable, we would see far fewer projects that end up stressed—or rather, with team members who are stressed because they are faced with a project that is not going anywhere near as planned. In practice, however, most projects end up stressed in one way or another, so a project that finishes on time, on budget, and with all of the initially planned features is a rarity (unless the numbers are adjusted as the project proceeds).

### *Project Failures Are Not Simple*

We would all like project life to be simple—to have a magic fix that would make project failures go away—but there is no such thing. Indeed, it seems that managers are trained to assume that project failures are due to a single specific cause—misunderstanding of what the users need, key items delivered late, defect counts rising as the project nears its release date, or loss of key people.

Reality suggests otherwise: Projects rarely fail because of one single identifiable cause. A classic example of this is the Challenger disaster. Yes, the “ultimate” cause made for a great sound bite on TV, with Dr. Richard Feynman showing that the O-rings were not elastic at low temperatures, but it was not as simple as that. The minority report [1] written by Feynman showed that there were many other things that were disasters just waiting to happen.

To avoid being taken in by the myth of risk management, we must recognize that software projects are *inherently* risky and failure prone. Rather than assume we can identify and “mitigate” the risks we will face, or assume that this time our project will be different, we must assume that we are doing something that is leading our project closer to failure.





IN FICTION IT MIGHT BE OK TO SAY THAT THE EMPEROR IS IDEALLY DRESSED FOR A PARTY AT THE NUDIST BEACH, BUT IN MOST ORGANIZATIONS TELLING THE TRUTH IS A CLASSIC “BAD CAREER MOVE.”

While it is true that there are some things for which every project should prepare, this is a short, generic, and generally useless list. Rather, you should focus on these following items:

- Politically unmentionable risks
- Uncontrollable factors
- Unquantifiable risks
- The unexpected

### POLITICALLY UNMENTIONABLE RISKS

Many organizations have the habit of “shooting the messenger” whenever there is bad news. So when a project crashes and burns, the developers and the users are not at all surprised, but the “senior” managers, who were supposedly providing “adult supervision” of the project, are utterly shocked that the project has failed, or they simply declare “mission accomplished” and leave the users to deal with the mess of a buggy, partially implemented system.

I recently observed this on an important project with a mandated completion date dependent on procuring some hardware that was required for a five-week suite of system tests. The hardware arrived a month late, but five weeks after the original planned start of system testing “questions were being asked” about why the testing was not yet complete. Several months later, the schedule slip had not been officially recognized in the project plan; instead there were repeated pronouncements that the timeline was “not allowed to slip.” Eventually, when the original project completion date was near, management was forced to acknowledge the slip.

When the people in charge of managing the risk are the ones creating the risk, it is not hard to see that risk management is an illusion. Examples include:

- Hoping that a major slippage can be caught up
- Mandating that a project be completed by an arbitrary due date that is a “stretch target”
- Delaying the start of a project but still expecting the original completion date
- Assigning unskilled developers to a project but not allocating any time for training and mentoring

It should be possible for team members to point out these problems and management to correct the mistakes. Unfortunately, life does not work that way. In fiction it might be OK to say that the emperor is ideally dressed for a party at the nudist beach, but in most organizations telling the truth is a classic “bad career move.”

So what really happens? All *minor* risks are identified and “mitigated” while there is a big elephant in the room that nobody is talking about. Sometimes, if you are lucky, you can get an outsider to raise the issue, but in most cases it just sits there unmentioned, the project crashes and burns, and the failure is attributed to other causes, generally something outside everyone’s control.

There is a simple fix for project politics, but it can be difficult to implement: Ensure that there is a *balance of power* between the business and technology sides of the project. This is necessary so that each side can hold the other accountable for its promises and decisions. Too often this is a one-way proposition.

### UNCONTROLLABLE FACTORS

Whenever I need some light relief on a project, I look at the mitigation strategies that are in place for the various risks

that have been identified. In most cases the “strategy” is nothing more than a reactive policy to work harder, spend more money, reduce the project scope, or slip the schedule. Rarely does the mitigation strategy include anything that implies that the team will have to think deeply about how to address the actual circumstances at the time the risk is detected.

Most risk mitigation strategies fail to take into account that software development is not predictable—there is an essential, uncontrollable uncertainty and randomness to it. Back near the dawn of software development, when people actually did empirical studies of how projects turned out, they found that early estimates of project size could easily be off by a factor of four [2]. So something that was estimated to take one person-year could take anywhere from three person-months to four person-years.

This level of uncertainty is not acceptable to senior managers, so most organizations have mandated a limit to the variability of estimates. Often a project is allowed to miss its estimates by up to 20 percent (maybe even 50 percent in more enlightened organizations). But all the managers do by mandating a 20 percent limit is make it politically unacceptable to talk about the real project variability, which is twenty times larger.

Organizations must adopt approaches that *acknowledge and embrace uncertainty* because many factors that influence a project are outside of the project’s control. This means organizations must select leaders who are open to alternatives and are not afraid of surprises. Unfortunately, most organizations do not follow this practice, in part because it is not easy to find such leaders.

## UNQUANTIFIABLE RISKS

In games of chance, the probability of an adverse event can be determined. However, knowing the probability of such an event in a software development project is essentially unknowable because software projects are essentially human activities, and humans are unpredictable. Even in games of chance where the probabilities are directly computable, most people will drastically overestimate their chance of winning as is evidenced by the number of people who buy lottery tickets. Yes, someone will eventually win the big prize, but the chance that it will be you is usually less than one in ten million, so the expected return on a \$1 ticket is in most cases less than fifty cents. (Lotteries have expenses, many are designed to raise money for charity, so few lotteries pay out more than 50 percent of the money they bring in.)

We are not good at assigning probabilities to events, even when we have good statistical data, and we fail miserably when we don't have any historical data. A good example is how much certainty we ascribe to our estimates. We typically believe our estimates are within 50 percent even though empirical studies show that estimates can be off by a factor of four.

Even when we have good statistical data, it is not always available to the project team. For example, when planning a project, it is useful to know the level of staff turnover. If it is more than 15 percent, there is a good chance that one in six team members will leave before completing a one-year project. But many companies will not make this information available to project managers, so although they know that their project will have some turnover, they cannot quantify the risk.

Even if the project manager is given the overall corporate turnover numbers, the risk is still unquantifiable because the circumstances of the project can change the probability that team members will leave. If a project is afflicted with particularly bozo decision making, then the team may suffer an almost complete turnover of staff. Even worse, you may find that although a central core survives, there is rapid attrition of all new hires, which could have been predicted given the overall average turnover for the organization.

Dealing with unquantifiable risks requires accepting that not everything is reducible to numbers and probabilities. It requires senior managers who are comfortable with unknowns and who are willing to accept that some things are unknowable. Risk averse managers cause many projects to fail because they make "safe" decisions that end up putting the entire project at risk.

## THE UNEXPECTED (NO ONE EXPECTS THE SPANISH INQUISITION!)

Even in organizations that seem to identify risks early, there always are some items that just pop up to cause a major impact on a project. Even worse, these often are things that catch everyone by surprise but seem obvious in hindsight. A good example of this is the "slashdot effect" [3], in which your Web site goes from total obscurity to overwhelming popularity and then crashes due to overload. Yes, in hindsight it is obvious that Web sites should be designed to support large numbers of visitors, but as few as five years ago, major corporations with eCommerce Web sites were regularly failing whenever they had a sale or major promotion.

The problem with drawing attention to such unexpected events, however, is that someone, somewhere will add them to a risk checklist for all future projects to assess. But again, it is not the events that you are prepared for that cause the biggest problems—it is the events that catch you by surprise. Few, if any, surprises in software development result in a project that is easier to deliver—most surprises mean that something has gone wrong.

Big checklists of possible risks are not much use. After all, the likelihood that your building will be destroyed is very low, but because we have seen some very visible examples of this type, most organizations now have some sort of disaster recovery plan in place. That means the organization is spending a substantial amount of money to address a single, unlikely risk and thereby has less money to spend on figuring out ways to respond to all of the other types of risks or on improving existing systems.

Most risk management plans, while useful in a CYA way, have no impact on the overall success of projects. The risks that make it onto the risk mitigation lists

are interesting in an academic sense, but not all that useful. Good project managers know that suppliers may be late so they schedule appropriately; they do not list this as an item in the risk mitigation plan. I'll say it again: It is not the known risks that cause problems; it is the completely unexpected things that get you. Too many projects crash and burn because senior managers are completely out of touch with the realities of their projects.

Flexibility is the key to surviving the unexpected. Rather than making detailed project plans and creating interminable mitigation strategies for every conceivable risk only to be blindsided by something that wasn't in the plans, organizations need to understand that bad stuff happens.

## *Forget about Risk Management, Create Your own Insurance*

The current methods used to manage risk are obviously failing or there would not be as many stressed projects. An "insurance model" of project management may sound a little crazy, but having insurance is usually less risky or expensive than the alternative. What would project insurance look like? What would we be insuring against?

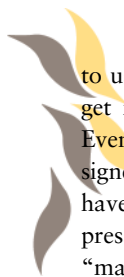
The insurance should cover:

- Incomplete information
- Learning the wrong lessons
- Death by a thousand cuts
- Late-breaking news

## INCOMPLETE INFORMATION

All projects must insure themselves against *incomplete information*, especially information about how a project is tracking. While most management teams set up structures that keep them informed of project concerns, most managers want to be brought "solutions, not problems." As a result they live in a world that is out of touch with the underlying project realities because nobody is allowed to raise an issue without having a well thought out plan for addressing the problem. The result of this policy is that intractable problems are not being raised as issues.

It's common among developers to say that things would go much better if managers would just listen to them and



to users, but sadly, such managers don't get far in today's management culture. Even the Big Visible Charts that are designed to make issues visible to the team have little impact. With enough political pressure, the number of critical defects "magically drops" as the deadline to go live approaches, but this does nothing to mitigate risk. As journalist George Monbiot [4] reminds us, "Tell people something they know already and they will thank you for it. Tell them something new and they will hate you for it."

### LEARNING THE WRONG LESSONS

We hope that organizations could learn through project retrospectives, but in practice few organizations bother to really analyze their failed projects; when they do, they often learn the wrong lesson. There are so many ways to really mess up a project that it is easy to focus on a specific problem in a project and then mandate that all future projects must do something to mitigate that problem. So, for example, we see many companies requiring a requirements document to be signed off ("frozen") by a senior manager before design can start because one project had a problem with requirements changing during the project.

A good example of learning the wrong lessons is the way NASA responded to the Challenger disaster discussed earlier in this article. Yes, there was an inquiry and a report, but it did not fundamentally change the way that people went about their jobs. What is especially tragic about this is that Richard Feynman had to write a minority report to get his findings included, and even then, very few people responded to his key message: "For a successful technology, reality must take precedence over public relations, for nature cannot be fooled."

As NASA has shown, it is difficult to avoid learning the wrong lesson. Yes, there were some specific issues with the technology, but NASA's main failing was the cultural one of downplaying the risks and trying to sell the idea that spaceflight was becoming routine. Insuring against this is difficult because cultures are resistant to learning new ways of doing things—things can be changed within the existing framework, but changing the framework requires revolution.

### DEATH BY A THOUSAND CUTS

What managers often fail to understand is how seemingly minor things can result in project failure—simple things like a few days' delay in getting access to other required systems. The developers must guess how the other system operates, and a few weeks or months later, that guess turns out to be wrong.

Projects fail because a lot of little things go wrong, and by the time these minor things have made their way through the reporting structures and become visible, the project is headed for disaster. Insuring against this requires the invention of a reporting mechanism that will allow the implications of a set of little issues to reach the project decision makers much earlier.

### LATE-BREAKING NEWS

Insurance also is needed for late-breaking news about project breakage. Typically there is ample early warning that things are not quite as they should be, but normally there is no *one thing* that is really specific, just a lot of little hints that something bad is likely to happen.

Although there is no simple mechanism for early detection of issues, the agile approaches have made it much easier. They have done this by taking the deadline—a much-loved project management tool—and making it a regular part of the day. Living with hard deadlines *every day* makes it virtually impossible for emerging problems to escape notice.

On the micro-scale, many agile teams who are now using test-driven development or a similar practice that ensures clean code have a daily milestone for checking their integrated, tested, working code into the source code repository. With this daily milestone in place, teams can easily see when code quality is decreasing if the effort required each day to check in code starts to increase. Similarly, there is an early warning about progress, or the lack thereof, by the amount of functionality that is added every day.

Some teams may ignore this daily feedback, but with diligence, the discipline of the daily deadline constitutes an effective early warning system about declines in code quality and team productivity.

On a larger scale, the agile practice

of having two-week to one-month development iterations provides an excellent deadline for measuring project progress. At the start of each iteration the team agrees with the key stakeholders about what team members will deliver, and at the end they show that the team has delivered what was agreed. This provides yet another early warning mechanism for all stakeholders about the ability of the team to deliver.

### *Focus on Becoming Flexible and Reactive*

The most important bit of insurance a project can buy, however, is insurance against not comprehending that things are changing. A project needs to be viewed as a *learning environment* for everyone involved. If new information is not coming to light every day and being acted upon, no one is learning anything about the project's true status, and the project is likely to fail. Constantly monitoring progress takes much of the drama out of uncertainty, and, therefore, much of the risk out of a project. Variability in project parameters is natural; being continually aware, informed, and responsive needs to be just as natural.

Rather than planning for specific, known risks, projects should work from the assumption that something will occur that impacts the project, and create mechanisms that provide for the *early detection* and *response* to such events. It is not the known things that cause us problems. It's the *unknown* and *unknowable* that cause projects to crash and burn.

As an insurance policy, learning is rare but crucial to project success. Perhaps it was best summed up by Multics wizard Tom Van Vleck [5], who said, "You learn something every day, unless you're careful." **{end}**

---

#### REFERENCES:

- 1] [science.ksc.nasa.gov/shuttle/missions/51-1/docs/rogers-commission/Appendix-F.txt](http://science.ksc.nasa.gov/shuttle/missions/51-1/docs/rogers-commission/Appendix-F.txt)
- 2] *Software Engineering Economics*, Boehm, B., Prentice Hall, 1981.
- 3] [en.wikipedia.org/wiki/Slashdot\\_effect](http://en.wikipedia.org/wiki/Slashdot_effect)
- 4] [www.monbiot.com](http://www.monbiot.com)
- 5] [www.multicians.org/thvv/](http://www.multicians.org/thvv/)



# Are you doing **automation** the smart way?

Companies look at test automation as the cure-all for their problems in improving testing cost and effort. By itself, automation cannot solve your problems.

Automation involves a significant up-front cost for building the automation test script suite. For this effort to generate sizeable returns, the scripts created need to be reusable and amenable for any user to execute them. This can only be ensured if the automation objectives are defined in advance and the entire automation exercise is well planned.

## The right approach to test automation

Obtaining a clear understanding of the automation landscape before commencing automation is crucial to delivering high returns on your automation.

An Automation Assessment will establish consistency around your automation approach across different teams; thereby reducing the development and maintenance cost by enhancing reusability. The assessment will help you standardize your automation in terms of process, framework, tools, metrics, operating model, environment and test data management. In addition to an evaluation of your current automation practices, incorporating mechanisms to analyze existing manual processes for automation readiness will maximize your automation benefits at an enterprise level.

Defining your automation scripting standards and guidelines and designing a reusable user-friendly automation framework will help you build high-quality test scripts with consistency.

## Cognizant makes it simple

Cognizant's expert group on Test Automation has developed a wide range of frameworks and tools that will help you get more returns for your automation.

**Automation Assessment** Cognizant's automation experts have developed a comprehensive framework to assess your test automation across a wide range of parameters and sub-focus areas.

**ROI Calculator** This proprietary tool provides an estimate of the ROI before the automation project commences. This will help you take an informed decision on whether, and when, to automate your test scripts.

**CRAFT** Cognizant's Reusable Automation Framework for Testing is designed to simplify the process of script development and execution.

**Win2Pro** Cognizant's Win2Pro will help you convert your HP WinRunner scripts to HP QuickTestPro; thereby drastically reducing the cost associated with designing the test scripts afresh for QTP.

*To know more about Cognizant - the world's largest testing service provider, please log on to [www.cognizant.com/testing](http://www.cognizant.com/testing)*



**Cognizant** | Testing Services  
Passion for building stronger businesses

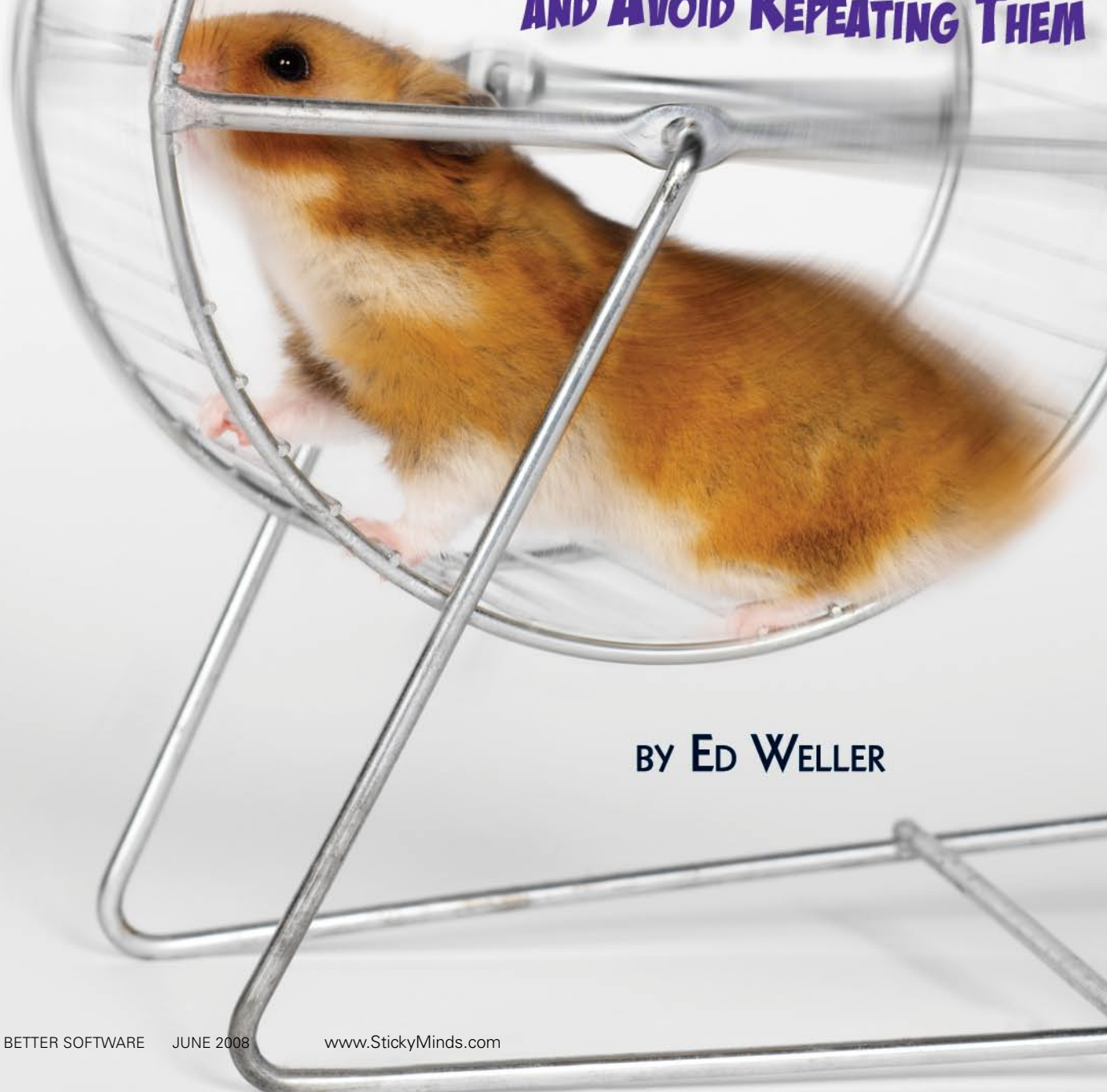
### World Headquarters

Cognizant Technology Solutions  
500 Frank W. Burr Boulevard  
Teaneck, NJ 07666  
Phone: +1 201 801 0233, Fax: +1 201 801 0243  
Toll Free: +1 888 937 3277  
Email: [inquiry@cognizant.com](mailto:inquiry@cognizant.com)



# STOP THE INSANITY

USING ROOT CAUSE ANALYSIS  
TO UNDERSTAND MISTAKES  
AND AVOID REPEATING THEM



BY ED WELLER



**"TO ERR IS HUMAN, TO FORGIVE IS DIVINE."**

**— ALEXANDER POPE**

**"INSANITY: DOING THE SAME THING OVER AND OVER AGAIN AND EXPECTING DIFFERENT RESULTS."**

**— ALBERT EINSTEIN**

**T**HESSE two statements summarize the underlying philosophy of root cause analysis (RCA). We must accept the fact that we make mistakes in product development, but we must also accept the responsibility for not repeating those mistakes. Absent any proactive steps to prevent mistakes, we won't be very successful in preventing them. And absent any proactive steps to understand our mistakes, we will repeat them, thus meeting Einstein's definition of insanity.

RCA has many definitions. To some, it is identifying the cause of an application failure and fixing it. To others, it is identifying the reasons for making the initial error or mistake that led to the failure. This confusion is in part caused by differences in analyzing and preventing physical failures versus analyzing and preventing errors in the intellectual activity used in developing software. Industrial accidents or system failures often have an unknown physical cause until investigated, and the RCA focuses on identifying the mechanical, chemical, electrical, or physical cause of the failure. These include mechanical stress, overheating, fatigue, etc. Prevention usually requires a change to one or more physical components of the system. On occasion, these failures are traced back to design or management decisions.

When we trace software failures back to the initial mistake, most often they are the result of an error in our thought processes or intellectual activities. In software development, the common view of RCA is identification of conditions or events that caused a person or team to make an initial error that later manifested as a defect or failure.

To further understand this difference, let's consider some well-known failures. There have been two space shuttle disasters. In both cases, the manifestation of physical failure had been observed multiple times in earlier flights. In both cases, management decisions did not correctly assess the risk of catastrophic failure. These disasters occurred through a combination of physical faults and decision-making faults. The TWA800 disaster was caused by multiple faults: an explosive air-fuel mixture in a fuel tank and an ignition source. The older model F-15 fighters are failing due to stress fractures. These two cases can be attributed to identifiable chemical or mechanical causes that were not identified or understood in

the design of these systems. In all four cases, RCA could stop at the "how to fix it" point or, as in the shuttle case, go beyond the physical cause to the underlying decision process.

However, when we consider software, we have a situation where trivial oversights or typos can cause expensive failures. A missing hyphen caused a \$500 million Mariner probe to Venus to malfunction. A one-character error in the AT&T 5ESS system caused a \$1 billion failure in the 800 phone system. On the other hand, one-character errors in software often are discovered in reviews, testing, or even in use with insignificant failure costs. The lack of a relationship between the error significance and the failure cost makes software RCA fundamentally different. The same trivial error can lead to failures with orders of magnitude of difference in cost. Or, looking at this from another viewpoint, identifying the root cause of a costly failure does not guarantee you will eliminate costly failures elsewhere due to similar causes. In all of these cases, the programmer made an error in creating the work; in other words, the intellectual thought process failed. In software work, there is no physical trail of evidence to lead us to the error. In addition, we have the problem of the frequency of errors. Any sizeable system will have hundreds of errors waiting for the right set of conditions to cause a failure.

Let's establish some definitions. The first three are adapted from *Software Metrics* by Fenton and Pfleeger:

- Human error—a mistake by a human in developing a software work product
- Fault—the encoding of the human error into the software; note that one error may result in multiple faults
- Failure—the manifestation of the fault in the execution of the software
- Defect—often used to mean any or all of the above, but in my opinion it's best used to refer to faults or failures discovered during reviews or testing

Software failures and defects are discovered in use or by testing. At times, the failures require extensive analysis to pinpoint the fault in the code. This problem solving is sometimes referred to as RCA. From the perspective of the operational staff or users, this definition works since they do not want the failure to occur again in the existing product; they want it fixed.

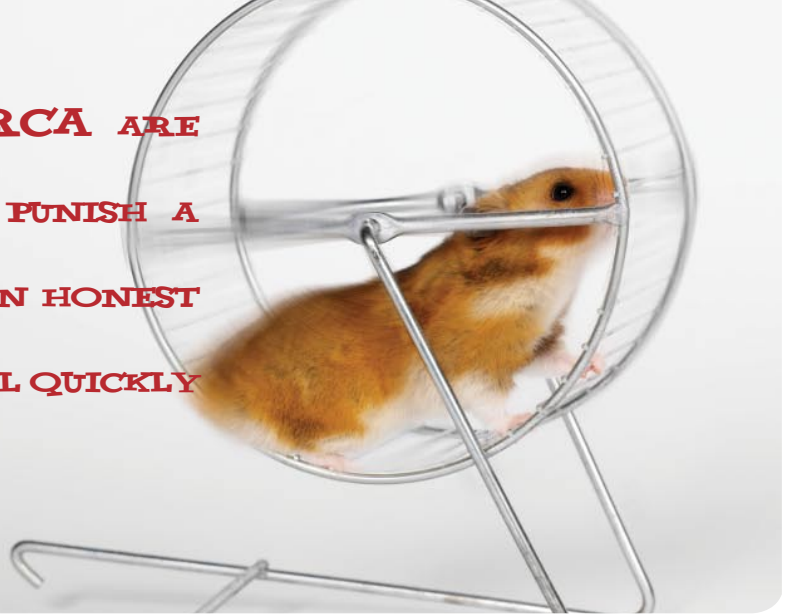
From a development perspective, we want to prevent errors due to this root cause from recurring in future products or development activities. This means we must look for the underlying situation that allowed us—or even encouraged us—to make the error. This can be much more difficult than identifying a physical cause. It also means that to fully understand how the error was made, we need information from the person who made the mistake. Others may guess at the cause, but in most cases only the person making the error can provide insight into the intellectual process that initiated the error.

## COMMON PROBLEMS WITH RCA

I have seen companies try to apply RCA to production failures in an attempt to improve operational reliability over the



**IF THE RESULTS OF AN RCA ARE USED TO REPRIMAND OR PUNISH A PERSON WHO HAS MADE AN HONEST MISTAKE, THE PROCESS WILL QUICKLY CEASE TO BE EFFECTIVE.**



short term. Their thinking seems to be “If we can just get the developers to make fewer errors, we can improve product quality.” While this is true over the long run, this approach ignores the time lag between understanding the root cause, identifying where the cause actually occurred, and then applying prevention to that activity. In the case of requirements errors, the next opportunity to apply the prevention is in the requirements gathering activity of the next version or even the next product. This means that actual improvement seen by users will be at least a full product release cycle away.

A second problem is failure to take action after the analysis has identified the root cause. The process seems to morph from “Find the root cause and prevent its recurrence” to “Hold the RCA meeting to meet a goal” and no further time, people, or money are spent in actually preventing the problem from recurring. Eventually this leads to “Why are we wasting time in this meeting? No one

ever does anything with the findings,” and RCA is abandoned as ineffective.

A third and even deadlier problem is punishing the person who made the error. If the results of an RCA are used to reprimand or punish a person who has made an honest mistake, the process will quickly cease to be effective. Making the same error over and over after identifying the problem and applying corrective action is another story. The key is to use the results of RCAs to improve a person’s or team’s capability. I would advise erring on the side of “forgiving and learning” rather than “reprimanding and punishing.”

## RCA VERSUS RETROSPECTIVES

While an important process improvement technique, retrospectives do not typically focus on root causes. They are a good source of information for RCA but often only identify things that worked well and should be done again or things

to be avoided the next time. For example, a finding of a retrospective might be “Too many defects found in module xyz in system test delayed delivery.” At this point, an RCA of the cause of those defects would be performed. Another finding might be “Initial estimates were 50 percent too optimistic.” An RCA of the estimating process could be conducted. In many cases, the output of retrospectives or lessons learned is identification of the “primary effect” (a term to be explained later) rather than the root cause.

## THE RCA PROCESS

The following three methods have been found useful for conducting RCAs for software failures:

- Apollo method
- Fishbone or Ishikawa diagrams
- 5 Why

I prefer the Apollo method for analyzing software failures, as it has *enough*

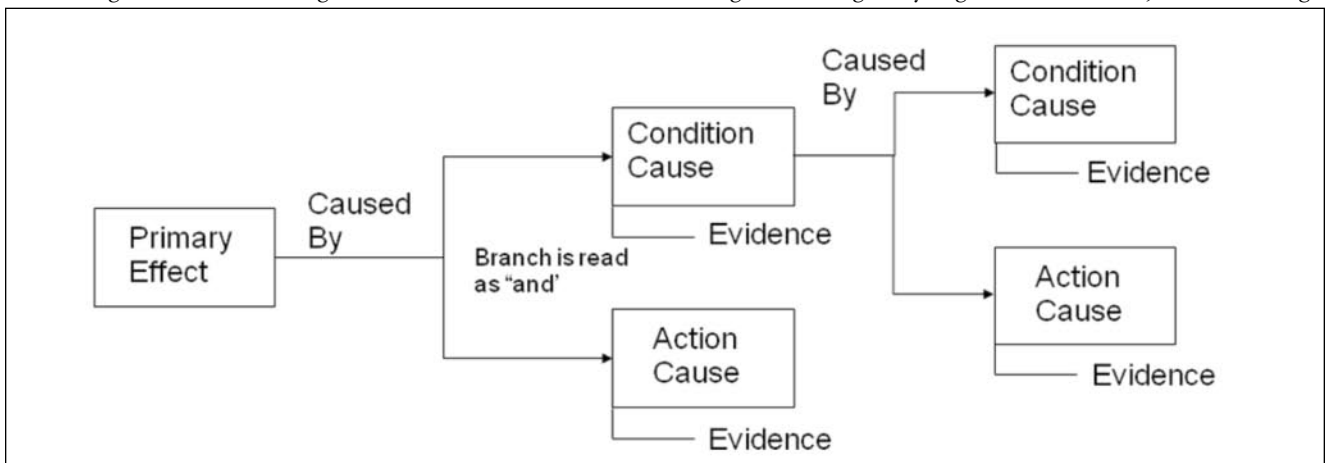
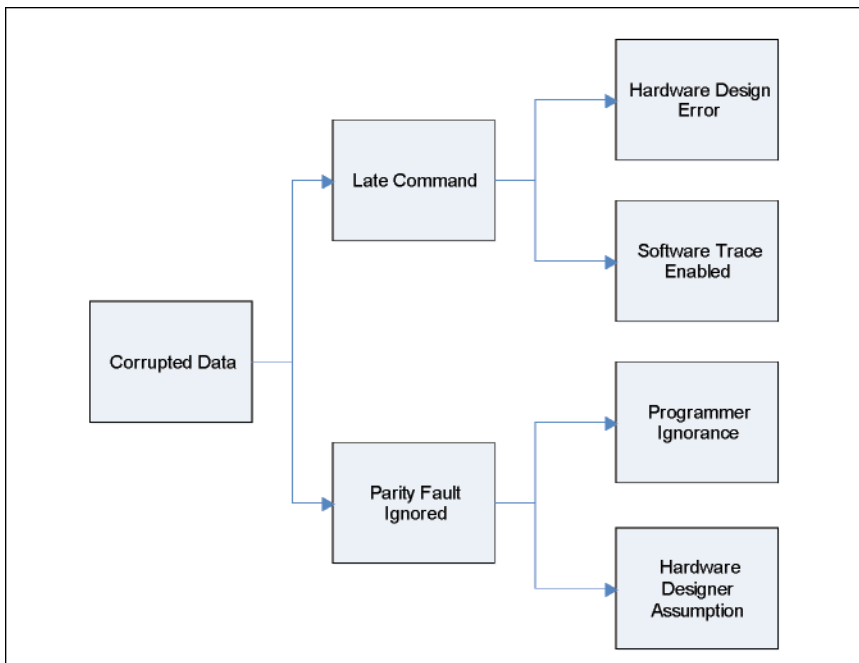


Figure 1: Apollo method cause-and-effect chart



**Figure 2: RCA cause-effect diagram for corrupted data primary effect**

structure but not too much. Fishbone analysis is best left to aggregations of RCAs or process problems. Apollo and 5 Why use similar approaches, but the Apollo method has more structure and guidance. For those wishing to dig deeper into this method, I suggest reading *Apollo Root Cause Analysis* by Dean L. Gano. We'll cover the basic process here.

The Apollo method uses a cause-and-effect chart. The two-level picture shown in figure 1 will be used to explain the principles.

The “primary effect” is the problem or failure we want to prevent from recurring. We need to identify the what, when, and where as well as the significance of the failure. To ensure all are on the same page, these must be clearly stated and understood by all members of the RCA team before the analysis starts. Understanding the significance is needed to allow the team to make appropriate cost-benefit recommendations for those failures where there is a relationship between cause-and-effect cost. Statements must be specific; “System down” has far less meaning than “One hour of production lost at a cost of \$1 million.”

Once the primary effect is identified, understood, and agreed upon, the next step is to diagram the cause-and-effect chain. In the Apollo method, the term “caused by” is used, and I suggest

using this exact phrase when building the chart. Saying “caused by” focuses participants on understanding the link between the two causes or actions. The causes can be divided into a “condition cause” (the necessary set of conditions that exist over time) and an “action cause” (the event that triggered the failure). A software example of a condition cause might be the failure to check for buffer overflow and the action cause a hacker overflowing the buffer in an attempt to breach the system, leading to a primary effect of a security failure. Another condition cause might be programmers failing to monitor device status for data errors and the action cause a reported parity error, leading to the primary effect of corrupted data accepted by the system.

The pairing of the condition cause and action cause is an “and” condition, in which both must exist to cause the primary effect. If either alone could cause the primary effect, then it is likely there are two separate paths, and the causes should be further divided into condition and action causes. Understanding there is a difference may allow the analysis to proceed to a better understanding of the failure cause.

The Apollo method strongly suggests that evidence for the causes be noted. Supposition and guesswork should not be used at this point. If you do not

know, close the path with a “?” and go on to examine other paths. Additional investigation should lead to stopping the path or further diagramming when more information is available. The analysis continues down the path until the point of “collective ignorance” is reached. In other words, no one can identify additional caused-by relationships.

Let's analyze a corrupted-data primary effect. In this example, the corrupted data was caused by the action of hardware and software timing (action cause) and software ignoring the parity fault (condition cause). Following the action cause, the timing was caused by a long delay in sending a command (action cause) and a hardware design error (condition cause), caused by enabling the software trace to log events. Following the first condition cause, ignoring the parity fault was caused by programmers unaware of the significance of the fault (“We don't know what to do with parity faults, so we ignore them”). Their ignorance could be traced to lack of knowledge in handling faults and an assumption by the hardware designer that the programmers would “obviously” know a parity error meant data was invalid. Figure 2 shows one way of drawing these relationships. Each branch should be read as “caused by,” with the action and condition causes as stated above. I have left the evidence off because in this analysis each of the causes was agreed to by the team and the level of complexity did not require this step.

Note that in this example the first branch is an “and” condition because if either path is prevented, the failure will not repeat.

If we wished, the “hardware design error” path could be further expanded.

As we develop this chart, it is useful to follow the path from left to right, stating “Corrupted data is *caused by* a late command *and* parity fault ignored; late command is *caused by* hardware design error *and* software trace enabled; and parity fault ignored is *caused by* programmer ignorance *and* hardware designer assumption.” This verifies that the “caused by” chain is valid and that we have not skipped any intermediate causes.

We should also walk the chart from right to left, stating: “Programmer ignorance and the hardware designer

assumption that programmers knew what to do with parity faults *caused* a parity fault to be ignored, which *caused* corrupted data.” This two-way verification of the analysis keeps the logic of the analysis on track.

Once the analysis is finished we are only about 25 percent done! The next step is identifying potential solutions. These should be in line with the resources pre-allocated by management for problem resolution, although there should be exceptions when a serious problem would require more than the agreed upon resources (in other words, although there may be guidelines, let common sense rule the decision).

In the example above, there are four possible solutions:

1. Fix the hardware design error.
2. Do not enable the software trace.
3. Train software programmers on the correct response to the error status.
4. Train the hardware engineer to fully communicate the impact of status errors.

In this case, solutions 1 and 2 do

not address the root cause. These are problem fixes. If the hardware-design error cause had been traced further, we might get to a root cause (designer oversight, inability to add sufficient hardware to address problem, etc.). To prevent the problem from recurring in other applications with other hardware, solutions 3 and 4 are preventive actions. Adding solutions 3 and 4 to a checklist used at project startup will help the staff “remember” to do this, as well.

## MULTIPLE DEFECTS

Most software systems have a large number of defects. How can root cause analysis be effective in this situation? Typically, causal analysis waits until multiple defects have been found to allow the RCA team to work through ten to twenty defects in a single meeting. This may seem like a large number, but most software defects can be analyzed fairly quickly. A team of developers would wait until a sufficient number of defects have accumulated, perform the RCA for each defect, and then group the causes to identify common causes, if any. If we are able to identify one cause contributing

four to six of ten defects, one action can prevent a large percentage of the errors. This is where the real power of RCA comes into play. Carrying this a step further, there should be a team responsible for coordinating results from multiple RCA teams, as an infrequently identified cause at the team level may have a wider impact across the organization.

## SUMMARY

The most important factors in the success or failure of root cause analysis are attitude toward the findings of the causes (“fix the problem” or “punish the people”) and follow through with action plans and solutions. If these are done properly, organizations can work through the process. No matter how well the analysis is done, lack of follow through will make the process another make-work task with no value. With the right support and training of participants, root cause analysis can play a significant role in improving product quality and organizational effectiveness.

{end}



## Expanding Agile Horizons

### LEARN HOW TO DELIVER BUSINESS VALUE WITH AGILE

Join us in Toronto for the biggest ever gathering of agile practitioners and thought leaders. This conference expands Agile from software development to delivering business value. At Agile 2008 you will discover how to put agile principles to work!

We are pleased to announce opening keynote speaker Jim Surowiecki, author of the book “Wisdom of Crowds”. Bob Martin, author of “Agile Software Development, Principles, Patterns, and Practices,” speaks at our last night conference banquet open to all attendees. Our closing keynote is Alan Cooper “Father of Visual Basic” and the inventor of using personas in Interaction Design.



Early bird registration is open now! Details at [www.agile2008.org](http://www.agile2008.org)



Toronto August 4 to 8, 2008





# Software Testing Certification

## *Certified Tester—Foundation Level Training*



***More than 80,000 Certified Testers Worldwide. Why Not You?***

### SUMMER/FALL 2008

Las Vegas, NV	June 8–10, 2008
Bethesda, MD	June 10–12, 2008
Ottawa, ON	June 17–19, 2008
Portland, OR	June 17–19, 2008
Milwaukee, WI	June 17–19, 2008
Boston, MA	August 26–28, 2008
New York/NJ Area	September 8–10, 2008
Minneapolis, MN	September 9–11, 2008
Salt Lake City, UT	September 9–11, 2008
Washington, DC	September 15–17, 2008
Philadelphia, PA	September 23–25, 2008
Atlanta, GA	September 23–25, 2008
Anaheim, CA	September 28–30, 2008
Indianapolis, IN	Sept. 30–Oct. 2, 2008
Jacksonville, FL	October 7–9, 2008

Toronto, ON	October 7–9, 2008
Rochester, NY	October 14–16, 2008
Kansas City, MO	October 14–16, 2008
San Francisco, CA	October 20–22, 2008
Sacramento, CA	October 21–23, 2008
Pittsburgh, PA	October 21–23, 2008
Omaha, NE	October 28–30, 2008
Charlotte, NC	October 28–30, 2008
Cincinnati, OH	October 28–30, 2008
Ft. Lauderdale, FL	November 4–6, 2008
Bethesda, MD	November 4–6, 2008
Tampa, FL	November 17–19, 2008
Sunnyvale, CA	November 18–20, 2008
Phoenix, AZ	December 2–4, 2008

[www.sqetraining.com/certification](http://www.sqetraining.com/certification)

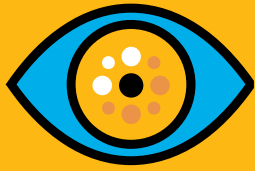
- **Basics of testing** – Goals and limits, risk analysis, prioritizing, completion criteria
- **Testing in software development** – Unit, integration, system, acceptance, and regression testing
- **Test management** – Strategies and planning, roles and responsibilities, defect tracking, and test deliverables

**PM** Project  
Management  
**R.E.P.** Institute

**SQE**  
TRAINING

[www.sqetraining.com](http://www.sqetraining.com)

**On-site Training Available**—For additional savings, bring this course to your organization for team training.



# Can you achieve application quality without application security?

Many companies are under the impression that testing for web application security simply involves a cursory check for easy-to-guess user names and passwords. Yet application security testing can and should involve more complex audits, such as testing for SQL injection and cross-site scripting vulnerabilities. Often this sort of review does not happen until the web application is in production, when it is too late to stop a hacker or a malicious program from attacking and much more expensive to remediate the vulnerability.

While quality assurance (QA) departments have traditionally focused on functional or performance testing—it is a clear trend that QA is becoming a critical participant in application security testing.

## Are you ready for security testing?

There are three ways that your QA department may become involved with web application security testing:

- Your company's web security experts may request that application security testing be done by the QA group to ensure that all fixes have been implemented and no security holes exist prior to releasing the product to production.
- Your compliance officer—facing concerns about Sarbanes-Oxley (SOX), Health Insurance Portability and Accountability Act (HIPAA), payment card industry (PCI), etc.—may request that further application security testing is performed during the QA process.

- Your QA department may request involvement with testing for web application security, because an application with potential security holes is not going to be perceived as high-quality by users.

No matter how the department gets involved, certain steps will need to be taken to establish the application security testing process. It will need to be determined whether there will be specific, dedicated staff members who will be performing web application security testing, or whether the task will be dispersed throughout your entire QA group. In addition, the timing of web application security testing during the QA process will need to be managed. Ideally, application security testing will be performed as early as possible, so that developers can fix any security issues in a timely manner without compromising the project's schedule. Finally, the right software for application security testing will need to be selected and implemented.

## The right approach to application security testing

The QA department will need application security testing software that is able to perform three different types of testing to determine the vulnerabilities inherent in each user class: as a non-authenticated user, an authenticated user, and an administrative user. Additionally, the web application security tool should be able to perform both automated and manual crawling/spidering of your web application.

**Run a free test of your web applications via our free 15-day trial of HP QAInspect® software and get a comprehensive vulnerability report.**  
**[www.hp.com/go/QAInspectdnl](http://www.hp.com/go/QAInspectdnl)**

Automated application security testing software will spider the entire application by clicking every button and link, filling out data fields to identify the structure of the program, and then auditing each page for vulnerabilities. It should do this from the outside in, reviewing each portion of the site the way an external hacker might. This comprehensive approach is valuable to ensure that all security holes have been identified and can be fixed. On the down side, it can also produce false positives, and it may not be able to access all of your web pages due to the way that certain pages are coded.

Manual testing allows a user to focus on specific pathways or tasks on a website while the software follows silently behind, tracking the process. The program can then audit the particular path that the user has taken for security vulnerabilities and provide a report. Manually crawling an application can be time consuming, but it also ensures that specific pages are tracked and analyzed.

## Choosing the right products

The following basic questions should be addressed when you are looking for a web application security testing product:

- How easy is the product to use?
- What kind of training will your QA department require in order to properly use the product?
- How well does the product integrate into the tools and software that are already used by your organization?
- How often is the product updated with new security checks—daily, weekly, monthly?
- What is the false positive rate of the product?  
While no product is perfect, you want to find a product with as a low a rate as possible so that your resources are not wasted going through false positives.

- How well does the product integrate with leading quality management platforms?
- Does the product appear to evaluate each page of your application or does it get stuck on certain pages?
- Does the product allow the end user to easily modify scan settings?
- What kinds of restrictions are in the product's license?
- In which formats are reports offered (PDF, HTML, XML)? Are they easy to read? Do they contain information on the location of the vulnerability, how to execute it, how to verify it and how to fix it?
- Will the company allow you to evaluate the product before committing to purchase it? Confidential vendors will often provide a seven- to 15-day evaluation period.

## HP Software makes it easy

Leading the charge in application quality and security, HP Software has recently completed the acquisition of SPI Dynamics, the leader in web application security testing. SPI Dynamics technology, which is already seamlessly integrated with HP Quality Center software, enables organizations to assess security vulnerabilities along the entire lifecycle of web applications—including development, QA and operations. Customers can also use SPI Dynamics software to validate application security and quality to meet auditing and compliance requirements, such as SOX.

To find out more about HP Software's integrated solutions for Application Quality and Security, please visit [www.hp.com/go/software](http://www.hp.com/go/software)





## Product Announcements

### Parasoft SOAtest ISO 8583

MONROVIA, CA—Parasoft announces native support for ISO 8583 messaging, a standard for exchanging financial transactions. Parasoft SOAtest—key component of Parasoft's SOA Quality solution—now helps teams to establish and manage a consistent, continuous quality process for the electronic payment systems vital to business operations.

The new Parasoft SOAtest ISO 8583 capabilities provide an easy-to-use graphical interface for configuring binary message formats. Teams working on systems that exchange financial transactions can now leverage Parasoft SOAtest to simulate clients for sending and receiving ISO 8583 messages.

Visit [www.parasoft.com](http://www.parasoft.com) for more information.

### GUIDancer Version 2.1

GERMANY—BREDEX GmbH releases version 2.1 of GUIDancer. With the addition of support for drag-and-drop actions, version 2.1 increases the amount of tests that can be automated with

GUIDancer. The new drag-and-drop actions can even be executed on individual tree nodes and table cells, allowing tests to achieve the same level of interaction with an application as a manual tester would.

The GUIDancer approach to test automation is keyword-driven. Tests are built from reusable modules (keywords or test cases). The keywords can be abstractly specified to ensure the highest possible reusability and flexibility in the test. Because no application under test is required to start test creation, integration testing can begin as soon as each version of an application is delivered. The modular structure means that maintenance is minimal, ensuring the stability of regression tests and making continuous testing possible and cost-effective.

Visit [www.guidancer.com](http://www.guidancer.com) for more information.

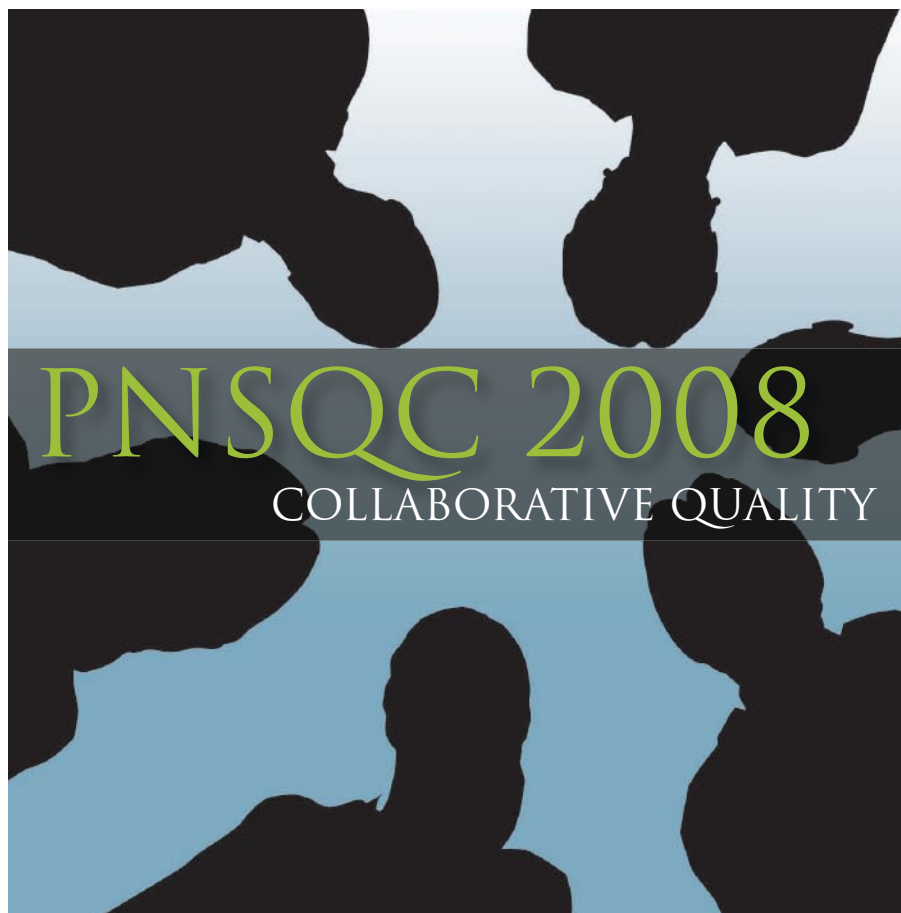
### PowerSQL

SANTA CLARA, CA—Embarcadero unveils its new professional-grade SQL development tool, PowerSQL. PowerSQL dramatically improves productivity for

Eclipse application developers charged with SQL development.

- SQL Code Assist ensures 100 percent object name accuracy, even when not connected to a database, plus real-time SQL syntax validation.
- SQL Project Insight provides project-level SQL file cataloging and search features to help streamline project organization and maintenance.
- Migration Wizard imports data sources from Eclipse DTP (Data Tools Project) or Quest TOAD.
- Data Source Explorer enables users to easily navigate, search, extract DDL, execute commands, and even browse an outline view without opening the SQL file.
- Formatting Profiles ensure consistent, quality code layout for easy review and extension. Profiles can be customized and shared.

Visit [www.embarcadero.com](http://www.embarcadero.com) for additional information



## 26TH ANNUAL PACIFIC NW SOFTWARE QUALITY CONFERENCE

October 13-15,  
2008  
Portland, OR

Collaboration spans levels, disciplines, and industries. Listen to these speakers discuss collaboration and how it can enable higher software quality.

- Ron Jeffries & Chet Hendrickson
- Sam Kaner
- Janet Gregory
- Karen N. Johnson
- Mike Kelly
- Steve Smith
- Tamara Suliaman & Hubert Smits

REGISTER EARLY AND SAVE		AFTER 9/8
3-Day Conference	\$950	\$1100
2-Day Technical Program	\$625	\$750
1-Day Workshop	\$475	\$550
Special discounts available.		

Visit Our New Website — [pnsqc.org](http://pnsqc.org)

## Software Partnership

AUSTIN, TX—QuickArrow, Inc. and Pervasive Software announce their partnership. This partnership enables QuickArrow to deliver a universal data integration solution that is easy to configure, maintain, and support. QuickArrow is leveraging the power of Pervasive Data Integrator to deliver QuickConnectIT, an integration engine capable of integrating QuickArrow with any application. This allows QuickArrow and Pervasive to deliver integrations that are consistent with the speed and economics of the Software-as-a-Service (SaaS) model on one code base. The partnership also enables QuickArrow to better focus on its core competency, Professional Services Automation.

The partners have already implemented more than fifty bi-directional integrations between QuickArrow and other best-of-breed applications such as salesforce.com, Microsoft Dynamics CRM, Peachtree, and QuickBooks. With QuickConnectIT and more than 150 standard Pervasive adapters in Pervasive Data Integrator v9, the partnership will enable QuickArrow to integrate with almost any other application using a configuration process versus a custom development exercise.

QuickConnectIT leverages Pervasive Data Integrator and QuickArrow's proven Web services API to support both on-premise and SaaS integrations and can also operate as an Integration-as-a-Service solution.

Visit [www.quickarrow.com](http://www.quickarrow.com) for more information.

## HOOD-to-UML

AMSTERDAM, THE NETHERLANDS—Artisan Software Tools and Objektrum Solutions are successfully working with BAE Systems to resolve the HOOD-to-UML migration challenge for its Nimrod project legacy designs. Combining Objektrum Solutions' HOODbridge utility, the Artisan Studio UML/SysML tool suite, and the BAE Systems Ada Profile results in a solution that significantly reduces the time it takes to migrate legacy designs into Artisan Studio ready for use.

BAE Systems reviewed its various migration options and engaged Objektrum

Solutions, a company with extensive expertise in the HOOD methodology, to investigate ways in which its HOOD data could be migrated into Artisan Studio, which was already widely deployed within BAE Systems. The outcome of this engagement has resulted in Objektrum Solutions' developing the HOODbridge, a standalone utility to migrate HOOD design data into an Artisan Studio UML model without any loss of design data integrity. Using the

Objektrum Solutions' HOODbridge with the BAE Systems' Ada Profile and code generation templates, it has proven the ability to migrate HOOD designs into Artisan Studio along with the generation of source code consistent with BAE Systems' original COTS HOOD tool.

Visit [www.artisansw.com](http://www.artisansw.com) for more information.

WHEN IT'S TIME



TO OUTSOURCE SOFTWARE TESTING



IT'S TIME TO CALL ACULIS.

ON-SITE, OFF-SITE, OFF-SHORE  
A BLENDED APPROACH TO SOFTWARE TESTING.

With state-of-the-art facilities in the U.S. & Argentina, our expert team of IT professionals deliver solid revenue producing solutions with speed and precision, while ensuring the deployment of high quality software.

On target, on time, on budget, contact ACULIS to accelerate your IT development and maximize your ROI.

**ACULIS** / IT ACCELERATOR  
DEVELOP IT / TEST IT / GLOBALIZE IT / STAFF IT

1-866-4ACULIS / WWW.ACULIS.COM

# Things You Might Not Know About

## Analysts' Nightmare Phrases

by Tim Lister

1

"DON'T WORRY ABOUT THE USERS. I'M THEIR MANAGER. I KNOW WHAT THEY NEED." Anytime you don't have the hands-on users on board, you are taking a big chance. Most bosses aren't experts on the detailed work of their employees.

2

"WHAT DO YOU MEAN THIS IS NOT IN SCOPE? WE ALWAYS ASSUMED YOU'D BUILD IT FOR THIS RELEASE." Implied requirements are the work of the devil. Do whatever you can to ensure that the scope is clear for all stakeholders. Draw a diagram of the scope and make in-scope and out-of-scope lists for all to read, especially for each release when you plan multiple releases to full function delivery.

3

"WE KNOW EXACTLY WHAT WE WANT." Ah, they prefer to self-medicate! These are the hardest clients to deal with. They don't want to talk about requirements; they know the solution. Try asking them, "If we build it exactly your way, what will you do with it?" Maybe you can find the underlying requirements by understanding its use.

4

"DON'T WORRY, THAT WILL NEVER HAPPEN." In analysis "never" equals "probability of occurrence is zero." It does not mean "very rare."

5

"THAT'S NOT A SPEC CHANGE; IT'S A CLARIFICATION." Call it a change, call it a clarification—you have just discovered work you didn't know existed. Usually these are variants or exceptions to a rule. Don't try to catch these later. Work hard to find these as you find the main policy.

6

"LET'S GET EVERYBODY IN ON THIS DECISION." No, let's get together those who are subject matter experts and those who will be impacted. Nobody else matters. Analysis is not a spectator sport.

7

"I'LL TELL YOU LATER ABOUT WHAT REPORTS WE WANT OUT OF THE NEW SYSTEM." No, we need these now to be sure we hold the right data at the right level of granularity.

8

"THE NEW SYSTEM SHOULD DO EVERYTHING THE OLD SYSTEM DOES, AND ... ." In other words, even though we have a chance to rethink it all, we decide to tear down the house, build it up just as it was, and add a new wing. This is an invitation to missed opportunity and mediocre outcome at best. Build a new system; everything is in play.

9

"WE'LL REVIEW THE SPEC WHEN IT IS COMPLETELY DONE." Why? Not one person has the expertise and the authority to rule on every piece of the spec. Even if he did, what a task! As you build, get sign-off only by those who understand the specific requirements. See No. 6.

10

"WE DON'T HAVE TIME FOR A DETAILED ANALYSIS; WE NEED THIS UP ASAP!" You can do the analysis now or do it later; you can't skip it. Somehow it seems more reasonable to do the analysis work in analysis, rather than in testing or operation.



# How to Fail Less and Enjoy More

by Frédéric Boulanger

If there were a coffee shop where 75 percent of the coffee tasted awful, how many times would you spend your money there? As consumers, we wouldn't put up with that at our local café, so why are we surprised when so many software products are commercial failures?

There are a lot of badly designed software products in the world. Sure, some products are slow, with clunky back ends that waste the users' time. But more importantly, with some products, it's hard for the user to accomplish his everyday tasks. And guess what? Users don't like poorly designed software products. Users will only run these applications at gunpoint or if management forces them—which is pretty much at gunpoint.

High-tech workers tend to love technology for its own sake. But we have to remember that our users don't. In fact, the whole idea of calling the people for whom we develop products “users” is wrong thinking. Calling them “users” of our products is product-centric; it makes the product more important than the person who needs it. No one calls a ditch-digger a “shovel user.” He's a guy with a task to accomplish. The shovel is the tool he uses to accomplish it.

That's how we have to think about what we do. A software application isn't an end in itself. Our task is not producing software; it's helping our customers finish their work faster. Our job isn't writing code; it's giving our customers fast tools to get their jobs done. Even if you make the quickest, shiniest software application in the world and ship it on time and under budget, it's a failure if it doesn't make someone's job easier.

Failures cost us customers, which costs us money. If you had to go to the terrible coffee shop and buy four cups to get one good one, you'd figure out pretty quickly that the one good cup had the price of three failures embedded in it.

Right now, our successful applications

have the price of our failures embedded in them. We've cut costs by outsourcing manufacturing and development. The next wave of software cost cutting will be reducing our cost of failure.

## Developing Products People Love

The flip side of products that fail are the products people *love* to use—and I use the word love deliberately. We need to create applications that people like as much as they like a good cup of coffee.

It's easy to imagine loving a cool product like an iPod or your car's GPS navigation system, but how does that translate to the Web interface of your company's time-tracking software? Could anyone ever love that?

The key in that situation would be to design the product to be simple, user-friendly, and to save time. When it comes to software we have to use, the difference between loathing it and loving it is how much time we save when using it. That's part of the reason why we skip Flash intros on Web sites, no matter how cool. They get in the way of accomplishing the task we need to do on the site. People would hate the iPod's scroll wheel if it didn't do exactly what you imagine it should (if turning it clockwise scrolled left or rewound the song).

## Get to Know the People Who Want Our Software

Creating applications that people love doesn't start with awesome database architecture or an EJB back end. It starts with knowing who your customers are and what tasks they want to accomplish.

If you designed a simple MP3 player that could only play, stop, and scroll through songs, where the titles were shown in a very large font, but the target



market was nightclub DJs, your product would fail. If you designed an MP3 player based on a sound mixing board and the target market was senior citizens, that's another failure.

## Design for Tasks—Not Features

If you read a manual for a word processing application (a good one, anyway),

you won't find a procedure for “Using the underlining feature”—that's feature-centric. But you will find a section on “How to underline words”—that's task-centric. This is because manuals are designed to help people accomplish what they need to do, not play around with technology. The people who use our software don't think about “features,” they think about their tasks.

## Design for Simplicity—Not Coolness

We all got into the software business because we love technology. But the people who buy our software don't necessarily love technology. They aren't buying technology; they are buying the tools they need to make their lives easier.

We have to remember this when we design. So when we consider adding a cool feature that showcases some new technology, we have to ask ourselves if it makes our customers' daily life easier or are we adding it because we can?

## One Cup of Coffee

Fewer software failures mean more software successes—we will build products people will love and want to use. We'll save all the time and money we spent brewing bad coffee. In the end, that will make our one cup of coffee taste so much sweeter. **{end}**



**More time...  
Less stress...**  
*eLearning is the  
perfect solution  
for training at  
your own pace.*



**TAKE A FREE DEMO TODAY! VISIT [WWW.SQE.COM/EMTD](http://WWW.SQE.COM/EMTD) FOR MORE INFORMATION.**

**Try our self-paced eLearning course  
delivered right to your desktop.**

Travel from your desktop with Software Quality Engineering's eLearning course, eMastering Test Design. Experience classroom value with the convenience of self-paced instruction on the Web. Because people learn in different ways and everyone learns best with multiple learning options, we've combined audio, video, text, graphics, examples, questions, exercises, and additional learning resources for the best Web-based delivery possible.

## **eMastering Test Design**

### **Classroom Value with the Convenience of Self-paced Instruction**

- Instructed and designed by two of our most experienced instructors, Lee Copeland and Rex Black
- Same valuable information as the 2-day classroom course

### **The Perfect Solution for Test Practitioners with Travel and Time Constraints**

- Complete an eLearning course from your own desktop
- Expert mentors provide answers to your specific questions

### **Course Content**

- Superior lesson material developed and delivered by testing experts
- Tutorials that place content into real-world situations
- Exercises that immediately apply your new learning
- Assessment questions that help you evaluate your learning
- Questions linked to content to reinforce your learning
- Video and audio clips enhance your learning experience
- Web access to an extensive list of additional resources
- Hyperlinks to a glossary of terms used in the course

## **Index to Advertisers**

Aculis	<a href="http://www.aculis.com">www.aculis.com</a>	45
Agile 2008 Conference	<a href="http://www.Agile2008.org">www.Agile2008.org</a>	40
AutomatedQA	<a href="http://www.testcomplete.com/try">www.testcomplete.com/try</a>	Inside Back Cover
Blackbaud, Inc.	<a href="http://www.blackbaud.com">www.blackbaud.com</a>	19
Codenomicon	<a href="http://www.codenomicon.com">www.codenomicon.com</a>	Opposite Page 16
Cognizant	<a href="http://www.cognizant.com/testing">www.cognizant.com/testing</a>	35
Empirix	<a href="http://www.empirix.com">www.empirix.com</a>	1
Hewlett-Packard	<a href="http://www.hp.com/go/software">www.hp.com/go/software</a>	42
Hewlett-Packard	<a href="http://www.hp.com/go/software">www.hp.com/go/software</a>	Back Cover
IBM	<a href="http://www.ibm.com/rational">www.ibm.com/rational</a>	6
Klocwork	<a href="http://www.klocwork.com">www.klocwork.com</a>	Inside Front Cover
Mountain Goat Software	<a href="http://www.mountaingoatsoftware.com">www.mountaingoatsoftware.com</a>	13
Pragmatic Software	<a href="http://www.SoftwarePlanner.com">www.SoftwarePlanner.com</a>	16
PNSQC	<a href="http://www.pnsqc.org">www.pnsqc.org</a>	44
Rally Software	<a href="http://www.rallydev.com/bsm">www.rallydev.com/bsm</a>	12
Seapine	<a href="http://www.seapine.com">www.seapine.com</a>	2
SQE Agile Development Practices	<a href="http://www.sqe.com/Agiledevpractices">www.sqe.com/Agiledevpractices</a>	17
SQE eLearning	<a href="http://www.sqetraining.com/eLearning">www.sqetraining.com/eLearning</a>	48
SQE STF Training	<a href="http://www.sqetraining.com/Certification">www.sqetraining.com/Certification</a>	41
STARWEST 2008	<a href="http://www.sqe.com/StarWest">www.sqe.com/StarWest</a>	5
Target Process	<a href="http://www.targetprocess.com">www.targetprocess.com</a>	21
TechExcel	<a href="http://www.techexcel.com">www.techexcel.com</a>	10
Telelogic	<a href="http://www.telelogic.com">www.telelogic.com</a>	Opposite Page 8
Web Performance, Inc.	<a href="http://www.webperformance.com">www.webperformance.com</a>	16

### **Display Advertising** **Shae Young [young@sqe.com](mailto:young@sqe.com)**

### **All Other Inquiries** **[info@bettersoftware.com](mailto:info@bettersoftware.com)**

*Better Software* (USPS: 019-578, ISSN: 1532-3579) is published ten times per year. Subscription rate is US \$49 per year. A US \$35 shipping charge is incurred for all non-US addresses. Payments to Software Quality Engineering must be made in US funds drawn from a US bank. For more information, contact [info@bettersoftware.com](mailto:info@bettersoftware.com) or call (800) 450-7854. Back issues may be purchased for \$15 per issue (plus shipping). Volume discounts available. Entire contents © 2008 by Software Quality Engineering (330 Corporate Way, Suite 300, Orange Park, FL 32073), unless otherwise noted on specific articles. The opinions expressed within the articles and contents herein do not necessarily express those of the publisher (Software Quality Engineering). All rights reserved. No material in this publication may be reproduced in any form without permission. Reprints of individual articles available. Call for details. Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to Better Software, 330 Corporate Way, Suite 300, Orange Park, FL 32073, [info@bettersoftware.com](mailto:info@bettersoftware.com).

# Checking Automated Testing Prices?



## TestComplete™

Sensible price. Superior automated testing.

Get superior automated testing software for a fraction of what the big guys want to charge you. TestComplete is an award-winning testing solution that gives you functional, regression, unit, load, client server testing and more in one low priced package. It tests any of your Windows applications: Desktop or Web, C++ or VB, .NET or Java, Flash or WPF. Don't throw your money away on overpriced software that does less than TestComplete. Download the free trial of TestComplete now!

✓ Award-Winning Features

✓ Easy Record & Playback

✓ Great Object Recognition

✓ Unlimited Extensibility



FREE TRIAL - DOWNLOAD NOW  
[www.testcomplete.com/mad](http://www.testcomplete.com/mad)

AutomatedQA  
test, debug, deliver!

(978) 236-7900





ALTERNATIVE THINKING ABOUT QUALITY MANAGEMENT SOFTWARE:

## Make Foresight 20/20.

Alternative thinking is "Pre." Precaution. Preparation. Prevention.  
Predestined to send the competition home quivering.

It's proactively designing a way to ensure higher quality in your  
applications to help you reach your business goals.

It's understanding and locking down requirements ahead of  
time—because "Well, I guess we should've" just doesn't cut it.

It's quality management software designed to remove the  
uncertainties and perils of deployments and upgrades, leaving  
you free to come up with the next big thing.

Technology for better business outcomes. [hp.com/go/quality](http://hp.com/go/quality)

