September/October 2011 $9.95 www.StickyMinds.com

BETTER SOFTWARE™

A TECHWELL™ PUBLICATION

THE CASE FOR A BUSINESS CASE
No MBA required.

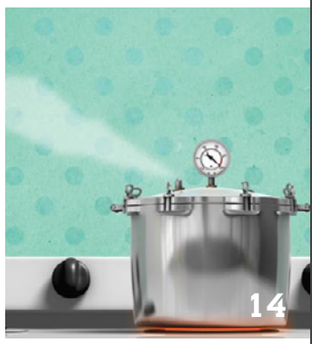
DO YOU SPEAK JVM?
No Java? No worries!

Testing Under Pressure

BETTER SOFTWARE
A TECHWELL PUBLICATION
Volume 13, Issue 5 • SEPT/OCT 2011

19

CONTENTS



features

14 COVER STORY
TESTING UNDER PRESSURE
A cast-in-concrete delivery date looms on your project's horizon. You have precious little time remaining, and the development team keeps delivering incomplete builds of unstable code. Is this a "death march" project, or can the testing team actually do something useful, or perhaps even save the day?
by Robert Sabourin

19 **ALTERNATIVE JVM LANGUAGES FOR JAVA PROJECTS**
Java Virtual Machine has become a successful platform for applications written in many languages, not just Java. Alternatives like JRuby, Scala, Clojure, and Groovy can be more concise and offer new ways to approach problems.
by Daniel Wellman

23 **SOFTWARE PROJECT MANAGERS: KNOW YOUR BUSINESS CASE**
Many professionals in the software industry chose to pursue software to avoid business schools and MBAs. In this article, Payson explains that some of that "Business BS" can be useful both tactically and strategically to software project managers.
by Payson Hall

BETTER™ SOFTWARE
A TECHWELL PUBLICATION

Publisher
Software Quality Engineering, Inc.

President/CEO
Drew Thoeni

Vice President of Publishing
Holly N. Bourquin

Editor in Chief
Heather Shanholtzer

Editorial

Managing Technical Editor
Lee Copeland

Online Editor
Joseph McAllister
Jonathan Vanian

Community Manager
David DeWald

Design

Creative Director
Catherine J. Clinger

Advertising

Director of Sales
Sonia Lavin

Sales Consultants
Daryll Paiva
Kim Trott

Customer Success Manager
April Evans

CONTENTS

columns

8 **TECHNICALLY SPEAKING**
GOODHART'S LAW • *by Lee Copeland*
Charles Goodhart stated: "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." In other words, "When a measure becomes a target, it ceases to be a good measure."

9 **INSIDE ANALYSIS**
MANAGING IN FLUID ENVIRONMENTS • *by Rick Brenner*
Most management and change management methodologies assume a traditional environment—one in which the time between changes is much greater than the time required to adapt to each change. In fluid environments, the next change event happens before we can finish adapting to the last one, and sometimes even the one before that.

32 **CAREER DEVELOPMENT**
LEARNING FOR AGILE TESTERS, PART 2
by Lisa Crispin and Janet Gregor
In part one of our Learning for Agile Testers series, we adressed general "thinking" skills that go beyond technical competence and how learning these enhances the value you contribute. In part two, we discuss some specific technical skills that benefit testers and how to acquire them.

in every issue

4 Mark Your Calendar
6 Contributors
5 Editor's Note
11 Virtual Resource Shelf
12 Executive Interview
27 Product Announcements
31 FAQ
34 Ad Index

MARK YOUR CALENDAR

SQE TRAINING
www.sqetraining.com

software tester certification

www.sqetraining.com/certification

September 19–21, 2011
Washington, DC

September 20–22, 2011
Atlanta, GA
Toronto, ON

September 27–29, 2011
St. Louis, MO
Pittsburgh, PA

October 2–4, 2011
Anaheim, CA

October 4–6, 2011
New York/New Jersey

October 11–14, 2011
Austin, TX
Chicago, IL

October 17–19, 2011
San Francisco, CA

October 18–20, 2011
Raleigh, NC
Philadelphia, PA

October 25–27, 2011
Cincinnati, OH
Vancouver, BC

training weeks

www.sqetraining.com/testing

Software Testing Training Weeks

September 19–23, 2011
Washington, DC

October 17–21, 2011
San Francisco, CA

November 14–18, 2011
Tampa, FL

conferences

STAR*WEST* 2011
Software Testing Analysis & Review
www.sqe.com/starwest
October 2–7, 2011
Disneyland Hotel
Anaheim, CA

Better Software Conference East
Agile Development Practices East
www.sqe.com/bsceast
November 6–11, 2011
The Rosen Centre
Orlando, FL

STAR*EAST* 2012
Software Testing Analysis & Review
www.sqe.com/stareast
April 15–20, 2012
The Rosen Centre
Orlando, FL

Better Software Conference West
Agile Development Practices West
www.sqe.com/bettersoftwarewest
June 10–15, 2012
Caesars Palace
Las Vegas, NV

BETTER™ SOFTWARE
A TECHWELL PUBLICATION

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 six issues.

Visit www.BetterSoftware.com or call 800.450.7854.

SOFTWARE QUALITY ENGINEERING · ASBPE American Society of Business Publication Editors · fma

CONTACT US
Editors: editors@bettersoftware.com

Subscriber Services:
info@bettersoftware.com

Phone: 904.278.0524, 888.268.8770

Fax: 904.278.4380

Address:
Better Software magazine
Software Quality Engineering, Inc.
340 Corporate Way, Suite 300
Orange Park, FL 32073

PRINTED WITH SOY INK

BUILDI

Recently,
always s
editors a
for our re
issues.

This year
publishin
more per

It's a tim
presence
knowled
content i

Another
Well, wit
ologies y

Please lo
minutes
audience

Speaking
managin
Machine

As alway
to check
ter Softw

Happy re

hshanhol

MS-TECH CORNER

Visual Studio Quality Assurance and Testing Tools: The Business Case

Reproduced with permission from Pique Solutions.

BUSINESS CASE SUMMARY

The business case for Microsoft Visual Studio 2010 quality assurance and testing tools demonstrates the compelling value along three key dimensions —economic benefits, business benefits, and strategic benefits. In terms of the business case, these are defined as follows:

- **Economic:** Demonstrates the judicious use of "scarce" financial resources relative to other potential investments. Shows a compelling ROI for investment in **Microsoft Visual Studio 2010 Ultimate and Microsoft Visual Studio 2010 Test Professional** based on the *tangible cost savings* associated with the acquisition and usage of the quality assurance and testing capabilities over a three-year period.
- **Business (Output and Quality):** Measures important, tangible metrics for improvements in software development processes and software quality, particularly related to Agile development practices and goals. These benefits may not explicitly map to tangible cost savings or return on investment, but they are often more important.
- **Strategic:** Highlights the impact on strategic elements of development and testing and the overall alignment with key business objectives.

Table 1 presents a summary of the business case based on the composite data collected through primary research interviews conducted by Pique Solutions. It is important to note that the economic benefits account for only a portion of the developer and tester productivity and efficiency savings associated with the use of Visual Studio quality assurance and testing tools.

Table 1: Business Case Summary

Economic	Business (Output and Quality)	Strategic
154% return on investment* over a three-year period, based on tangible cost savings/value: ■ Solution acquisition costs/savings ■ Development and test process savings ■ Management and administration savings ■ Post-production release savings *Based on a $526,000 net present value (NPV) on an investment of $342,000 over three years.	■ 91% increase in the amount of bugs/defects found and fixed per development cycle ■ 38% increase in code coverage ■ 28% reduction in new developer/tester "ramp-up" time ■ 26% reduction in the development-to-test cycle time; results in a 14% reduction in production release cycle time ■ 11% reduction in post-release maintenance, patches, and hot-fix items required	■ Better collaboration among developers, testers, and IT management ■ Enablement of Agile development practices ■ Higher/more predictable software quality ■ Increased visibility into the development process

THE ECONOMIC AND BUSINESS BENEFITS

An important aspect of the overall business case involved the quantification of the interlinked economic and business benefits of the Visual Studio 2010 quality assurance and testing solution as evidenced by customers and partners who had significant historical deployments with the tools.

To capture that information, Pique Solutions engaged in an in-depth, multiphase, data-collection process. The process involved initial and follow-up interviews with participants and the use of a data-collection spreadsheet instrument that included inputs for 70 unique quantitative data elements, along with a provision for qualitative descriptions of the responses. The questions and entries spanned the lifecycle of the deployment, including the acquisition of the solution (and benefits associated with acquisition), the impact on development and testing processes, management and administrative activities, and, ultimately, the impact on production software releases. The composite profile used for the economic and business benefit analysis, based on all of the companies interviewed in the study, is provided in **Table 2.** It is important to note that the composite profile corresponds to a typical team in the industry, which in

Table 2: Composite Profile for Business Case

Average Number of Test Staff	17 (11 Visual Studio Test Professional; 6 Visual Studio Ultimate)
Average Number of Developer Staff (directly involved in testing)	14 (Visual Studio Ultimate)
Average Total Size of Developer Staff	55
Average Number of Active Development Projects	8
Average Major Release Cycle	6 months
Previous System for Quality Assurance and Testing	Migrated from manual process
Acquisition Method for Visual Studio 2010	MSDN subscription

many cases is representative of a subset of the broader development team in the larger companies interviewed.

Based on the composite profile, an analysis was conducted of the return on investment of the Visual Studio quality assurance and testing solution—specifically, the net present value of the costs and tangible benefits achieved, on average, by the interview participants. The summary ROI analysis is presented in **Table 3**, demonstrating a 154% ROI with a payback of 17.75 months.

The investment of $342,000 shown in **Table 3** includes the three-year MSDN subscription with Visual Studio 2010 Ultimate and Test Professional, as well as the implementation and training costs indicated by the aggregate data provided from the research participants.

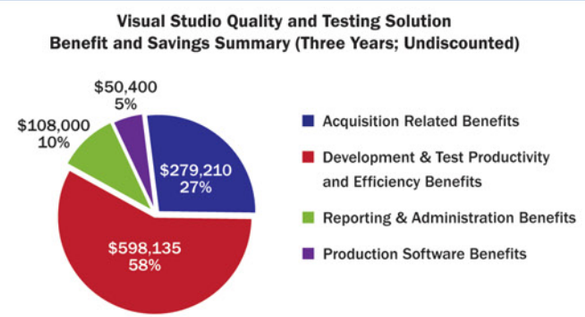
Figure 1 provides a breakdown of the undiscounted, three-year benefits by category for the Visual Studio quality assurance and testing solution. The solution acquisition benefits, specifically relating to MSDN subscription benefits and the value of the Load Test Virtual User Pack product included with Visual Studio Ultimate, are significant at 27%.

It should be noted that the acquisition-related benefits do not include the substantial savings associated with the retirement of software licenses or the termination of support contracts for those interviewee participants who migrated from other quality assurance and testing tools such as HP Quality Center (which represented only 21% of the respondents). The majority of the benefits and savings (58%) come in the form of the reduced or avoided costs associated with the productivity and efficiency of development and testing staff. That being said, those who did migrate from HP Quality Center did achieve substantive benefits in terms of developer and tester productivity and efficiency, and that data is factored into the economic analysis.

Table 3: ROI Summary	
Quality and Testing Return on Investment: ROI and NPV (3 Yr. Analysis)	
Total Visual Studio Value/Cost Savings (PV*)	$867,946**
Total Investment in Visual Studio MSDN Subscriptions (PV*)	$342,071**
Total NPV*	$525,875**
Payback (Months)	17.75
ROI for Visual Studio	153.7%

*Present value (PV) calculations are based on a discount rate of 6%. **U.S. dollar/exchange rate = 1.

Figure 1. Benefits and Savings by Category



SUMMARY OF STRATEGIC BENEFITS

The final element of the business case sought to understand the strategic benefits of the Visual Studio quality assurance and testing solution, in an effort to find relevance for a broader IT or even business strategy. In our data-collection exercise, we presented a list of eight strategic benefits and asked respondents to rank the top three benefits. The synthesized results for the top three stragic benefits are listed in **Table 4**.

Collaboration was by far the top choice across the interviewee population and is very consistent with the economic and business benefits that the study brought to light. The ability for developers and testers to work much more closely together, on the same platform and from the early stages of development, drives most of the benefits quantified in the study.

Table 4: Strategic Benefits of Visual Studio 2010 (Top 3 Based on Stack Ranking)	
Rank	Strategic Driver
1	Better collaboration among development, testing, and IT management
2	Enablement of Agile development practices
3 (tie)	Higher/more predictable software quality *and* Increased visibility into the development process

Pique Solutions is a marketing consulting and global market research firm working with large technology vendors in the United States and the European Union. Visit www.piquesolutions.com to learn more about Pique Solutions' consulting and market research services.

CLOSE WHITE PAPER

Microsoft

NG CONTENT AND COMMUNITIES

, I attended the Florida Magazine Association's annual publishing conference. It's uch a treat to go, because I rarely have the opportunity to talk face to face with other nd publishers about the challenges we encounter, such as creating the best content eaders, surviving a recession, developing an audience, and navigating ethical and legal

, almost every speaker emphasized the importance of making social media a part of your g strategy: Give your readers a place to connect and discuss issues that matter to them. Interact on a sonalized level with Twitter and Facebook. Invite your subscribers to become part of a *community*.

ely message for us at Software Quality Engineering, as we are growing TechWell.com, our newest online , and want to encourage all *Better Software* magazine readers to join and become actively involved in ge sharing and helping your peers and your teams build high-quality software. Membership is free and the s invaluable, so what's not to love?

take away from the FMA conference was "Give your readers what they want." Sounds pretty simple, right? thout your input, we can only guess what topics you think are important and what technology and method- ou want to learn more about.

ok for a reader survey in your inbox soon, if you haven't already received one. It will only take a few to complete, and the feedback you provide will be crucial to our delivering the right content to the right (YOU).

of content … we have some great articles in this issue covering topics including the challenges of g in a fluid environment, testing in turbulent projects, the variety of languages that run on Java Virtual , and the importance of understanding your project's business case.

s, I hope you find much in this issue to help you improve and facilitate your software projects. Don't forget your inbox for our reader survey. I look forward to learning more about what you want and need from *Bet- vare* magazine.

ading,

Shankletzer

ltzer@sqe.com

Building Content and Communities

Recently, I attended the Florida Magazine Association's annual publishing conference. It's always such a treat to go, because I rarely have the opportunity to talk face to face with other editors and publishers about the challenges we encounter, such as creating the best content for our readers, surviving a recession, developing an audience, and navigating ethical and legal issues.

This year, almost every speaker emphasized the importance of making social media a part of your publishing strategy: Give your readers a place to connect and discuss issues that matter to them. Interact on a more personalized level with Twitter and Facebook. Invite your subscribers to become part of a *community*.

It's a timely message for us at Software Quality Engineering, as we are growing TechWell.com, our newest online presence, and want to encourage all *Better Software* magazine readers to join and become actively involved in knowledge sharing and helping your peers and your teams build high-quality software. Membership is free and the content is invaluable, so what's not to love?

Another take away from the FMA conference was "Give your readers what they want." Sounds pretty simple, right? Well, without your input, we can only guess what topics you think are important and what technology and methodologies you want to learn more about.

Please look for a reader survey in your inbox soon, if you haven't already received one. It will only take a few minutes to complete, and the feedback you provide will be crucial to our delivering the right content to the right audience (YOU).

Speaking of content … we have some great articles in this issue covering topics including the challenges of managing in a fluid environment, testing in turbulent projects, the variety of languages that run on Java Virtual Machine, and the importance of understanding your project's business case.

As always, I hope you find much in this issue to help you improve and facilitate your software projects. Don't forget to check your inbox for our reader survey. I look forward to learning more about what you want and need from *Better Software* magazine.

Happy reading,

hshanholtzer@sqe.com

Contributors

Rick Brenner, principal of Chaco Canyon Consulting, works with people needing state-of-the-art teamwork in problem-solving organizations producing complex products and with organizations that want stronger relationships among their people. His interests are personal and organizational effectiveness in abnormal situations, such as dramatic change, enterprise emergencies, and high-pressure projects. His articles and weekly newsletter are available at www.chacocanyon.com.

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. He 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@sqe.com.

Lisa Crispin is the co-author (with Janet Gregory) of *Agile Testing: A Practical Guide for Testers and Agile Teams* and a contributor to *Beautiful Testing*. A tester on agile teams for the past ten years, Lisa enjoys sharing her experiences at conferences and user group meetings around the world. For more about Lisa's work, visit lisacrispin.com.

The co-author (with Lisa Crispin) of *Agile Testing: A Practical Guide for Testers and Agile Teams*, Janet Gregory specializes in helping teams build quality systems. As tester or coach, she has helped introduce agile development practices into companies and has successfully transitioned several traditional test teams into the agile world. Janet is a frequent speaker at agile and software testing conferences in North America, including the STAR conferences.

Contributors

Payson Hall is a consulting project manager for Catalysis Group, Inc. in Sacramento, California. Payson consults on project management issues and teaches project management. Email Payson at payson@catalysisgroup.com. Follow him on twitter.com/paysonhall.

Chris Menegay is the EPM Practice Lead for Imaginet, a consulting and training firm specializing in software development process on the Microsoft platform. He has been working with Project Server and the Visual Studio ALM tools since 2004. Chris has worked with a variety of companies ranging in size from fifty employees up to Fortune 100 companies. He has written white papers and articles on Project Server and Team Foundation Server for MSDN and *MSDN Magazine*. Chris is a Microsoft MVP (ALM) and a Microsoft Regional Director.

With more than twenty-five years of management experience, Robert Sabourin, P. Eng, has managed, trained, mentored, and coached hundreds of top professionals in the field. He frequently speaks at conferences and writes on software engineering, SQA, testing, management, and internationalization. Robert is the author of *I am a Bug,* the popular software testing children's book; an adjunct professor of software engineering at McGill University; and the principal consultant (and president/janitor) of AmiBug.com, Inc. Contact Robert at rsabourin@amibug.com.

Bill Wake is a software coach, author, and a senior consultant with Industrial Logic, Inc. (industriallogic.com). His interests include d Daniel Wellman is a technical lead at Cyrus Innovation (www.cyrusinnovation.com), a leading agile consultancy based in New York, where he leads development projects and coaches teams on adopting agile software development practices. Daniel has more than ten years of experience building software systems and is an expert in agile methodologies, object-oriented design, and test-driven development. Contact Daniel at dan@danielwellman.com.

Goodhart's Law

Connecting rewards and punishments to specific goals can negatively imp
usefulness.

by Lee Copeland | lcopeland@sqe.com

For many years, I have argued against the poor set of metrics we use in IT. Recently, I came across a reference to Goodhart's Law, which explains our problem. Proposed in 1975 by Charles Goodhart—a former advisor to the Bank of England and emeritus professor at the London School of Economics—the law states that once a social or economic indicator is made a target for the purpose of guiding policy, then it will lose the information content that originally made it useful. Goodhart wrote, "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." [1] Professor Marilyn Strathern has restated Goodhart's Law more succinctly and more generally: "When a measure becomes a target, it ceases to be a good measure." [2]

We often see this in the world of IT metrics. Management chooses a metric in an attempt to understand the behavior of a system or process. Later, this same metric becomes a goal ("Attain this value or else"). When this occurs, the "or else" part motivates people to change their behavior to achieve the goal.

A classic metric of this type is "lines of code written per day." When used as a measure, this metric can be valuable for estimating. However, when LOC/day becomes a goal, developers may be enticed to write more lines of less efficient code. For example, a classic Java "for" loop has the generic form:

```
for (initialization; termination; increment) {
    statement(s)
}
```

An example of this loop is:

```
for(int    i=1;    i<1
println("Count is: " +
```

If writing more lines of code
my personal goal), I can write t

```
int i=1;
while (i<11) {
System.out.println("Co
i++;
}
```

In the early days of the
ented paradigm, a number
were proposed as indicators o
of system design. One of th
subclass:superclass ratio—som
the *specialization ratio*. The s
ratio measures the extent to w
class has captured an abstract
value indicates a high reuse
classes. For structures using s
tance, the values range from 1
close to 1 suggest a poor desig
trees are generally considered t
dard class libraries have specia
to 4 [2].

Tech

G

Conn

usef

by Le

For m
we us
Law, v
Good
itus p
states
for th
tion c
"Any
pressu
Maril
and r
ceases

We
choos
systen
tain t
motiv

A c
When
matin
may b
ample

```
for
}
```

pact a metric's

```
1;    i++){System.out.
+ i);}
```

is rewarded (and thus becomes
he equivalent Java as:

```
ount is: " + i);
```

e object-ori-
of metrics
f the quality
ese was the
etimes called
specialization
hich a super-
idea. A large
by the sub-
single inheri-
to ∞. Values
n since deep, linear inheritance
o be poor design. Various stan-
alization ratios ranging from 1

At one company I know, to facilitate the evaluation of object-oriented designs by people who knew nothing about what made a good design, not only was the specialization ratio measured, but a goal was set—the specialization ratio had to be 3 or above. Designs not meeting that goal were sent back for rework. Developers, not liking rework, simply added cleverly disguised empty classes until the ratio was met.

In other cases, rewarding testers for the number of test cases resulted in many poorly written test cases; rewarding testers for the number of bugs they found resulted in a high number of unimportant or duplicate bugs reported; and penalizing testers for bugs rejected by the development staff resulted in important bugs going unreported.

It's not always about the design or code. An example is a metric for the age of trouble tickets (the length of time a ticket was open before being resolved). The organization set a goal of x days or less for a ticket to be open. Since many tickets did not close within the established goal, the service desk manager met the goal by closing and then reopening the tickets, resetting the clock and thus faking the numbers. However, management got what they seemed to want—a report showing that everything was fine.

> **"When a measure becomes a target, it ceases to be a good measure."**

Goodhart's Law reminds us that connecting rewards and punishments to the achievement of specific goals can create unintended consequences. Some will strive to reach those numbers without concern for anything else. If the person being measured is affected by the outcome, she is likely either to lie, thus subverting the usefulness of the measurement, or to focus on what is being measured without regard for the consequences. **{end}**

Goodhart's Law

Connecting rewards and punishments to specific goals can negatively impact a metric's usefulness.

by Lee Copeland | lcopeland@sqe.com

For many years, I have argued against the poor set of metrics we use in IT. Recently, I came across a reference to Goodhart's Law, which explains our problem. Proposed in 1975 by Charles Goodhart—a former advisor to the Bank of England and emeritus professor at the London School of Economics—the law states that once a social or economic indicator is made a target for the purpose of guiding policy, then it will lose the information content that originally made it useful. Goodhart wrote, "Any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes." [1] Professor Marilyn Strathern has restated Goodhart's Law more succinctly and more generally: "When a measure becomes a target, it ceases to be a good measure." [2]

We often see this in the world of IT metrics. Management chooses a metric in an attempt to understand the behavior of a system or process. Later, this same metric becomes a goal ("Attain this value or else"). When this occurs, the "or else" part motivates people to change their behavior to achieve the goal.

A classic metric of this type is "lines of code written per day." When used as a measure, this metric can be valuable for estimating. However, when LOC/day becomes a goal, developers may be enticed to write more lines of less efficient code. For example, a classic Java "for" loop has the generic form:

```
for (initialization; termination; increment) {
    statement(s)
}
```

An example of this loop is:

```
for(int    i=1;    i<11;    i++){System.out.
println("Count is: " + i);}
```

If writing more lines of code is rewarded (and thus becomes my personal goal), I can write the equivalent Java as:

```
int i=1;
while (i<11) {
System.out.println("Count is: " + i);
i++;
}
```

In the early days of the object-oriented paradigm, a number of metrics were proposed as indicators of the quality of system design. One of these was the subclass:superclass ratio—sometimes called the *specialization ratio*. The specialization ratio measures the extent to which a superclass has captured an abstract idea. A large value indicates a high reuse by the subclasses. For structures using single inheritance, the values range from 1 to ∞. Values close to 1 suggest a poor design since deep, linear inheritance trees are generally considered to be poor design. Various standard class libraries have specialization ratios ranging from 1 to 4 [2].

> **"When a measure becomes a target, it ceases to be a good measure."**

At one company I know, to facilitate the evaluation of object-oriented designs by people who knew nothing about what made a good design, not only was the specialization ratio measured, but a goal was set—the specialization ratio had to be 3 or above. Designs not meeting that goal were sent back for rework. Developers, not liking rework, simply added cleverly disguised empty classes until the ratio was met.

In other cases, rewarding testers for the number of test cases resulted in many poorly written test cases; rewarding testers for the number of bugs they found resulted in a high number of unimportant or duplicate bugs reported; and penalizing testers for bugs rejected by the development staff resulted in important bugs going unreported.

It's not always about the design or code. An example is a metric for the age of trouble tickets (the length of time a ticket was open before being resolved). The organization set a goal of x days or less for a ticket to be open. Since many tickets did not close within the established goal, the service desk manager met the goal by closing and then reopening the tickets, resetting the clock and thus faking the numbers. However, management got what they seemed to want—a report showing that everything was fine.

Goodhart's Law reminds us that connecting rewards and punishments to the achievement of specific goals can create unintended consequences. Some will strive to reach those numbers without concern for anything else. If the person being measured is affected by the outcome, she is likely either to lie, thus subverting the usefulness of the measurement, or to focus on what is being measured without regard for the consequences. **{end}**

Managing in Fluid Environments

Successful managers depend on careful planning and risk management. Managing in fluid environments requires its own skill set.

by Rick Brenner | rbrenner@chacocanyon.com

Managing often entails moving from surprise to surprise while somehow staying almost on track. On good days, we can avoid sacrificing our goals.

And then there are the other days. Most people now work in environments that can best be characterized as fluid, because they are subject to continual change. In fluid environments, no status quo endures for long.

Perhaps a haiku best captures the troubles of traditional managers in fluid environments:

We prepared for change.
We had alternative plans.
How did we miss that?

Although most changes are predictable in the large, they are rarely predictable in detail. For instance, in the short term, we'll probably experience a hiring freeze, a capital freeze, a contractor price increase, or unanticipated turnover. But we probably can't predict which of these will happen, when they will happen, or how much impact they will have.

Among other factors, successful management depends on planning carefully, managing risk, closely monitoring current activities, and intervening promptly when corrective action is needed. When we succeed, we meet or exceed projections; but in fluid environments, that standard is often impossibly high.

Mastering change management in itself offers little protection, because most methodologies assume a traditional environment in which the time between changes is much greater than the time required to adapt to each change. In fluid environments, the next change event often happens before we can finish adapting to the last one.

The problem is actually even worse. In fluid environments, we know little about coming changes. Preparing doesn't help much, because we don't know what will happen next and the possibilities are endless.

> "In fluid environments, the next change event often happens before we can finish adapting to the last one."

But there is much we can do. Following are four recommendations for managing in fluid environments.

Know Your Situation

In traditional environments, we can wait for official announcements about coming changes. In fluid environments, we cannot wait. We need what the military and intelligence communities call situational awareness—the perception, comprehension, and temporal projection of relevant environmental elements. Situational awareness tells you what's happening around you and how your own actions affect overall outcomes.

Although there are sophisticated theories of situational awareness [1], a few simple guidelines suffice for us:

- Build and maintain your personal network.
- Encourage subordinates and teams to do the same.
- Monitor relevant world news with Google alerts and social media.
- Attend organizational social functions.
- Systematically collect and record situational data.
- Use, but don't rely on, official information channels.

Master Multi-threaded Planning

Most plans are weak in risk management, but even when we do address risk, our plans are usually single-threaded. That is, we proceed along one line of action, switching to another only when necessary.

In traditional environments, single-threaded plans usually suffice. The delays involved in switching to another thread are acceptable. In fluid environments, any delay can threaten the entire effort. Alternate threads must be "warmed up" for ready use.

The US Air Force used multi-threaded planning to develop the air-launched cruise missile, commissioning both Boeing and General Dynamics to develop entries for a fly-off that Boeing eventually won [2].

Concerns about duplication in multi-threaded planning are real—but often shortsighted. When all goes well, single-threaded planning is less costly. But when troubles arise, multi-threaded planning limits the revenue losses that result from switching to new lines of attack. Those losses can be dramatic, because being late to market often leads to failure.

Master Change

Mastering change begins with understanding how people change. Of the dozens of change models, I've found the one developed by family therapist Virginia Satir to be the most useful for understanding how people deal with change [3].

The Satir Change Model contains six elements:

Old Status Quo

The initial state, before the change cycle begins. Example:

September 10, 2001

Foreign Element

The incident or new information that disrupts the Old Status Quo. Example: the events of September 11

Chaos

The confusion and disruption following the Foreign Element's arrival. Example: September 12 and the months following

Transforming Idea

The concept that gives us a path out of Chaos. Example: the security regime adopted worldwide during 2001

Integration and Practice

How we integrate the Transforming Idea into our operations. Example: the first eighteen months after September 11

New Status Quo

The period following integration, when we continue to enhance performance. Example: where we are now

This model provides three important principles:

- Our preference for Old Status Quo over Chaos is the main source of what many call resistance, which is often little more than attachment to Old Status Quo.
- Chaos is upsetting. Avoid making important decisions during Chaos.
- Integration and Practice take effort. To learn the new ways, people need some relief from normal workloads.

For example, after a reorganization, expect individual performance to drop significantly, albeit temporarily. Avoid making important decisions during Chaos.

For modeling organizational responses to change, I favor a model called OODA, which is an acronym for its four elements: observe, orient, decide, and act. Colonel John Boyd, a Korean War ace fighter pilot in the US Army Air Corps and US Air Force, developed OODA [4, 5, 6, 7]:

Observe

We sense the environment using all available means. Example: A corporation's senior management learns of a hostile takeover attempt.

Orient

We synthesize images, views, and impressions of the world. Example: Management researches the takeover offer and available defenses.

Decide

We select one of many possible responses. Example: Management decides to approach a competitor about a merger.

Act

We execute our decision and return to the beginning of the loop for the next cycle. Example: Management agrees to merge, and the two companies make a public announcement.

When systems interact, they traverse their OODA loops. OODA offers three important principles for fluid environments:

- Success depends on cycling through your OODA loop rapidly enough to stay ahead of the situation.
- If you have an opponent, getting inside your opponent's OODA loop gives you a decisive advantage.
- You can slow your opponent's OODA cycle by means of disinformation and by disrupting your opponent's ability to understand the situation.

Master Logistics and Force Protection

Logistics is the management of materials and services from acquisition up to, but not including, application. In organizations, force protection ensures staff availability, despite pressures for reassignments and terminations.

In fluid environments, mastering logistics and force protection requires high levels of situational awareness, multi-threaded planning, applying the Satir Change Model, and applying Boyd's OODA model.

As an engineering manger, I applied situational awareness and the Satir Change Model to implement force protection. One day, a friend in finance told me that the current quarter's revenues would be disappointing. Shortly after that, all managers were directed to review their subordinates. I immediately recognized that downsizing was likely, because performance reviews were normally anniversary-based, and that led me to make adjustments that protected our staff.

To apply situational awareness and multi-threaded planning to logistics, don't wait for the one-hundred-year flood to begin ordering from a flood-safe manufacturer. To ensure that an alternate manufacturer understands your needs, order regularly from an alternate. To aid with situational awareness, set a Google alert for the primary manufacturer anded with "flood" and the geographical location in question.

Looking Ahead

Organizational environments are likely to become even more fluid. Fluidity is probably dimensional, because the degree of fluidity can be different for organizational structure, demographics, strategic goals, business focus, and more. Since an organization can be more fluid in one dimension than another, shear stresses can arise, limiting organizational success.

Leaders who master management in fluid environments and who recognize the need to adapt to fluidity in all dimensions of the business are the most likely to endure. **{end}**

For more on the following topic go to www.StickyMinds.com/bettersoftware.
■ Resources

Visit **TechWell.com** to comment on this article

Virtual **Resource** Shelf

Author recommended books, blogs, gadgets, websites, and other tools for building better software

Q: What tools do you use to manage your personal productivity?

Gmail—I use the "starred" feature to prioritize important tasks. I also organize most of my teaching and consulting schedules using the Gmail calendar.

Mind mapping (I use Freemind and Mind Manager)—I mind map to solve problems, plan my work, organize my thoughts, and take notes. I also use mind maps to guide discussions and to whip up visual models to help when designing tests.

Spreadsheets—Everything else seems to fit neatly into a spreadsheet. Spreadsheets help me evaluate multiple options and explore what-if scenarios.
–Rob Sabourin

A Franklin Day Planner and a purple pen.
–Lee Copeland

I use a combination of a task- or kanban-style board and a good-old "to-do" list. In my home office, I have a white laminate cabinet and I put sticky notes with tasks in "to-do," "wip," and "done" columns. I keep a "to-do" list in a text document on my laptop, so I have it around in case I'm not in the office or at home. If there's something important for the day I want to be sure to remember, I put a sticky note on my laptop. And for chores around the house, we put sticky notes on the microwave door!
–Lisa Crispin

I've recently found Nudgemail, an email reminder service, to be particularly helpful. If I receive an email message and I don't think I'll be able to respond to it right away, I'll forward it to Nudgemail with a future reminder date, e.g., 4 p.m., Thursday, two weeks, etc. I find this helps keep my inbox clean and removes some worries from my mind so I can focus on other tasks at hand.
–Daniel Wellman

I use iCal in combination with Applescript to set-and-forget programmable tasks. For instance, I publish a daily tip through an RSS feed. Triggered by an iCal alarm, iCal fetches the daily update from a database and posts it to my RSS feed. iCal also emails appointment reminders to those of my coaching clients who need reminding.

Brian Dunagan's Remind Me Later, a free Mac app, lets me type a calendar event description in normal conversational English. It then posts the event to iCal. For example, typing "Dinner w the Obamas today 7 pm 10 pm at WH" does the right thing.
–Rick Brenner

Q&A

Executive Interview
Sam Guckenheimer
MICROSOFT VISUAL STUDIO

interview by Chris Menegay

Sam Guckenheimer, product owner for Microsoft's Visual Studio product line, is in charge of "everything that has the Visual Studio name on it." Sam recently met

Q: What does a product owner do?

A: "Chief customer advocate": My job is roughly 50/50 external and internal. I think in terms of where the product line needs to evolve. How do we balance the interests of different customers and the different business priorities that we have? How do we delight our customers, grow the community, support new technology, and bring the business forward?

Q: What's the strategy behind Microsoft's ALM tools?

A: I joined Microsoft in 2003, and at that time, Microsoft had the world's most widely used individual development environment. The vision that drew me to join was the opportunity to produce the world's best team development environment. We've tried to create a product line that spans a productive team. Not to focus on individual developer activities, but to think about the entire process from backlog to delivery as one total flow done by a collaborative team. That functionality is something that we put into the different products. For more insight, I describe these principles in my new book, *Agile Software Engineering with Visual Studio, from Concept to Continuous Feedback*.

Q: How does Team Foundation Server (TFS) relate to Visual Studio?

A: TFS is the server and the hub for the team. The basics are source control, work tracking, build automation, test management, and test lab management. The backlog is managed in TFS; build automation, continuous deployment into the test lab, and the test management are in TFS. You access that through the different clients.

Q: With the release of Visual Studio 2010 Test Professional, Microsoft has formally entered the QA tools space. Why?

A: The players in the test tools space had grown up thinking of testing as an isolated activity. We focused on the interaction between the tester and developer, so that bugs would get fixed. We would not be looking at testing as something that is done in isolation, but we would be looking at it as an integral part of what a team did in order to

Executive Interview CONTINUED
Sam Guckenheimer

Q&A

improve the software that the team delivered to the end customer. We thought it was important to think about test within the context of the whole delivery of software.

Q: What do you think is something you've nailed?

A: Developer/tester interaction is one area where, although we have more work to do, we made a huge step forward. We introduced six mechanisms to facilitate the fixing of every bug that gets filed. With every bug, you can automatically have a full action log of everything that was done by the tester leading up to the bug being filed: You have full motion video, automatic configuration reports, screenshots captured, an IntelliTrace log of the activity that happened on the servers, and snapshots of virtualized labs that capture the server machines in the state of failure when the bug was seen.

Q: With TFS, what's the easiest thing for customers to get value out of?

A: TFS lets you get started right away and covers the full lifecycle from backlog to deployment. The greatest thing is that you can provision a team of any reasonable size in under an hour to get started, and in a day to be productive with continuous integration, test management, and a fully functional collaborative workflow that lets your testers and developers work together to apply modern agile practices. So you can be productive day one, but you do not run out of value as you keep going.

Q: What features exist in Visual Studio Ultimate that people have but likely aren't using?

A: I'm always amazed that frequently people don't know that with Ultimate they have all of Visual Studio Test Professional, a fully capable product for exploratory testing. They can do all of the capture I described before, reporting fully actionable bugs. They can use this in conjunction with the test lab management with TFS. We provide unlimited load testing. We think load testing and performance improvement should be done all along during development. Visual Studio Ultimate gives you the ability to drive an unlimited number of virtual users with no additional licensing. This is how we test properties like Microsoft.com, and millions of users hit these websites.

Q: What's next?

A: Team Foundation Services. We're taking the best hub for team development and making it available from the cloud. It's now in public preview. Teams can just sign up with their Windows Live ID to get an account provisioned in seconds. You can be up and running with your TFS and can plug into that from the web, any Visual Studio edition, and from Eclipse.

If you are involved in testing software, then we've probably shared the same pain. Like you, I have tested under pressure, without much time, and with severe resource constraints. I have managed to test in turbulent projects within many different planned or agile lifecycles. In order to succeed, I stick to my core values and apply five key principles:

1. Purposeful testing: Begin with the end in mind
2. Active context listening
3. Flexible decision-making workflows
4. Ruthless triage of requirements, testing, and bugs
5. Always know the last best build

Here are examples of how each principle has been applied in real projects.

Beginning with the End in Mind, Next-generation Rendezvous

The first principle—"Purposeful testing: Begin with the end in mind"—addresses software engineering's fundamental question: How do we know we are finished? An understanding of the project's purpose is fundamental to focusing development and testing efforts.

Rendezvous, an example that demonstrates this principle, was a "death march" project. The fate of the company was in play. More than 200 existing customers had received government grants to purchase Rendezvous. The product had to be operational by a fixed date or clients would be penalized and forced to return the grants.

Development had been going on for well over a year with a team of five programmers, a development lead, and a product manager. The development team and testing teams were set up and staffed. They were expected to collaborate using an iterative, feature-driven development strategy, progressively adding features to the product and delivering the work in progress to the testing team. The trouble was that the development team could not even get one build ready to deliver to the testing team. How could we test something that developers couldn't even build?

A closer look at the situation revealed some interesting characteristics. Next-generation Rendezvous was being developed as a general-purpose product. The previous version had been a collection of more than 200 different custom configurations. Customers had their own private code base. They paid for the customizations on a case-by-case basis. This was a lucrative development model, and the customers loved it. The developers loved it. They were treated as if they were knights in shining armor. The current product management goal,

however, was to develop a single product for the entire market.

Next-generation Rendezvous had hundreds of menus, dialogues, and controls. Testing and development were building the system on a feature basis. Due to the unstable code and the feature-based test focus, it would have been impossible to meet the mandated delivery dates.

I studied the project from many perspectives. I met dozens of people familiar with Rendezvous. I spoke with many customers, operators, support staff, trainers, and other subject matter experts.

It was clear that the development and testing efforts needed to be refocused. To that end, I introduced a usage-scenario-based testing approach, identifying twenty-seven critical Rendezvous usage scenarios. I set up a small test team comprising a test guru and several subject matter experts with field experience. Weekly builds were delivered to the testing team. Testing of each build focused on walking through the twenty-seven usage scenarios with sanitized real data. Progressively, the developers stabilized the code on a scenario basis. The testers were able to identify critical bugs that blocked users from getting their jobs done.

A month ahead of the deadline we had pilot installations at several clients, and we successfully released Next-generation Rendezvous on time. In subsequent releases, developers removed superfluous menus, dialogues, and controls. We simplified the product, creating a graceful general-purpose application.

By focusing on how the customer *used* Rendezvous, testing was able to drive product convergence and thus tame a crucial death march project.

How Active Context Listening Saved the Communiqué Project

The second principle—"Active context listening"—is geared at understanding how shifting business, technical, and organizational context drivers influence testing priorities. Active context listening is a dynamic and continuous process of identifying opportunities to offer value. Like magic, testing can change from a bottleneck to a project's critical success factor.

The Communiqué testing team was going nuts. Audrey, the SQA manager, couldn't believe it. Build after build was exposing ugly bugs. The server crashed. The automated test suite could only run a few cases before choking. Memory leaks, viruses, and flakey OEM patches were a daily drama. The development manager shifted the best developer onto a different project. Midway through, the vice president of research and development assigned the project to me with a clear directive: Converge the project and deliver a commercial release before our next board meeting.

Why did we need the release? Survival! We needed continued financial support from private investors. They would pull the plug if we could not spin their capital into a commercial product. The upcoming board meeting was our last chance to prove that we were not wasting the investors' money.

My first step was to identify as many project context drivers as I could find. I looked for sources of information. I looked for sources of wisdom. I looked for business issues, technical issues, and even legal issues. Who were the investors? What were they looking for? What commitments did the company really make? Were any requirements real needs? Which were wishes? Why did the R&D department shift resources away from this critical product?

I arranged short, daily stand-up meetings with all team members. I made baked goods for the team. I listened to them. What was blocking them? How were we progressing? I listened to their fears and frustrations. I discovered that the test team was trying to be diligent. I learned that developers tried to implement solid unit and integration testing but were blocked by weak OEM components. Developers had provided us many testability hooks and a custom embedded test harness, too. I discovered that the only thing the product manager wanted was a simple solution that worked in many environments. I discovered that our board of directors cared a great deal about this project because we needed to obtain additional financing. Investors had been promised a new product release based on the previous investment. If the board could demonstrate our ability to deliver a working solution, the investors would very likely invest more than $2.3 million dollars, which would keep us afloat. I listened to the sales team and found an especially interesting sales lead at the French stock market in Paris.

The mid-project staff shift was caused by several VPs interested in showing investors our new web-based technology. These VPs did not realize that the new project put Communiqué at risk. They assumed that R&D could do both projects.

By actively listening to project context drivers, I devised a winning strategy enabling the testing team members to become project heroes. Instead of targeting all of the environments, why not start with the French stock market environment? Get its operating system, network configuration, even its actual hardware. The testing team was able to work with the customer to capture real test data. The test environment became the project focus. Direct communication channels gave developers and testers direct access to our customer's technical staff. The customer loved the attention, leading to a great reference site generating plenty of solid sales leads.

We were able to stabilize the single configuration and confidently process customer transactions. We delivered a working solution more than a month before the critical board meeting, the company raised additional capital, and the investors were delighted.

Decision Making

The third principle—"Flexible decision-making workflows"—is useful in dealing with organizational politics and avoiding escalation-oriented deployment decisions.

How do we decide what to test and what not to test? What bugs to fix and which bugs to leave? What requirements to focus on and which features to be defer?

I set up the software engineering workflows at MVM, famous for virtual modeling at popular fashion sites. The product managers, architects, development leads, and testers were all involved in the prioritization of features, bugs, and testing objectives. Decision makers had access to the information they needed throughout the project. Delivering a quality product meant delivering value to all project stakeholders. Developers and testers knew specifically how each feature was valued and by whom.

One Friday afternoon, the decision-making workflow broke down. An unexpected visitor arrived at our prioritization meeting. It was an anxious gentleman with a big wad of paper. I did not know who he was or why he was barging into the meeting, but he demanded our immediate attention. "Here is a list of seventeen product issues. Drop everything you are doing. Resolve these seventeen issues, and ship the product. I don't care what you think. Do as I say," he commanded.

This person was the company president, whom I had never seen before. The president held weekly phone calls with each major customer. The last of these weekly calls was on Thursday, and in it he reviewed the customer's key-issue list. The president had no idea how we made decisions. I had no idea that the president kept such great contact with all of the major accounts.

We did what the president demanded: We resolved the seventeen issues. Some were already on our bug list; some were completely new to us. We did nothing else and then we shipped the product. It took more than three months to rework the software development process to deal with all the problems introduced by the president's list.

I learned a valuable lesson: When you are developing software in a highly turbulent context, it is very important to get buy-in and support from executive sponsors and project stakeholders about the mechanism and workflow used to make decisions. Now, I am always on the lookout for stakeholders missed. I need their support in how we make decisions, especially those made on the fly about focusing requirements, tests, and product bugs.

> "On time, on quality, and on budget are meaningless unless you are on purpose. Purposeful testing occurs when everyone involved understands why the software is being developed."

Ruthless Triage

"Ruthless triage of requirements, testing, and bugs" is the fourth principle. Prioritization is like emergency medical triage. For each issue, you need to consider business criticality and technical risks, as well as the available testing skills. Each and every decision is triaged.

Edsger Dijkstra taught that software testing can show the presence of bugs, but it can never demonstrate that a product is bug free. For even the simplest application, there are more possible tests that can be run than there are particles of matter in the universe.

In turbulent projects, there are daily changes in business, technological, and organizational context drivers. Dealing with turbulence and focusing testing necessitates a mechanism for test triage. A small group of people review and evaluate test findings, development progress, bugs, and testing priorities.

I set up the mechanism to enable one short meeting to triage testing issues. We reprioritize our testing based on business knowledge, technical risk, and our findings to date. We decide what to test and what not to test. I assess the benefit and consequence of implementing test objectives. Also, I assess the benefit and consequence of skipping test objectives.

Test objectives are managed as a prioritized heap. Bubbling to the top are test objectives that offer more value to our stakeholders. Sinking down are test objectives that we might skip entirely. Low-priority objectives are assigned less testing effort; high-priority items are tested earlier and given more effort.

During the course of testing, new test ideas are continuously collected and triaged. Often, teters on the job identify the most valuable test objectives. Sometimes, objectives that looked high priority at the start of the project become low priority as the project context is revealed.

The most reliable product I ever worked on was the AVT-710, an embedded system driving a video display terminal. It was a fixed-delivery-date project. We were to replace a weak, failing product in the Middle Eastern market. Unexpected project turbulence was largely due to freight rates, which increased dramatically during the first Gulf War. A few months before release, we refocused our development to the Eastern and Central European markets. Ruthless daily triage was the key to project success. All team members were aware of the project purpose, and we reacted quickly to major changes. The AVT-710 was on the market for more than ten years. There were zero field-reported software defects over the life of the product. The AVT-710 won the *Byte Middle East* Product of the Year award for its reliability. But, the AVT-710 had more than 300 known bugs in it. The success of the product was due to the effectiveness of triage—deciding what to test, what not to test, what to fix, and what not to fix. The message is to ruthlessly and purposefully triage.

Know the Last Best Build

The fifth principle—"Always know the last best build"—comes into play when there is a fixed-delivery-date. When the release date comes up, you want to ship the best build you can. You will know the least about the most recent build, but one

of the penultimate builds may have a reasonable balance of working features and known weaknesses that best meets the project goals.

My first commercial software release was on June 4, 1982. I completed and delivered the 2D graphics package Quarto. In its day, Quarto was pretty slick technology. It allowed applications to drive bus-mounted graphics cards on PDP-11 computers and included support for digitizing tablets and frame grabbers. Quarto supported windowing and menus years before Xerox, Apple, or Microsoft developed their ubiquitous windowing systems.

Every year I celebrate the release of Quarto. In 2007, I took my wife, Anne, on a month-long romantic tour of New Zealand to commemorate the twenty-fifth anniversary of the launch. Of course, Anne does not believe we were celebrating a software release. She thinks we were celebrating something completely different—our twenty-fifth wedding anniversary. We were married on June 5, 1982. Quarto was released the previous day. No, I was not waiting for the Quarto release before marrying Anne, the Quarto release was a fixed-delivery-date release. I was the main developer, the main tester, and the technical writer responsible for the project. Without me, nothing was going to get done.

The Quarto project used daily builds and included automated test harnesses to help regress the application. Every day, we would take a build into the testing environment and exercise it based on our knowledge of what had changed. We kept a simple log of the product features. We had a lot of interesting regression bugs due in part to the harsh environment with very limited amount of memory available and high-speed responses required. We implemented such risky programming practices as self-modifying code in order to optimize line-drawing and area-filling primitives. When a modification was made to one part of the software, it could risk breaking some other part.

Every build was assessed and scored on each functional area or testable object. We used four grades: confident, no knowledge, partial knowledge, or blocked. Every day, I was able to review the stability and status of all the builds ever made. When hitting the fixed-delivery-date, I did not ship the last build, which was the build I knew the least about. Instead, I decided to ship the last best build.

Shipping the last best build has let me meet many fixed-delivery-date releases in many development contexts, especially those with regulated or contractually binding release targets.

The Five Pressure Principles in Action

On time, on quality, and on budget are meaningless unless you are on purpose. Purposeful testing occurs when everyone involved understands why the software is being developed. When you are well informed, you are able to make better microscopic decisions when testing and better macroscopic decisions when planning your test strategy. When you observe an unexpected emergent behavior, your understanding of the project "why" will guide you in determining how much effort you should invest in exploring it. Knowing the project purpose guides bug reporting, helping you describe how the consequences of not fixing the problem might work against the project purpose. Coming up with new testing ideas increases dramatically when the testers are in sync with why they are testing the project. I urge testers to understand the "why" before getting caught up in the tyranny of the urgent. Testing without purpose is often wasteful. Testers should ask why every time they are assigned a test objective.

Active context listening is all about being proactive. There is always a reason for project turbulence. If the testing team can anticipate the change, then it can adapt. If the testing team is the last group to know about the change, it may have wasted a lot of time focusing on aspects of the project that are no longer important.

> "The last thing I want to do when a critical decision is required is start asking how we might make such a decision."

Build relationships with your peers in other departments. Development peers can give you a heads-up on technological changes. Sales peers can help you understand who will be buying the product and what they plan to do with it. Customer support peers can help you understand potential areas of weakness and trends in system usage. The accounting and finance department can help you understand the potential financial impact of the product under development. Living in a restricted testing silo leads to being locked out of important information.

The last thing I want to do when a critical decision is required is start asking how we might make such a decision. Decisions related to prioritizing requirements, tests, and bugs should be set up before you start development. The decision-making process should vary depending on the level of testing. Unit testing issues might be left up to the developer. Integration testing bugs might be prioritized without the input of product management, because there could be a lot of phantom bugs due to the use of stubs and drivers in place of missing code. System test issues might be decided with the combined input of product managers, developers, and testers. Customer-acceptance-testing bugs may be exclusively prioritized by the customer and account manager. Set up the decision-making workflows in advance and then adapt them as required if the context changes.

Triage can be implemented by extending the traditional bug prioritization or changing control board sessions. Ruthless triage means you must come to a decision. To quote Yoda from *The Empire Strikes Back*, "Do or do not. There is no try." As early as

you can, insist that triage meetings come to a conclusion. Encourage people to commit: fix it now, fix it later, or do not fix it. Avoid and eliminate priority levels such as "fix optional" or "fix if we have time." The priority of bugs can be reviewed after important project milestones and should also be reviewed whenever the decision-making workflows change.

As soon as development starts issuing builds to be tested, I start reporting information about the builds. I answer the question "What choices do we have if we were to ship today?" Some builds are stable but are missing features. Some builds work on some platforms but fail on others. Some builds are feature complete but unstable. I report information showing the alternatives we have available if we were forced to ship the product today. In my experience, stakeholders quickly grasp the significance of these reports. I limit my report to the five best builds. My assessment is based on features or capabilities identified by product management.

Testing with these principles helps me focus on what matters and react to change. Applying these principles leads to a dynamic testing experience tied closely to the important business drivers of the project. When operating under extreme time pressure, I am able to consistently focus precious testing resources on what matters the most. **{end}**

Alternative JVM Languages for JAVA Projects

by Daniel Wellman

ISTOCKPHOTO

Wiring the 783rd Spring Bean. Creating the 2,162nd getter-and-setter JavaBean pair. Using the collection utility class to filter a list and losing count of all the curly brackets, angle brackets, and parentheses. Some days, life on a long-lived Java project can be repetitive and tiresome. This is because some concepts in Java require a lot of code and syntax to express, earning the language a reputation as being verbose.

The good news is that in 2011 there are several languages that run on the Java Virtual Machine (JVM), work with existing Java libraries, and have a successful history of deployment in production. This means these languages are stable enough for businesses to trust them for important systems, just like Java. Some of the most widely used alternative JVM languages are Groovy, Scala, JRuby, and Clojure. Each of these languages has commercial support options and a growing community of talented developers, which means you won't be alone if you pick a language and need help. Companies such as Akamai, Foursquare, JBoss, LinkedIn, Netflix, SAP, and Twitter are using these languages.

But choosing another language isn't just about finding some variety in your daily work. Some languages let you express

"The team had started to feel frustration at how much code it took to express certain concepts in Java, especially for manipulating collections of objects."

certain concepts with less code than Java requires. For example, all the alternative languages listed in this article offer closures and functions as first-class objects. These require far less code than Java's equivalent, anonymous inner classes. Other languages offer radically different approaches to problems like concurrency. Scala and Clojure provide programming models using actors or software transactional memory that can be simpler to reason about for some use cases than typical Java shared-state multithreaded code. For example, to maintain a variable that may be accessed and changed by several different threads, a safe Java solution requires a careful analysis and synchronization strategy to avoid deadlocks or logic errors. However, with a software transactional memory, programmers access and modify variables inside atomic transactions (similar to how databases are modified) and let the transaction manager handle the complexities of concurrent access.

Here are some stories about how developers have integrated alternative JVM languages in their Java projects.

Groovy

Groovy is a dynamic language that offers a gentle transition from Java. Most Java code is valid Groovy code, which means a Java programmer can ease his way into the more dynamic features of the language. Runtime metaprogramming, or writing code that can write code (imagine an object representing a database record that can connect to the database at runtime, inspect its target table schema, then generate methods to access each column), is one of the most powerful dynamic features of Groovy, yet it cannot be emulated in Java—all Java methods must exist at compile time. The popular web application framework Grails borrows much of the convention-over-configuration philosophy (designing libraries by preferring intuitive, common defaults versus extensive explicit configuration files) from Ruby on Rails, yet is built on established Java components like Spring and Hibernate.

Moss Collum, a developer at Cyrus Innovation, was working with his team on a five-year-old Java project. The team had started to feel frustration at how much code it took to express certain concepts in Java, especially for manipulating collections of objects (for example, finding the largest item in a set or transforming a list of objects by applying a function to each value.) They experimented by starting to write new controllers and model objects in Groovy. "Groovy's very Java-like syntax was helpful with the large existing codebase as it minimized the cost of context switching between Java and Groovy code," said Collum. "It also meant that porting Java to Groovy was dead simple; since most Java is also valid Groovy, we got into a rhythm of moving the Java code into a Groovy file and then removing the excess Java noise. Groovy's closures, easy code reflection, and concision addressed our biggest Java complaints. We've found several gotchas and a few bugs, mainly in IDE support and Java interoperability, but even so the gains are so great that we wouldn't go back to Java."

Scala

Scala offers a fusion of object-oriented and functional language programming models. This means that in Scala, programs can be composed in terms of both objects (the bread and butter of Java programs) and functions (something that can only be simulated rather verbosely in Java using anonymous inner classes). Scala is the only statically typed alternative language mentioned in this article, but it offers flexible syntax and type inference that can give Scala programs the feel of a dynamic language.

Using a new language for implementing tests is one of the most common ways to introduce a new language into a Java project. Kris Nuttycombe, cofounder of ReportGrid.com, got started with Scala at a previous job by writing tests for a Java application.

"I picked Scala because my code had been moving towards a functional style for a long time, and a stint with Ruby made me want to use lambdas (anonymous functions)," Nuttycombe said. "I started using it

The Java Virtual Machine

The Java Virtual Machine (JVM) is the runtime platform designed for applications written in the Java language. The JVM has been ported to several operating systems and hardware devices. It also includes a mature garbage collector and just-in-time compiler that boosts runtime performance. The wide deployment and runtime optimizations of the JVM make it an appealing target for alternative languages beyond Java.

Concision and Reflection

Java tends to be verbose for certain operations (see the sidebar on anonymous inner classes). Other operations like reflection, or inspecting and invoking code at runtime, also require a fair amount of code. The alternative languages listed in this article have simpler reflection features, requiring just one line of code where several would be required in Java.

Functions as First-class Objects, Closures, and Lambdas

In Java, objects can be passed as arguments to method calls, assigned to variables, and returned as a result of a method call. This is not possible with functions. That is, functions are not first-class objects in Java.

A closure is a function that may refer to the variables that were in scope when the function was created. Lambdas are functions without a name—that is, they are anonymous. These language features enable creating flexible, reusable control structures and succinct libraries.

Java's Anonymous Inner Classes, Collections, and Concision

Java does not provide first-class functions or lambdas. Instead, they must be simulated using anonymous inner classes, which can be verbose.

Imagine you had a list of numbers and wanted the list of only those numbers that were less than five. This is how you might write the code to filter that list in Java:

```
Collection<Integer> filtered =
  Collections2.filter(numbers,
    new Predicate<Integer>() {
      public boolean apply(Integer
input) {
        return input < 5;
    }
  });
```

Note that the Predicate class is being used to simulate a function that takes one argument and returns true or false. The Collections2 class is a utility class and is required since the original Java collections libraries did not include support for operations involving predicates. And here's how the same code would look in Scala, which includes lambdas and a library of lists and collections that use lambdas:

```
numbers.filter { x => x < 5 }
```

This defines an anonymous function that takes a parameter x and returns the result of the expression x < 5. Note also that the Scala version does not need to specify the types of the function arguments; the Scala compiler can infer them using a process known as type inference. Scala, like Java, is a statically typed

(*CONTINUED ON PAGE 22*)

just for testing, but I was writing better code for the tests than in the project, so 'adding Scala' eventually transitioned to 'replacing Java code with Scala.' We started running Scala in production a month after starting to use it for testing, but migrating the full codebase was a longer process; we would basically port code from Java to Scala as the opportunity arose in the refactoring process. That codebase is still in production use, continues to be developed, and handles tens of millions of dollars in transactions a year."

JRuby

JRuby is an implementation of the Ruby language on the JVM. Ruby's popularity exploded with the Ruby on Rails web framework, known for its high productivity. JRuby can run Ruby on Rails applications in an existing Java infrastructure.

Najati Imam was working for Cyrus Innovation at a large financial services client with an investment in Java and Unix infrastructure. When the team needed to add a feature to a legacy Perl CGI script, they realized they could rewrite the application faster and make it easier to change in Ruby on Rails than they could by modifying the Perl code. But getting a Ruby production environment approved would have taken several months.

"We picked JRuby so that we could run Ruby on Rails in a Java application server," said Imam. "We demonstrated that JRuby was a well-trusted, open source Java library like so many others the client was already using."

Their investment paid off; the team was able to deliver new features on time and at a pace that far exceeded the client's expectations.

"JRuby ended up being a Trojan horse that allowed us to sneak adaptive, rapid feature development in to the enterprise."

Sometimes, combining an alternative language with Java can yield something unique, pulling together the best parts of each. Bob McWhirter of JBoss is the founder of the TorqueBox project, which lets developers write JRuby applications using frameworks like Rails, Sinatra, and Rack that are served on the JBoss Application Server. This means developers get the flexibility and conciseness of Ruby with the high-performance messaging, asynchronous task processing, and scheduling facilities of a robust Java application server.

"I fell in love with Ruby," said McWhirter, "and thus needed some way to tie that back in to the Java-centric company for which I worked. The JRuby interpreter had really been doing well, and I figured JBoss had a rather large bag of enterprise-grade technology. Mix the two, and you have TorqueBox."

Clojure

Clojure is a dialect of Lisp, and unlike the other languages listed here, is not object-oriented. It features a radically different approach to managing program state; everything is immutable (unmodifiable after initialization) by default, and any changes must be coordinated through Clojure's software transactional memory or other concurrency libraries.

Dan Chamberlain of Chamberlain Consulting was working with a team building a new feature that required implementing computationally expensive, recursive search algorithms. Due to the size of the data set and the performance requirements, the searches needed to be run in parallel. When the team realized just how complex and how much code would be needed to implement these searches in parallel in Java, they started looking for implementation alternatives. Chamberlain's team chose to build this feature in Clojure.

"It really came down to immutability, recursion, and parallelism," said Chamberlain. "You can do these things in Java, but they're dead simple in Clojure."

"Things didn't always go so smoothly," continued Chamberlain. "The learning curve was pretty steep since we didn't have a background in a Lisp programming language. There have also been some issues with some of the tools and features of the language that have made us change course a couple of times. But, overall, it has been a pleasant experience."

Growing Beyond Java

The JVM has become a great platform for trying out new languages. It offers a diverse collection of alternatives, each providing a unique opportunity to learn something new. And by using languages that run on the JVM, you get the benefits of experimentation in addition to being able to use these languages in your existing Java projects.

Perhaps one of the greatest benefits of learning a new language is in self-improvement. Andrew Hunt and David Thomas write in *The Pragmatic Programmer*, "Learn at least one new language every year. Different languages solve the same problems in different ways. By learning several different approaches, you can help broaden your thinking and avoid getting stuck in a rut." **{end}**

dan@danielwellman.com

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- Supplemental Materials

(*CONTINUED FROM PAGE 21*)

language, which means that argument types are checked at compile time.

Here is a similar implementation in Ruby, a dynamic language:

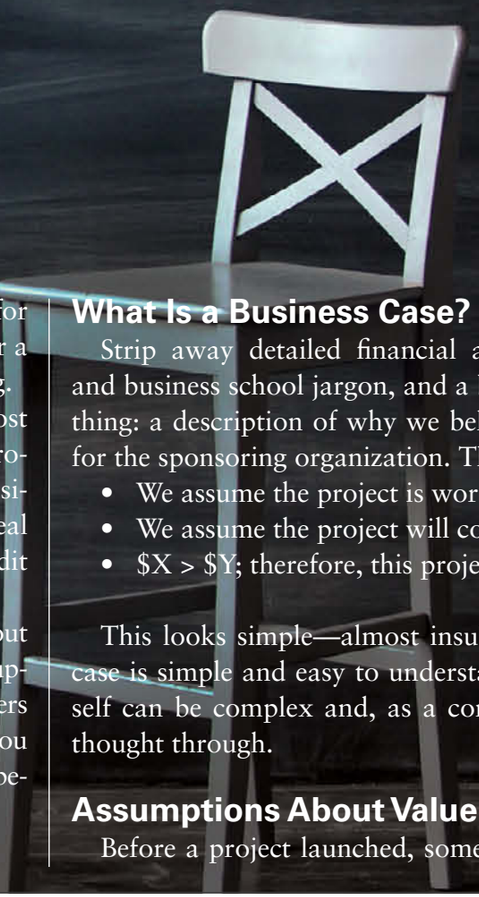
```
numbers.select { |x| x < 5 }
```

Note that in the Ruby version, the types are also omitted. This is because Ruby is a dynamically typed language, where the parameter and variable types are not declared in the source code. Groovy and Clojure are also dynamically typed languages and share the same concision.

Software Project Managers: Know Your Business Case

by Payson Hall

ISTOCKPHOTO



Software project management is typically a second or third career for people with a history of getting things done on software projects as analysts, engineers, programmers, or testers. Armed with real-world experience building products, they find themselves promoted out of a hands-on role and thrust into a position of leadership, planning, and coordination of the people and organizations building products—often with little formal training for the role. Taking a project management class or reading a book can convey the basics of the discipline—processes for defining, planning, and executing projects—but the subtleties take longer to learn and master. The same can be said of learning Java: You can take a class or read a book, but this is only starts your journey to proficiency with the language and only begins to prepare you to learn advanced topics of programming, design, and software architecture.

When I ask project managers about the business cases for their projects, common reactions include eye rolls, groans, or a short pause followed by pretending that I didn't say anything.

"Business case" sounds too much like MBA jargon for most software people. One trait commonly shared by software professions is disdain for business majors. What were those business guys learning in business school that applies in the real world anyway—how to dress for success while creating credit default swaps to crash the world economy?

You don't need an MBA to manage a project, but it turns out that your project's business case is your friend. It can help support better business decisions and navigate the dangerous waters of organizational governance and politics. Contrary to what you might think, creation and use of a basic business case is *not* beyond your pay grade, and it serves you and your project.

What Is a Business Case?

Strip away detailed financial analysis, complex templates, and business school jargon, and a business case is a pretty basic thing: a description of why we believe a project is a good idea for the sponsoring organization. This often takes the form of:

- We assume the project is worth $X.
- We assume the project will cost $Y.
- $X > $Y; therefore, this project is a good thing.

This looks simple—almost insulting. The idea of a business case is simple and easy to understand, but the business case itself can be complex and, as a consequence, often hasn't been thought through.

Assumptions About Value

Before a project launched, someone in the nosebleed section

of the org chart said something like:

We really need the product/service that project Alpha would provide. I bet it would be worth at least $X in terms of:

- Actual revenue
- Cost reduction
- Customer satisfaction
- Market position
- Improving our capacity to do similar projects
- Public relations value
- Contribution to some larger strategic objective

The process in some organizations is pretty informal, but we have to hope that someone, somewhere, is thinking about these things before we get too invested in the project. What should give you pause are the questions: Who made those assumptions about value? What were they based upon? What was the ultimate first guess about project value—in the form of a number or range?

Why do you care? The organizational will to complete a project derives from organizational leaders who choose to sponsor it. As long as the project is a priority for them and they remain in power, it is a priority for the organization. Information key to managing your project includes insights into who the project's sponsors are and what their assumptions were. This is important because the assumptions that may have been reasonable when the project began may turn out to be optimistic or pessimistic as the project progresses. In the ongoing process of managing a project, it is essential that the project manager knows the approximate value of the project and how that value was derived from the underlying assumptions so the assumptions can be monitored for significant changes. This also can provide a wealth of information about relative priorities for a project that can otherwise seem subtle or arbitrary.

For example, imagine a project that is undertaken in a race to be the first to get a product to market. The value of the project might be defined in terms of assumptions about capturing the hearts, minds, and initial purchase revenue of early adopters before your competitor can launch a fuller-featured product. In this scenario, a sponsor might assume that a basic product with few bells and whistles that is quick to market can take the wind from the sails of a competitor. Think of how this potential value might change if the competition releases its product before you are ready for market. It might be that your project should be killed. It might be that your project and product should be retooled to a more full-featured product that will be better able to compete with your competition, and certainly one that matches key features of your competitor. In either event, the assumptions of value upon which your project was based have changed, which should trigger an immediate re-assessment: Does your project as currently defined still have value?

Assumptions About Cost

Non-trivial software projects burn through money *fast*. Each person assigned often costs more than $500 per day if you consider salaries and benefits and the cost of managing them, giving them a place to work, providing equipment and other materials, and kicking in their allocated share of the cost of the CEO's limousine and first-class plane tickets. Some might say, "Yeah, but we are salaried so they would pay us whether we worked on the project or played World of Warcraft," but they are forgetting that most organizations have more projects in their portfolios than they have resources to get those projects done; so the fact that you are working full time on the Alpha project means that you are not available to work full time on the Beta project. The business school guys call this "opportunity costs," but you can summarize it by saying that if the Alpha project were cancelled, management probably wouldn't pay you to play WOW but would instead assign you to a different project.

Early estimates of project costs are often very rough, poorly documented, and notoriously lower than their final costs. There are no facts about the future; everything we estimate is an educated (or uneducated) guess based upon information available today and assumptions we make about the future. In some organizations, initial cost estimates are treated as constraints. "But when we sketched this out on the white board nine months ago, we said it might cost as little as a million dol-

Sponsor Questions to Help Build a Business Case

1. Why is our organization doing this project?
2. What will this project create that has value to the organization?
3. Is this project expected to improve our capacity to deliver projects in the future? If so, in what ways?
4. In what ways does the product or service this project creates support the strategic or tactical goals of the organization?
5. What are the consequences of not doing this project?
6. When would you like to see this project completed? What is driving that date?
7. What are the business implications of the project completing three months earlier? Three months later?
8. In rough financial terms, what is successful completion of this project on the target date worth to the organization?
9. How much has been budgeted for this project in terms of hard dollars? How much in terms of personnel time?
10. Are other resources (facilities, materials, services) allocated to the project?
11. What is the relative priority of this project in terms of other projects going on in our organization?
12. If we could deliver this project 10% earlier by spending 10% more resources, would you want me to explore that option with you?
13. If we could deliver this project 10% cheaper by delaying the schedule 10%, would you want me to explore that option with you?
14. To recap, we are assuming that this project can be completed successfully by [target date] using [resource allocation] and that the value of the successful project is worth the cost and risk. Is that correct?
15. If at any time I become concerned that our assumptions about the value of the project or the cost of completing it may be incorrect, when would you like me to bring this to your attention?

lars!" In reality, as a project progresses we should be getting a better idea of how much it really will cost. Rookie project managers often focus on tracking costs and wringing their hands about costs without backing up to ask, "Where did this cost estimate come from, and what assumptions was it based upon?"

Why do you care? As a project progresses and we get smarter about actual costs, a project manager should be comparing those costs to original expectations to understand the sources of error in estimates and also to support a reassessment of project value if the project costs begin to significantly exceed estimates. Assumptions underlying cost estimates should be known and monitored to provide early warning for changes—both positive and negative—in the project's cost structure.

For example, if a project is initially assumed to require ten person years' effort and early in design you have already consumed four person years, a project manager should ask whether the initial estimate remains credible and if the human resources available will be sufficient to complete the project successfully.

Is $X > $Y?

Given that the projected value and cost are suspect, you can imagine that monitoring the relationship between these two factors (called "return on investment" or ROI by the business geeks) to see if it remains relatively stable can be quite a chore. In addition, there are the factors outside of the project—other opportunities that the organization might (or must) pursue—that can derail a project that was otherwise on track with its cost and value assumptions.

For example, imagine your project, "Alpha," requires $1M over the next year to realize $2M in value in the two following years. Each dollar invested returns two dollars. Things are progressing well when a new opportunity, "Beta," is presented to management. Beta requires $0.5M investment to generate $2.5M in value over the next year, but those resources are not currently available (they currently are assigned to Alpha). Be prepared for your Alpha project to be slowed or killed to free up resources for the more attractive Beta. From an organizational perspective, that is good management.

Write It Down and Keep It Close

Some organizations require a preliminary business case before they will fund or authorize a project. What surprises me is how many organizations do not. How are they prioritizing and choosing among their options? How are they monitoring the quality and risk of their investments? Didn't these guys go to business school? If your organization requires a business case, make sure you know it intimately and keep it handy.

If your organization doesn't currently require a business case, speak with the project's sponsors and advocates, and try to build a crude business case for the project yourself (See the side bar for a list of helpful questions). The process of asking questions and writing down the answers will often reveal surprising things about the sponsors, project, and priorities. Capturing the assumption's underlying assertions about value and cost will give you a competitive advantage when you are negotiating for priority and enable you to better support effective decision making in your organization—and that is how engineers and technicians break into the ranks of senior management. **{end}**

payson@catalysisgroup.com

CONFERENCE SCHEDULE

Build your own conference—tutorials, training classes, keynotes, concurrent sessions, the Leadership Summit, and more—packed with information covering the latest technologies, trends, and practices in software testing.

SUNDAY

Agile Testing Practices (2 days)
Requirements-Based Testing Workshop (3 days)
Software Tester Certification—Foundation Level Training (3 days)
Using *Visual Studio® 2010 Ultimate* to Improve Software Quality (3 days)
Bonus Session—Making Agile Work: An Introduction to Agile Development Practices (1 day)

MONDAY – TUESDAY

31 In-depth half- and full-day tutorials

Fundamentals of Agile (2 days) and multi-day Training Classes and Bonus Sessions continue

WEDNESDAY – THURSDAY

5 Keynotes
42 Concurrent Sessions
The EXPO
Special Events
Bonus Sessions
TestLab
...and More!

VISIT THE EXPO!

Visit Top Industry Providers Offering the Latest in Testing Solutions

TOOLS ★ SERVICES ★ TECHNIQUES ★ DEMOS

WHO SHOULD ATTEND?

Software and test managers, QA managers and analysts, test practitioners and engineers, IT directors, CTOs, development managers, developers, and all managers and professionals who are interested in people, processes, and technologies to test and evaluate software intensive systems

KEYNOTES BY INTERNATIONAL EXPERTS

All That Testing Is Getting in the Way of Quality
James Whittaker, Google

Testing Lessons from Comic Book Superheroes
Rob Sabourin, AmiBug.com

I Didn't Know I Knew That: A Story of Self-Learning
David Hayman, Qual IT Software Testing Solutions

Test Automation Magic: Pushing the Frontiers
Krishna Iyer and Mukesh Mulchandani, ZenTEST Labs

Lightning Strikes the Keynotes
Facilitated by Lee Copeland, Software Quality Engineering

"I LOVED THE KEYNOTE SPEAKERS—FEATURED GREAT KNOWLEDGE SHARING AND A GREAT RESOURCE FOR PROBLEM SOLVING."
— Maryanne Sutton, Certification Specialist for Wegmans

FRIDAY

Testing & Quality Leadership Summit

Attend the Testing & Quality Leadership Summit Thursday evening and Friday. Join senior leaders from the industry to gain new perspectives and share ideas on today's software testing issues.

Balancing Business Agility and Software Quality
Rod Jardine, OptumHealth Vice President IT Architecture, UnitedHealth Group

Maximizing the Value of Testing
Dan Taylor, Director of Professional Services, GridPoint, Inc.

Testing in the Face of Relentless Change and Time Pressure
Joel Hynoski, Engineering Manager in Test, Google, Inc.

Think Tank Discussion
Facilitated by Jeff Payne, Chief Executive Officer, Coveros, Inc.

www.sqe.com/starwest GROUPS OF 3 OR MORE SAVE BIG!

SALARY SURVEY

Better Software magazine
TechWell.com

2011

CALLING ALL SOFTWARE DEVELOPERS, TESTERS, AND MANAGERS

The 2011 *Better Software* magazine and TechWell.com Salary Survey is in full swing!

Don't miss your chance to contribute to the collective knowledge of industry salaries and employment trends.

Follow the link that best describes your employment level to answer a few questions.

The results will appear in the November/December 2011 issue of *Better Software* magazine.

STAFF LEVEL:
www.stickyminds.com/2011staff

MANAGEMENT LEVEL:
www.stickyminds.com/2011management

DIRECTOR LEVEL:
www.stickyminds.com/2011director

BETTER SOFTWARE™
A TECHWELL™ PUBLICATION

TECHWELL™

Microsoft® Visual Studio® Test Professional 2010

Accelerate Your Deployment of Microsoft Visual Studio Test Professional 2010

If you are a Microsoft Software Assurance customer, you are entitled to Planning Services days, which can be redeemed for deployment planning engagements with qualified partners or Microsoft services. It's simple—redeem your SA Benefit points for a voucher, contract a qualified partner, they deliver the engagement and then the partner redeems that voucher with Microsoft.

So if you are ready to adopt a modern quality assurance and testing platform, then this new offering provides a plan to simplify and streamline your test planning and manual test execution processes by integrating Team Foundation Server 2010 with Test Professional 2010.

www.microsoft.com/click/services/Redirect2.ashx?CR_CC=200055076

Productivity Tools P ortfolio

WALTHAM, MA—Telerik has announced the release of its developer productivity tools portfolio. Among the highlights of the release are the addition of the "Metro" theme to Telerik's RadControls for Silverlight and WPF, and a new profiler for OpenAccess ORM.

Features include:

- HTML5-driven charting featuring a lightweight JavaScript footprint in Telerik Extensions for ASP.NET MVC, paving the way for rapid HTML5 adoption
- RadListView, a databound control supporting different views, and RadPropertyGrid, offering sophisticated theming and customizations in RadControls for WinForms
- Super-fast RadChart, DataBoundListBox, and Calendar controls in RadControls for Windows Phone 7
- Powerful Image Editor and Web Notification controls, myriad new RibbonBar features, and four new skins in RadControls for ASP.NET AJAX
- Multiple new controls and the addition of RadTreeMap to the rich data visualizations in RadControls for Silverlight and WPF

www.telerik.com

Simulink Design Verifier 2.0

NATICK, MA—MathWorks has announced that Simulink Design Verifier now includes Polyspace analysis technology for automated error detection in Simulink models. Simulink Design Verifier 2.0 integrates Polyspace error detection with existing property proving and test generation capabilities to help reduce the time required to find and fix the root cause of design errors, decreasing the overall cost of verification and validation.

Engineers across the aerospace, automotive, medical, and industrial automation and machinery industries can now apply model-based design with formal analysis methods provided by Simulink Design Verifier 2.0 to identify design errors in Simulink and Stateflow models without extensive testing or simulation.

Key product features include:

- Detection of dead logic, integer and fixed-point overflows, division by zero, and assertion violation
- Blocks and functions for modeling functional and safety requirements
- Test vector generation from functional requirements and model coverage objectives
- Property proving, with generation of violation examples for analysis and debugging
- Fixed-point and floating-point model support

mathworks.com/products/sldesignverifier

Virtualize Platform

MONROVIA, CA—Parasoft has unveiled the new release of Parasoft Virtualize, which allows organizations to automatically capture and emulate dependent system behavior. Instead of wrestling with schedules to access banking mainframes, multi-national ERP systems, third party information systems, or any system that is out of your general control, Virtualize creates a Parasoft Virtualized Asset that emulates dependent system behavior—including control over performance parameters and data.

New Product Highlights:

Testing Constraints—Parasoft Virtualize provides concurrent, 24/7 access to dependent test resources, eliminating the constraints that commonly stifle testing, such as multiple development and test teams having limited access to mainframe cycles, having shared and tightly scheduled access time, and so forth.

Performance Scaling—Parasoft Virtualize scales virtualized assets to support large-scale, high-throughput load and performance tests.

Environment Management—Parasoft Virtualize easily manages interdependent system connections, facilitating rapid development, testing, and deployment as well as reducing the overhead for configuration or infrastructure management.

Parallel Development—Parasoft Virtualize removes parallel development delays by giving teams the ability to rapidly emulate or model the needed behavior rather than having to wait for others to upgrade, configure, and manage the dependent systems.

www.parasoft.com/virtualize

CloudTest Mobile

SANTA CLARA, CA—SOASTA has partnered with DeviceAnywhere to launch CloudTest Mobile, the industry's first complete solution for mobile application performance testing. CloudTest Mobile with DeviceAnywhere allows customers to conduct rigorous performance testing that includes the end-user's experience on real mobile devices to ensure application quality and performance, even under extreme load.

CloudTest Mobile with DeviceAnywhere combines DeviceAnywhere's capability of testing and controlling real mobile devices with SOASTA's CloudTest Platform. CloudTest Mobile with DeviceAnywhere enables customers to:

- Correlate end-user experience on real devices with performance metrics
- Watch tests in real-time on real devices while the system is under load
- Measure realistic and complete user scenarios under extreme conditions
- Test user experience on real devices from distributed locations
- View end-to-end performance metrics combined on a single timeline from a single dashboard

www.soasta.com

Freescale HC08/HCS08

WIRRAL, UK—LDRA delivers full certification capabilities for Freescale's HC08 and HCS08 applications. By exploiting the resources of the CodeWarrior IDE, LDRA enables developers to run host-style analysis and test generation on resource-constrained processors. Now, automotive, industrial, and avionics applications using the HC08/HCS08 processors can automate testing and verification processes, reducing the cost of certification.

The HC08 and HCS08 fill an increasingly important role in cost-effective, low-power applications that must meet stringent certification requirements. Using the LDRA tool suite, developers can achieve programming compliance with industry standards, such as MISRA, IEC 61508, ISO 26262, and DO-178B, to the highest levels. The LDRA Tool Qualification Support Pack documents the software development process, providing the qualified output needed for certification.

The integration of the LDRA tool suite with CodeWarrior for HC08 and HCS08 streamlines user interaction. With compiler and debugger control, developers gain easy access to data on the target that can be output to the host. In addition to providing static and dynamic analysis of the target via LDRA Testbed, test cases can be generated via LDRA's TBeXtreme test case facility. These test cases can be generated and validated on the host and then rerun for qualification on the target.

www.ldra.com

uTest Version 4.0 Testing Platform

BOSTON, MA—uTest is launching version 4.0 of its software testing platform—the company's most significant update in more than a year. In addition to an improved user interface, this version incorporates two major features designed to provide tighter integration between the testing community and our cus-

tomers' engineering teams: bug fix verification and test case support.

One of the most compelling new features is bug fix verification. Since uTest's community tests apps under real-world conditions—working on real devices, carriers, browsers, and operating systems—it can be difficult for customers to validate that a bug fix has addressed the problem. Now, with the single click of a button, companies can activate bug fix verification for any defects discovered, which automatically sends a request to uTest's community to attempt to reproduce the functional bugs on a new version of the application and report back whether the defects have been successfully eliminated.

Depending on the number of apps a company has in development at any given time, it may have hundreds or even thousands of test cases to manage. This makes tracking and consolidating the results of test case execution a highly time-consuming task. With uTest's new test case support feature, customers can see a real-time summary of their testing coverage and quickly identify where problems exist.

utest.com

Conference Schedule

Build your own conference—multi-day training classes, tutorials, keynotes, conference classes, Summit sessions, and more—packed with information covering the latest technologies, trends, and practices in agile methods and software development.

SUNDAY

Software Tester Certification-Foundation Level Training (3 days)

Certified ScrumMaster Training (2 days)

Product Owner Certification (2 days)

Agile Testing Practices (2 days)

Making Agile Work: An Introduction to Agile Development Practices (Free bonus session)

MONDAY–TUESDAY

34 In-depth half- and full-day Tutorials

Multi-day training classes continue

Business Analysis and Requirements Workshop (2 days)

Fundamentals of Agile Certification (2 days)

WEDNESDAY–THURSDAY

4 Keynotes

48 Conference Classes

Networking EXPO

Special Events

...and More!

Who Should Attend?

Software professionals seeking the latest practices in software development today—from highly structured plan-driven approaches to highly creative, customer-intimate agile ones.

- Software managers, directors, CTOs, and CiOs
- Project managers and leads
- Measurement and process improvement specialists
- Requirements and business analysts
- Software architects
- Lead developers and software engineers
- Security engineers
- Test and QA managers

The EXPO

November 9–10, 2011

Find Solutions to Your Software Development Challenges!

TOOLS • TECHNIQUES • SERVICES • DEMOS • SOLUTIONS

BETTER SOFTWARE CONFERENCE EAST

AGILE DEVELOPMENT PRACTICES CONFERENCE East

www.sqe.com/betteragileeast REGISTER EARLY AND SAVE UP TO $200!

KEYNOTES BY INTERNATIONAL EXPERTS

Mission Critical Agility
Jeff Norris, NASA

No Silver Bullet? Silver Buckshot May Work
Gregory Pope, Lawrence Livermore National Laboratory

What's Surgery Got To Do with Improving Software Practices?
Edward Kit, Software Development Technologies

Enterprise DevOps: Breaking Down the Barriers between Development and IT Operations
Jez Humble, ThoughtWorks

The Agile Mindset: Principles for Collaborating and Innovating with Agility
Adrian Cho, IBM

Agile Requirements: Not an Oxymoron
Ellen Gottesdiener, EBG Consulting

"Keynote speakers were both informative and entertaining."

—Frankg Agrusa, ACL Corporate Systems

FRIDAY

Agile Leadership Network Summit

Join your peers and industry veterans to explore one of the biggest challenges facing agile today—leadership! You'll hear what's working on agile teams—and what's not—and have the opportunity to share your experiences and successes.

Agile LEADERSHIP NETWORK SUMMIT

Beyond Scope, Schedule, and Cost: Optimizing Value
Pat Reed, Senior Director, Gap Inc.

Moving Past ROI to Real and Lasting Value
Maxwell Keeler, Vice President, The Motley Fool, Inc.

Don't Bask in Your Agile Successes: There Are Bigger Problems to Solve
Kevin Fisher, Associate Vice President of Corporate Internet & Mobile, Nationwide Insurance

Think Tank Discussion

www.sqe.com/betteragileeast REGISTER EARLY AND SAVE UP TO $200!

FAQ

expert answers to frequently asked questions

What Is the Right Ratio of Testers to Developers on an Agile Team?

by Janet Gregory
janet@agiletester.com

Many testers I know feel overwhelmed and want help. Many test managers I meet feel they are understaffed. Developers often want more testers for that extra safety net. I believe that people want justification for hiring more testers for their teams, so they ask this question in the hopes that they can go back to their organization and say that "the standard is 'x:1,' and we are severely understaffed." Unfortunately, there is no magic formula to answer this question.

However, being on an agile team allows us to be much more flexible with how we approach our testing effort. A critical success factor for testers on an agile team is the "whole team approach." When I say team, I mean the project team, which includes testers, developers, product champion, perhaps technical writers, and business analysts. When team members understand that they, as a whole, are responsible for the quality of the product they are developing, they learn to use their individual strengths for the greater good. Many testers say it sounds too good to be true.

I have been on teams that have an 8:1 ratio and teams that have a 1:1 ratio. So much depends on the application you are building and the make up of the team. For example, when I was on the team that had an 8:1 ratio, I played much more of a test consultant and business analyst role, and the developers created all the automated tests. One of the reasons I believe this worked was that the project was for an event notification system [1] and had very little business logic and a very light interface. The second reason was the developers' complete buy-in to good coding practices.

On the flip side, I have worked with teams that were 1:1 or 1:2 ratios of testers to developers. These projects usually were large systems that had interconnected applications. The testers on these teams needed good domain knowledge to help the team think about implications to the system. The teams that were the most successful used collaborative automation tools so that the acceptance tests could be automated during each iteration. The testers would design and specify the tests, and the developers would create the test fixtures and methods to run them.

One team I talked to had no testers, only developers and business analysts. They pooled their resources and put out a quality product with every release.

The key to finding the right ratio for your team is to experiment. During your iteration and project retrospectives, if you feel that your testing isn't keeping up, really look at the problem and address it. Your first instinct should not be "Let's add more testers." Quite often, the problem is totally unrelated to that and you will only mask the problem by adding another tester. Instead, look at the root cause and see if other people on your team can help. **{end}**

Please see this issue's StickyNotes for references.

Learning for Agile Testers, Part 2

In part one (March/April 2011), we discussed the general "thinking" skills. Now, let's build on those with specific technical skills for your tester's toolkit.

by Lisa Crispin and Janet Gregory | lisa.crispin@gmail.com janet@agiletester.com

Automation Skills

For automation to succeed, we need to apply good design practices. For example, we strive to keep each automated test to a single, clear purpose and extract duplication into macros and modules. Our goal is to make it easy to diagnose test failures and change the tests in only one place when the code changes. When testers collaborate with programmers and other team members, this is much easier to accomplish.

New-and-improved tools are available to help agile teams improve software development.

Lisa's Story: Back in 2003, my team had zero test automation and no automated build process, but we knew it had to be our number-one priority. Our system administrator researched build tools, and we decided to use CruiseControl. Meanwhile, I was evaluating GUI test tools, and we decided to use Canoo WebTest, partly because it integrated seamlessly with CruiseControl. My previous teams had continuous integration (CI), but I wanted to learn more, so I read the CI article on Martin Fowler's website [1] *and we used his guidelines to set up our own. As a team, we have worked to keep up with the latest CI practices and tools, and our build process continues to evolve and improve.*

> **"Learning takes time. If you are constantly running just to keep up, you won't have time to learn and try new ideas."**

Learn how to evaluate and choose the right tools, so you can help your team create maintainable automated regression tests. You can free up time for essential testing activities such as exploratory testing.

Acceptance Test-driven Development

Communication skills and good domain understanding enable testers to help business experts give good examples of both desired and undesired system behavior. We can turn these examples into tests that help the programmers understand what code to write. This is called acceptance test-driven development, and it is a major step toward building quality into the code and preventing defects. When we automate the acceptance and story tests during coding using a collaborative test automation tool, such as FitNesse, the tests can immediately be added to the regression suite. This helps testing "keep up" with programming.

Learning Styles

Some of us are auditory learners—we learn by listening. Some of us are visual learners—we need to see pictures. And then there are those of us who need to do—we call that *kinaesthetic learning*. Many times we need to take in information in more than one way. For example, Janet is an auditory learner, but she needs to practice skills to "set" them. When she tries to get a concept across, Janet needs to draw it out to help explain. Understand your style so you can get the best out of each learning experience.

Emotional aspects of learning have a big impact. We all have blind spots that may prevent us from learning or triggers where we shut down and don't hear the message anymore. To learn and question, we need a safe environment. Keep your emotional "hot buttons" in mind and focus on what you can learn from instructors, material, or teammates to enhance your abilities.

A good mentor can give you guidance when you don't even realize you need it. Mentors with different backgrounds or from other industries besides testing and software development might work best with your learning style. Don't limit yourself to coaches, mentors, and instructors who work specifically in software testing.

Janet's Story: When Lisa and I started developing and teaching our agile testing course, I began to read about teaching strategies. I picked up a few tips from the books and applied them. I also watched other instructors and tried to find the style that suited me. However, one of the people I turned to the most to mentor me was my sister. She is a first grade teacher with her master's degree in early childhood learning. She opened my eyes to new ways to approach teaching and working with my classes.

Learning Resources

Keep your learning style in mind as you seek out places to hone your skills. We're fortunate to have many good software and testing books and plenty of free material on the Internet. Communities of practice are another good place to find mentors and learn together with your fellow testing professionals. Take

advantage of the resources your peers can provide. "Lunch and Learns" and book groups within an organization are another effective learning mechanism.

Conferences are an obvious way to get a lot of new ideas in a very short period of time. Most importantly, you'll meet practitioners and thought leaders with whom you can form a lasting network—a constant source of inspiration and ideas. Testing conferences are an obvious choice for testers, but consider other conferences, such as those that help you work on specific skills like scripting languages.

Mailing lists and social networks such as Twitter can introduce you to articles and blog posts on topics that interest you. Testing communities, such as Weekend Testers [2], give you the opportunity to learn in real time from other testers. Podcasts and online videos simulate the experience of actually attending a conference session or course.

Local testing user groups can be a great source of free training and information. Sharing your own experience is a great way to learn. Consider presenting an experience report or facilitating a workshop for your local user group.

Your best learning opportunities might be right where you are. Do you work with some smart programmers, database or system administrators, and testers? What can they teach you? Pair with your fellow team members: There's a lot you can learn from your coworkers, and a lot you can teach them.

Lisa's Story: I was part of a team that decided we would pair on every coding and testing task. When I paired with a programmer to write tests with FitNesse and SWAT, I noticed how naturally the programmer would notice duplication in the test code and immediately extract it out into our library of macros. I sharpened my test design skills considerably as a result.

Time for Learning

Learning takes time. If you are constantly running just to keep up, you won't have time to learn and try new ideas. Work with your managers and stakeholders to budget time into your schedule to learn new skills, understand the business domain more, and try new tools. There's always pressure to meet deadlines, but taking the time to build in quality and keep technical debt to a manageable level helps the team deliver more value over the long term.

Set one learning goal for the next three months. Get out of your comfort zone. Take control of your own professional development. You'll have a more fun and rewarding career. **{end}**

Sticky Notes

For more on the following topic go to www.StickyMinds.com/bettersoftware.

- References
- Further Reading

index to advertisers

ADP East 2011	www.sqe.com/adpeast	29
Better Software Conference and Expo 2011	www.sqe.com/adpeast	29
Brandt Technologies	www.brandttechnologies.com	12
Hewlett-Packard	www.hp.com/go/alm	34
McCabe	www.mccabe.com	30
Microsoft	www.microsoft.com/visualstudio/test	1
Microsoft	www.microsoft.com/visualstudio/test	2
Microsoft	www.microsoft.com/visualstudio/test	5
Mindtree	www.mindtree.com	18
Neotys	www.neotys.com	11
Orasi	www.orasi.com	33
Perforce	www.perforce.com	22
QA InfoTech	www.qainfotech.com	29
Ranorex	www.ranorex.com	5
Seapine	www.seapine.com	3
SmarteSoft	www.smartesoft.com	18
SQE Training—Live Virtual Training	www.sqetraining.com/VirtualTraining	6
STAR*WEST* 2011	www.sqe.com/starwest	25
TechExcel	www.techexcel.com	4
Telerik	www.telerik.com/automated-testing-tools	7
Userlytics	www.userlytics.com	22
uTest	www.utest.com	8
Web Performance	www.webperformance.com	13

Display Advertising
advertisingsales@sqe.com

All Other Inquiries
info@bettersoftware.com

Better Software (USPS: 019-578, ISSN: 1553-1929) is published six times per year January/February, March/April, May/June, July/August, September/October, November/December. Subscription rate is US $40.00 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 or call 800.450.7854. Back issues may be purchased for $15 per issue (plus shipping). Volume discounts available.

Periodicals Postage paid in Orange Park, FL, and other mailing offices. POSTMASTER: Send address changes to Better Software, 340 Corporate Way, Suite 300, Orange Park, FL 32073, info@bettersoftware.com.