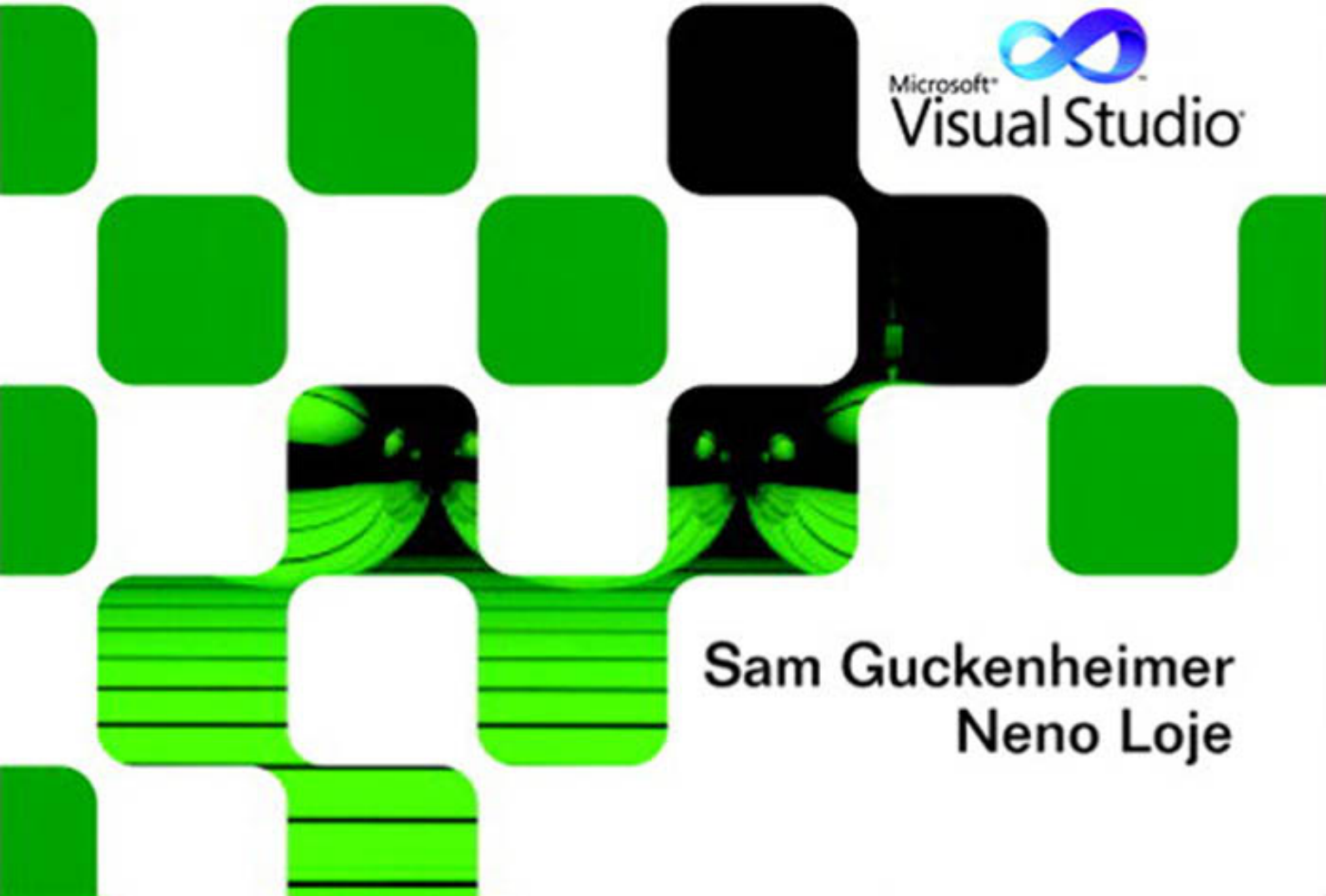
Foreword by Ken Schwaber

Agile Software Engineering with Visual Studio

From Concept to Continuous Feedback



Sam Guckenheimer
Neno Loje

Praise for
Agile Software Engineering with Visual Studio

"Agile dominates projects increasingly from IT to product and business development, and Sam Guckenheimer and Neno Loje provide pragmatic context for users seeking clarity and specifics with this book. Their knowledge of past history and current practice, combined with acuity and details about Visual Studio's agile capabilities, enable a precise path to execution. Yet their voice and advice remain non-dogmatic and wise. Their examples are clear and relevant, enabling a valuable perspective to those seeking a broad and deep historical background along with a definitive understanding of the way in which Visual Studio can incorporate agile approaches."

—**Melinda Ballou**, Program Director, Application Lifecycle Management and Executive Strategies Service, International Data Corporation (IDC)

"Sam Guckenheimer and Neno Loje have forgotten more about software development processes than most development 'gurus' ever knew, and that's a good thing! In *Agile Software Engineering with Visual Studio*, Sam and Neno distill the essence of years of hard-won experience and hundreds of pages of process theory into what really matters—the techniques that high performance software teams use to get stuff done. By combining these critical techniques with examples of how they work in Visual Studio, they created a de-facto user guide that no Visual Studio developer should be without."

—**Jeffrey Hammond**, Principal Analyst, Forrester Research

"If you employ Microsoft's Team Foundation Server and are considering Agile projects, this text will give you a sound foundation of the principles behind its agile template and the choices you will need to make. The insights from Microsoft's own experience in adopting agile help illustrate challenges with scale and the issues beyond pure functionality that a team needs to deal with. This book pulls together into one location a wide set of knowledge and practices to create a solid foundation to guide the decisions and effective transition, and will be a valuable addition to any team manager's bookshelf."

—**Thomas Murphy**, Research Director, Gartner

"This book presents software practices you should want to implement on your team and the tools available to do so. It paints a picture of how first class teams *can* work, and in my opinion, is a must read for anyone involved in software development. It will be mandatory reading for all our consultants."

—**Claude Remillard**, President, InCycle

"This book is the perfect tool for teams and organizations implementing agile practices using Microsoft's Application Lifecycle Management platform. It proves disciplined engineering and agility are not at odds; each needs the other to be truly effective."

—**David Starr**, Scrum.org

"Sam Guckenheimer and Neno Loje have written a very practical book on how Agile teams can optimize their practices with Visual Studio. It describes not only how Agile and Visual Studio work, but also the motivation and context for many of the functions provided in the platform. If you are using Agile and Visual Studio, this book should be a required read for everyone on the team. If you are not using Agile or Visual Studio, then reading this book will describe a place that perhaps you want to get to with your process and tools."

—**Dave West**, Analyst, Forrester Research

"Sam Guckenheimer and Neno Loje are leading authorities on agile methods and Visual Studio. The book you are holding in your hand is the authoritative way to bring these two technologies together. If you are a Visual Studio user doing agile, this book is a must read."

—**Dr. James A. Whittaker**, Software Engineering Director, Google

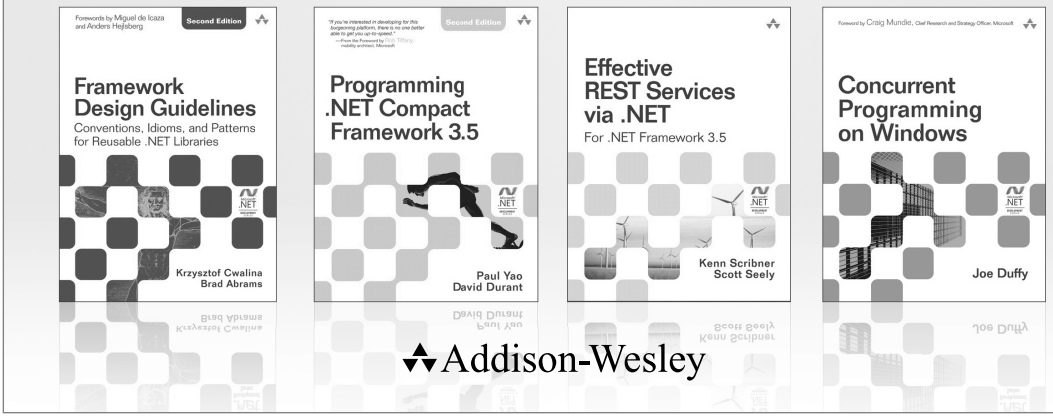
"Agile development practices are a core part of modern software development. Drawing from our own lessons in adopting agile practices at Microsoft, Sam Guckenheimer and Neno Loje not only outline the benefits, but also deliver a hands-on, practical guide to implementing those practices in teams of any size. This book will help your team get up and running in no time!"

—**Jason Zander**, Corporate Vice President, Microsoft Corporation

Agile Software Engineering with Visual Studio

From Concept to Continuous Feedback

Microsoft® .NET Development Series

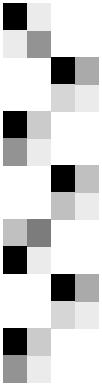


Visit **informit.com/msdotnetseries** for a complete list of available products.

The award-winning **Microsoft .NET Development Series** was established in 2002 to provide professional developers with the most comprehensive, practical coverage of the latest .NET technologies. Authors in this series include Microsoft architects, MVPs, and other experts and leaders in the field of Microsoft development technologies. Each book provides developers with the vital information and critical insight they need to write highly effective applications.

PEARSON

Addison-Wesley | Cisco Press | EXAM/CRAM | IBM Press | QUE | PRENTICE HALL | SAMS | Safari Books Online



Agile Software Engineering with Visual Studio

From Concept to Continuous Feedback

- Sam Guckenheimer
- Neno Loje

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

The Library of Congress cataloging-in-publication data is on file.

Copyright © 2012 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

The .NET logo is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries and is used under license from Microsoft.

Microsoft, Windows, Visual Studio, Team Foundation Server, Visual Basic, Visual C#, and Visual C++ are either registered trademarks or trademarks of Microsoft Corporation in the U.S.A. and/or other countries/regions.

ISBN-13: 978-0-321-68585-8
ISBN-10: 0-321-68585-7

Text printed in the United States on recycled paper at R.R. Donnelly in Crawfordsville, Indiana.

First printing September 2011

To Monica, Zoe, Grace, Eli, and Nick,
whose support made this book possible.

—Sam

Contents

Foreword

It is my honor to write a foreword for Sam's book, *Agile Software Engineering with Visual Studio*. Sam is both a practitioner of software development and a scholar. I have worked with Sam for the past three years to merge Scrum with modern engineering practices and an excellent toolset, Microsoft's VS 2010. We are both indebted to Aaron Bjork of Microsoft, who developed the Scrum template that instantiates Scrum in VS 2010 through the Scrum template.

I do not want Scrum to be prescriptive. I left many holes, such as what is the syntax and organization of the product backlog, the engineering practices that turned product backlog items into a potentially shippable increment, and the magic that would create self-organizing teams. In his book, Sam has superbly described one way of filling in these holes. He describes the techniques and tooling, as well as the rationale of the approach that he prescribes. He does this in detail, with scope and humor. Since I have worked with Microsoft since 2004 and Sam since 2009 on these practices and tooling, I am delighted. Our first launch was a course, the Professional Scrum Developer .NET course, that taught developers how to use solid increments using modern engineering practices on VS 2010 (working in self-organizing, cross-functional teams). Sam's book is the bible to this course and more, laying it all out in detail and philosophy. If you are on a Scrum team building software with .NET technologies, this is the book for you. If you are using Java, this book is compelling enough to read anyway, and may be worth switching to .NET.

When we devised and signed the Agile Manifesto in 2001, our first value was "Individuals and interactions over processes and tools." Well, we have the processes and tools nailed for the Microsoft environment. In Sam's book, we have something developers, who are also people, can use to understand the approach and value of the processes and tools. Now for the really hard work, people. After 20 years of being treated as resources, becoming accountable, creative, responsible people is hard. Our first challenge will be the people who manage the developers. They could use the metrics from the VS 2010 tooling to micromanage the processes and developers, squeezing the last bit of creativity out and leaving agility flat. Or, they could use the metrics from the tools to understand the challenges facing the developers. They could then coach and lead them to a better, more creative, and more productive place. This is the challenge of any tool. It may be excellent, but how it is used will determine its success.

Thanks for the book, Sam and Neno.

Ken Schwaber
Co-Creator of Scrum

Preface

Five years ago, we extended the world's leading product for individual developers, Microsoft Visual Studio, into Visual Studio Team System, and it quickly became the world's leading product for development teams. This addition of Application Lifecycle Management (ALM) to Visual Studio made life easier and more productive for hundreds of thousands of our users and tens of thousands of our Microsoft colleagues. In 2010, we shipped Visual Studio 2010 Premium, Ultimate, Test Professional, and Team Foundation Server. (We've dropped the Team System name.)

We've learned a lot from our customers in the past five years. Visual Studio 2010 is a huge release that enables a high-performance Agile software team to release higher-quality software more frequently. We set out to enable a broad set of scenarios for our customers. We systematically attacked major root causes of waste in the application lifecycle, elevated transparency for the broadly engaged team, and focused on flow of value for the end customer. We have eliminated unnecessary silos among roles, to focus on empowering a multidisciplinary, self-managing team. Here are some examples.

No more no repro. One of the greatest sources of waste in software development is a developer's inability to reproduce a reported defect. Traditionally, this is called a "no repro" bug. A tester or user files a bug and later receives a response to the effect of "Cannot reproduce," or "It works on my machine," or "Please provide more information," or something of the sort. Usually this is the first volley in a long game of Bug Ping-Pong, in which no software gets improved but huge frustration gets vented. Bug

Ping-Pong is especially difficult for a geographically distributed team. As detailed in Chapters 1 and 8, VS 2010 shortens or eliminates this no-win game.

No more waiting for build setup. Many development teams have mastered the practice of continuous integration to produce regular builds of their software many times a day, even for highly distributed Web-based systems. Nonetheless, testers regularly wait for days to get a new build to test, because of the complexity of getting the build deployed into a production-realistic lab. By virtualizing the test lab and automating the deployment as part of the build, VS 2010 enables testers to take fresh builds daily or intraday with no interruptions. Chapter 7, "Build and Lab," describes how to work with build and lab automation.

No more UI regressions. The most effective user interface (UI) testing is often exploratory, unscripted manual testing. However, when bugs are fixed, it is often hard to tell whether they have actually been fixed or if they simply haven't been found again. VS 2010 removes the ambiguity by capturing the action log of the tester's exploration and allowing it to be converted into an automated test. Now fixes can be retested reliably and automation can focus on the actually observed bugs, not the conjectured ones. Chapter 8, "Test," covers both exploratory and automated testing.

No more performance regressions. Most teams know the quickest way to lose a customer is with a slow application or Web site. Yet teams don't know how to quantify performance requirements and accordingly, don't test for load capacity until right before release, when it's too late to fix the bugs that are found. VS 2010 enables teams to begin load testing early. Performance does not need to be quantified in advance, because the test can answer the simple question, "What has gotten slower?" And from the end-to-end result, VS profiles the hot paths in the code and points the developer directly to the trouble spots. Chapters 6 and 8 cover profiling and load testing.

No more missed changes. Software projects have many moving parts, and the more iterative they are, the more the parts move. It's easy for developers and testers to misunderstand requirements or overlook the impact of changes. To address this, Visual Studio Test Professional introduces test

impact analysis. This capability compares the changes between any two builds and recommends which tests to run, both by looking at the work completed between the builds and by analyzing which tests cover the changed code based on prior coverage. Chapters 3 and 4 describe the product backlog and change management, and Chapters 6 through 8 show test impact analysis and the corresponding safety nets from unit testing, build automation, and acceptance testing.

No more planning black box. In the past, teams have often had to guess at their historical velocity and future capacity. VS 2010 draws these directly from the Team Foundation Server database and builds an Excel worksheet that allows the team to see how heavily loaded every individual is in the sprint. The team can then transparently shift work as needed. Examples of Agile planning are discussed in Chapters 2 and 4.

No more late surprises. Agile teams, working iteratively and incrementally, often use burndown charts to assess their progress. Not only does VS 2010 automate the burndowns, but project dashboards go beyond burndowns to provide a real-time view of quality and progress from many dimensions: requirements, tasks, tests, bugs, code churn, code coverage, build health, and impediments. Chapter 4, "Running the Sprint," introduces the "happy path" of running a project and discusses how to troubleshoot project "smells."

No more legacy fear. Very few software projects are truly "greenfield," developing brand new software on a new project. More frequently, teams extend or improve existing systems. Unfortunately, the people who worked on earlier versions are often no longer available to explain the assets they have left behind. VS 2010 makes it much easier to work with the existing code by introducing tools for architectural discovery. VS 2010 reveals the patterns in the software and enables you to automatically enforce rules that reduce or eliminate unwanted dependencies. These rules can become part of the check-in policies that ensure the team's definition of *done* to prevent inadvertent architectural drift. Architectural changes can also be tied to bugs or work, to maintain transparency. Chapter 5, "Architecture," covers the discovery of existing architecture, and Chapter 7 shows you how to automate the definition of *done*.

No more distributed development pain. Distributed development is a necessity for many reasons: geographic distribution, project complexity, release evolution. VS 2010 takes much of the pain out of distributed development processes both proactively and retrospectively. Gated check-in proactively forces a clean build with verification tests before accepting a check-in. Branch visualization retrospectively lets you see where changes have been applied. The changes are visible both as code and work item updates (for example, bug fixes) that describe the changes. You can visually spot where changes have been made and where they still need to be promoted. Chapters 6 and 7 show you how to work with source, branches, and backlogs across distributed teams.

No more technology silos. More and more software projects use multiple technologies. In the past, teams often have had to choose different tools based on their runtime targets. As a consequence, .NET and Java teams have not been able to share data across their silos. Visual Studio Team Foundation Server 2010 integrates the two by offering clients in both the Visual Studio and Eclipse integrated development environments (IDEs), for .NET and Java respectively. This changes the either-or choice into a both-and, so that everyone wins. Again, Chapters 6 and 7 include examples of working with your Java assets alongside .NET.

These scenarios are not an exhaustive list, but a sampling of the motivation for VS 2010. All of these illustrate our simple priorities: reduce waste, increase transparency, and accelerate the flow of value to the end customer. This book is written for software teams considering running a software project using VS 2010. This book is more about the *why* than the *how*.

This book is written for the team as a whole. It presents information in a style that will help all team members get a sense of each other's viewpoint. I've tried to keep the topics engaging to all team members. I'm fond of Einstein's dictum "As simple as possible, but no simpler," and I've tried to write that way. I hope you'll agree and recommend the book to your colleagues (and maybe your boss) when you've finished with it.

Enough About Visual Studio 2010 to Get You Started

When I write about Visual Studio (or VS) I'm referring to the full product line. As shown in Figure P.1, the VS 2010 family is made up of a server and a small selection of client-side tools, all available as VS Ultimate.

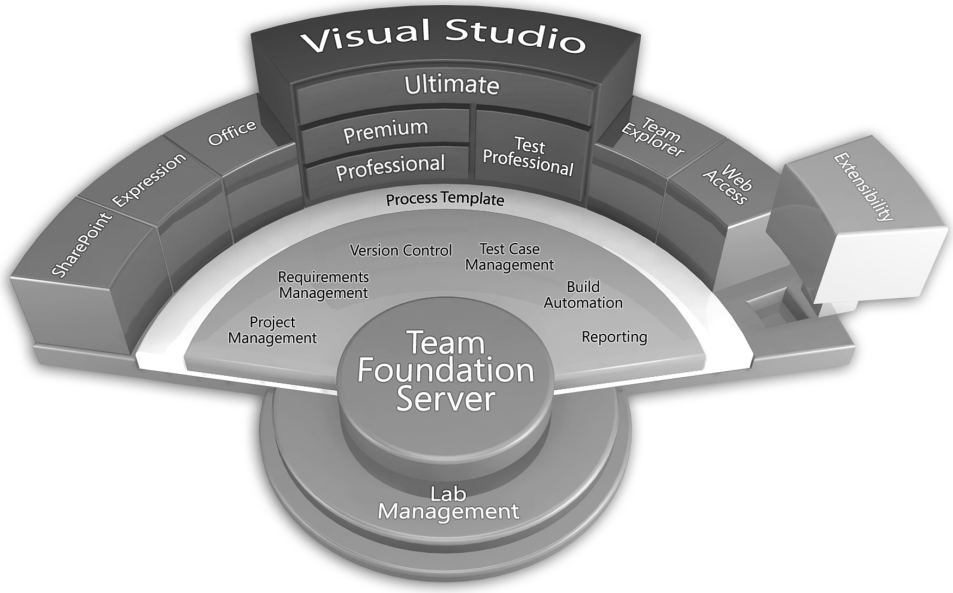


Figure P-1: Team Foundation Server, now including Lab Management, forms the server of VS 2010. The client components are available in VS Ultimate.

Team Foundation Server (TFS) is the ALM backbone, providing source control management, build automation, work item tracking, test case management, reporting, and dashboards. Part of TFS is Lab Management, which extends the build automation of TFS to integrate physical and virtual test labs into the development process.

If you just have TFS, you get a client called Team Explorer that launches either standalone or as a plug-in to the Visual Studio Professional IDE. Team Explorer Everywhere, a comparable client written in Java, launches as an Eclipse plug-in. You also get Team Web Access and plug-ins that let you connect from Microsoft Excel or Project. SharePoint hosts the dashboards.

Visual Studio Premium adds the scenarios that are described in Chapter 6, "Development," around working with the code. Visual Studio Test

Professional, although it bears the VS name, is a separate application outside the IDE, designed with the tester in mind. You can see lots of Test Professional examples in Chapter 8. VS Ultimate, which includes Test Professional, adds architectural modeling and discovery, discussed in Chapter 5.

There is also a rich community of partner products that use the extensibility to provide additional client experiences on top of TFS. Figure P.2 shows examples of third-party extensions that enable MindManager, Microsoft Word, and Microsoft Outlook as clients of TFS. You can find a directory at www.visualstudiowidgets.com/.

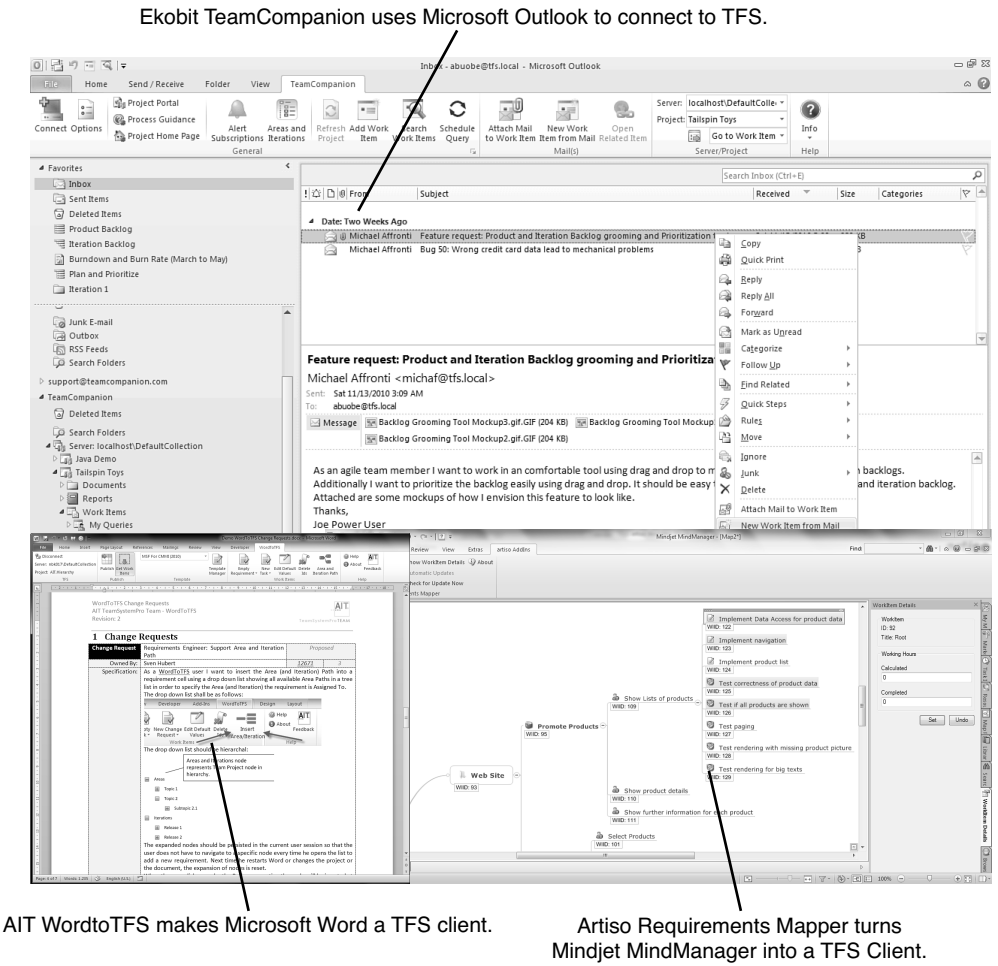


Figure P-2: A broad catalog of partner products extend TFS. Shown here are Artiso Requirements Mapper, Ekobit TeamCompanion, and AIT WordtoTFS.

Of course, all the clients read and feed data into TFS, and their trends surface on the dashboards, typically hosted on SharePoint. Using Excel Services or SQL Server Reporting Services, you can customize these dashboards. Dashboard examples are the focus of Chapter 4.

Unlike earlier versions, VS 2010 does not have role-based editions. This follows our belief in multidisciplinary, self-managing teams. We want to smooth the transitions and focus on the end-to-end flow. Of course, there's plenty more to learn about VS at the Developer Center of http://msdn.microsoft.com/vstudio/.

Acknowledgments

Hundreds of colleagues and millions of customers have contributed to shaping Visual Studio. In particular, the roughly two hundred "ALM MVPs" who relentlessly critique our ideas have enormous influence. Regarding this book, there are a number of individuals who must be singled out for the direct impact they made. Ken Schwaber convinced me that this book was necessary. The inexhaustible Brian Harry and Cameron Skinner provided detail and inspiration. Jason Zander gave me space and encouragement to write. Tyler Gibson illustrated the Scrum cycles to unify the chapters. Among our reviewers, David Starr, Claude Remillard, Aaron Bjork, David Chappell, and Adam Cogan stand out for their thorough and careful comments. And a special thanks goes to Joan Murray, our editor at Pearson, whose patience was limitless.

About the Authors

Sam Guckenheimer

When I wrote the predecessor of this book, I had been at Microsoft less than three years. I described my history like this:

I joined Microsoft in 2003 to work on Visual Studio Team System (VSTS), the new product line that was just released at the end of 2005. As the group product planner, I have played chief customer advocate, a role that I have loved. I have been in the IT industry for twenty-some years, spending most of my career as a tester, project manager, analyst, and developer.

As a tester, I've always understood the theoretical value of advanced developer practices, such as unit testing, code coverage, static analysis, and memory and performance profiling. At the same time, I never understood how anyone had the patience to learn the obscure tools that you needed to follow the right practices.

As a project manager, I was always troubled that the only decent data we could get was about bugs. Driving a project from bug data alone is like driving a car with your eyes closed and only turning the wheel when you hit something. You really want to see the right indicators that you are on course, not just feel the bumps when you stray off it. Here, too, I always understood the value of metrics, such as code coverage and project velocity, but I never understood how anyone could realistically collect all that stuff.

As an analyst, I fell in love with modeling. I think visually, and I found graphical models compelling ways to document and communicate. But the models always got out of date as soon as it came time to implement

anything. And the models just didn't handle the key concerns of developers, testers, and operations.

In all these cases, I was frustrated by how hard it was to connect the dots for the whole team. I loved the idea in Scrum (one of the Agile processes) of a "single product backlog"—one place where you could see all the work—but the tools people could actually use would fragment the work every which way. What do these requirements have to do with those tasks, and the model elements here, and the tests over there? And where's the source code in that mix?

From a historical perspective, I think IT turned the corner when it stopped trying to automate manual processes and instead asked the question, "With automation, how can we reengineer our core business processes?" That's when IT started to deliver real business value.

They say the cobbler's children go shoeless. That's true for IT, too. While we've been busy automating other business processes, we've largely neglected our own. Nearly all tools targeted for IT professionals and teams seem to still be automating the old manual processes. Those processes required high overhead before automation, and with automation, they still have high overhead. How many times have you gone to a 1-hour project meeting where the first 90 minutes were an argument about whose numbers were right?

Now, with Visual Studio, we are seriously asking, "With automation, how can we reengineer our core IT processes? How can we remove the overhead from following good process? How can we make all these different roles individually more productive while integrating them as a high-performance team?"

Obviously, that's all still true.

Neno Loje

I started my career as a software developer—first as a hobby, later as profession. At the beginning of high school, I fell in love with writing software because it enabled me to create something useful by transforming an idea into something of actual value for someone else. Later, I learned that this was generating customer value.

However, the impact and value were limited by the fact that I was just a single developer working in a small company, so I decided to focus on helping and teaching other developers. I started by delivering pure technical training, but the topics soon expanded to include process and people, because I realized that just introducing a new tool or a technology by itself does not necessarily make teams more successful.

During the past six years as an independent ALM consultant and TFS specialist, I have helped many companies set up a team environment and software development process with VS. It has been fascinating to watch how removing unnecessary, manual activities makes developers and entire projects more productive. Every team is different and has its own problems. I've been surprised to see how many ways exist (both in process and tools) to achieve the same goal: deliver customer value faster though great software.

When teams look back at how they worked before, without VS, they often ask themselves how they could have survived without the tools they use now. However, what had changed from the past were not only the tools, but also the way they work as a team.

Application Lifecycle Management and practices from the Agile Consensus help your team to focus on the important things. VS and TFS are a pragmatic approach to implement ALM (even for small, nondistributed teams). If you're still not convinced, I urge you to try it out and judge for yourself.

1
The Agile Consensus

A crisis is a terrible thing to waste.

—Paul Romer (attributed)

Wars and recessions become focal points for economic and engineering trends that have developed gradually for many years before. The Great Recession of 2007 through 2010 is a case in point. In 2008, for example, Toyota—youngest of the world's major automobile manufacturers—became the world market leader, as it predicted it would six years earlier.[1] Then in 2009, two of the three American manufacturers went through bankruptcy, while the third narrowly escaped. The emergence from this crisis underscored how much the Detroit manufacturers had failed to adapt to competitive practices that had been visible and documented for decades. In 1990, Jim Womack and colleagues had coined the term *Lean* in their exquisitely researched book *The Machine That Changed the World* to describe a new way of working that Toyota had invented.[2] By 2010, Lean had become a requirement of doing business. As the *New York Times* headline read, "G.M. and Ford Channel Toyota to Beat Toyota."[3]

The Origins of Agile

Software companies, of course, experienced their own spate of bankruptcies in the years 2000–02 and 2008–10, while internal IT organizations were

newly challenged to justify their business value. In this period, many industry leaders asked how Lean could have a similarly major impact on software engineering.

Lean was one of several approaches that became known as "Agile processes." On a weekend in 2001, 17 software luminaries convened to discuss "lightweight methods," alternatives to the more heavyweight development processes in common use. At the end of the weekend, they launched the Agile Alliance, initially charged around the *Agile Manifesto*.[4] At the end of the decade, in a 2010 study of 4,770 developers in 91 countries, 90% of respondents worked in organizations that used Agile development practices to some degree (up from 84% the previous year).[5] Contrary to the early days of Agile, the most frequent champions for introducing Agile practices are now in management roles. By now, "agility" is mainstream. In the words of Forrester Research:

> Agile adoption is a reality. Organizations across all industries are increasingly adopting Agile principles, and software engineers and other project team members are picking up Agile techniques.[6]

It seems that every industry analyst advocates Agile, every business executive espouses it, and everyone tries to get more of it.

Agile Emerged to Handle Complexity

In prior decades, managers and engineers alike assumed that software engineering was much like engineering a bridge or designing a house. When you build a bridge, road, or house, for example, you can safely study hundreds of very similar examples. The starting conditions, requirements, technology, and desired outcome are all well understood. Indeed, most of the time, construction economics dictate that you build the current house or bridge according to a proven plan very much like a previous one. In this case, the requirements are known, the technology is known, and the risk is low.

These circumstances lend themselves to a *defined process model,* where you lay out the steps well in advance according to a previously exercised baseline, derived from the process you followed in building the previous

similar examples. Most process models taught in business and engineering schools, such as the Project Management Body of Knowledge (PMBOK),[7] are defined process models that assume you can know the tasks needed to projected completion.

Software is rarely like that. With software, if someone has built a system just like you need, or close to what you need, chances are you can license it commercially (or even find it as freeware). No sane business is going to spend money building software that it can buy more economically. With thousands of software products available for commercial license, it is almost always cheaper to buy, if what you need already exists.

Accordingly, the software projects that are worth funding are the ones that haven't been done before. This has a significant implication for the process to follow. Ken Schwaber, inventor of Scrum, has adapted a graph from the book *Strategic Management and Organisational Dynamics*, by Ralph D. Stacey, to explain the management context. Stacey divided management situations into the four categories of simple, complicated, complex, and anarchic (as shown in Figure 1-1).[8]

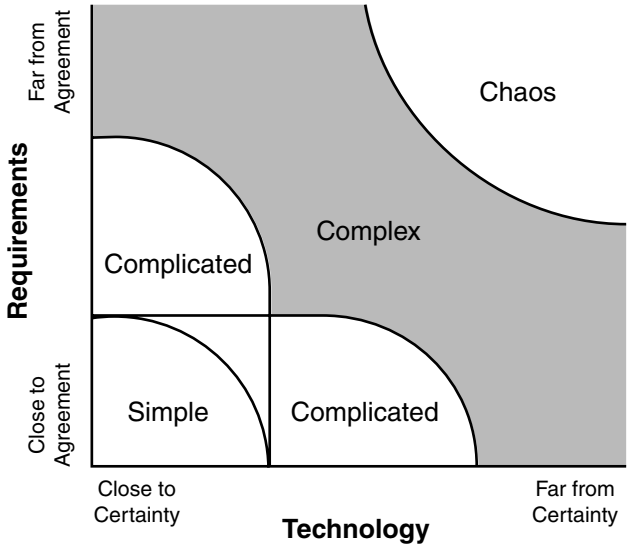


Figure 1-1: The Stacey Matrix distinguishes simple, complicated, complex, and anarchic management contexts and has been an inspiration for Scrum and other Agile practices.

Empirical Process Models

When requirements are agreed and technology is well understood, as in the house or bridge, the project falls in the simple or complicated regions. Theoretically, these simple and complicated regions would also include software projects that are easy and low risk, but as I discussed earlier, because they've been done before, those don't get funded.

When the requirements are not necessarily well agreed or the technology is not well known (at least to the current team), the project falls in the complex region. That is exactly where many software projects do get funded, because that is where the greatest opportunity for competitive business differentiation lies.

The uncertainties put these projects in Stacey's complex category, often referred to as the "edge of chaos." The uncertainties also make the defined process model quite ill suited to these projects. In these cases, rather than laying out elaborate plans that you know will change, it is often better that you create more fluid options, try a little, inspect the results, and adapt the next steps based on the experience. Indeed, this is exactly what's known as the *empirical process model,* based on what works well in product development and industries with continuous process control.[9]

An everyday example of an empirical process control is the thermostat. We don't look up hourly weather forecasts and set our heaters and air conditioners based on Gantt charts of expected temperatures. Rather, we rely on a simple feedback mechanism to adjust the temperature a little bit at a time when it is too hot or too cold. A sophisticated system might take into account the latency of response—for example, to cool down an auditorium in anticipation of a crowd or heat a stone in anticipation of a cold spell—but then the adjustment is made based on actual temperature. It's a simple control system based on "inspect and adapt."

A New Consensus

As software economics have favored complex projects, there has been a growing movement to apply the empirical models to software process. Since 1992, Agile, Lean, Scrum,[10] Kanban,[11] Theory of Constraints,[12] System

Thinking,[13] XP,[14] and Flow-Based Product Development[15] have all been part of the trend. All of these overlap and are converging into a new paradigm of software engineering. No single term has captured the emerging paradigm, but for simplicity, I'll call this the *Agile Consensus*.

The Agile Consensus stresses three fundamental principles that reinforce each other:

1. Flow of value, where *value* is defined by the customer who is paying for or using this project
2. Continual reduction of waste impeding the flow
3. Transparency, enabling team members to continually improve the above two

These three principles reinforce each other (as shown in Figure 1-2). Flow of value enables transparency, in that you can measure what is important to the customer (namely, potentially shippable software). Transparency enables discovery of waste. Reducing waste, in turn, accelerates flow and enables greater transparency. These three aspects work together like three legs of a stool.

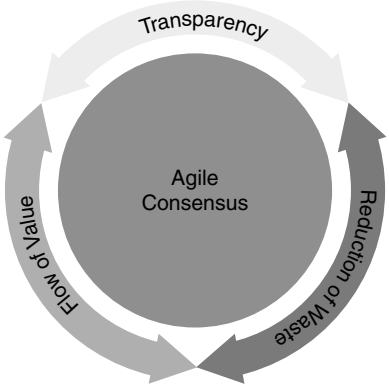


Figure 1-2: Flow of value, transparency, and reduction of waste form the basis of the Agile Consensus.

Microsoft's Visual Studio Team System 2005 and its successor Visual Studio Team System 2008 were among the first commercial products to support software teams applying these practices. Visual Studio 2010 (VS 2010;

Microsoft has dropped the words *Team System* from the name) has made another great leap forward to create transparency, improve flow, and reduce waste in software development. VS 2010 is also one of the first products to tackle end-to-end Agile engineering and project management practices. A key set of these practices come from Scrum.

Scrum

As Forrester Research found recently, "When it comes to selecting an Agile methodology, Scrum is the overwhelming favorite."[16] Scrum leads over the nearest contender by a factor of three. Scrum has won acceptance because it simplifies putting the principles of flow of value, reduction of waste, and transparency into practice.

Scrum identifies three interlocking cadences: release or product planning, sprint (usually 2–4 weeks), and day; and for each cadence, it prescribes specific meetings and maximum lengths for the meetings to keep the overhead under 10% of the total time of the cycle. To ensure flow, every Sprint produces a potentially shippable increment of software that delivers a subset of the *product backlog* in a working form. Figure 1-3 shows the cycles.[17]

Core to Scrum is the concept of self-managing teams. Rather than rely on a conventional hierarchical structure with a conventional project manager, a self-managing team uses transparently available metrics to control its own work in process and improve its own velocity of flow. Team members are encouraged to make improvements whenever necessary to reduce waste. The sprint cadence formally ensures that a "retrospective" is used at least monthly to identify and prioritize actionable process improvements. Scrum characterizes this cycle as "inspect and adapt." Although more nuanced than a thermostat, the idea is similar. Observation of the actual process and its results drives the incremental changes to the process.

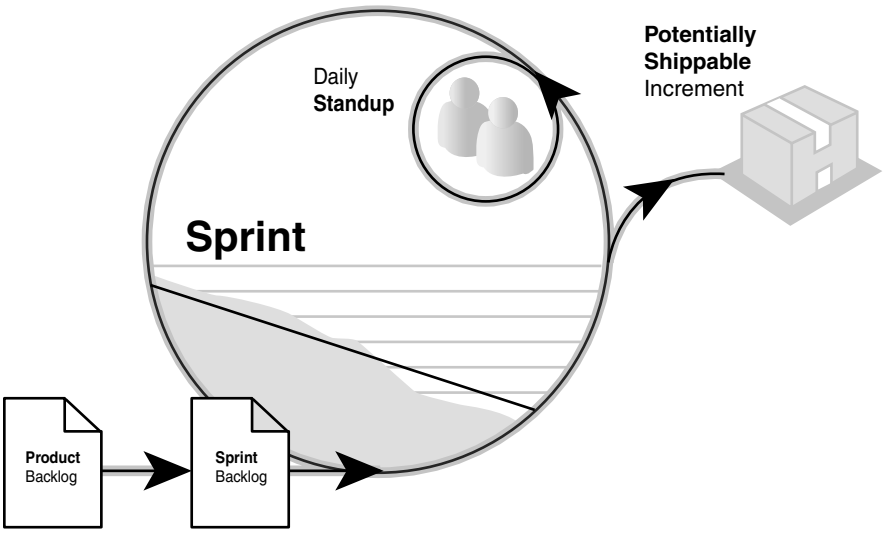


FIGURE 1-3: The central image of the Scrum methodology is a great illustration of flow in the management sense.

Potentially Shippable

Scrum also enables transparency by prescribing the delivery of "potentially shippable increments" of working software at the end of every sprint. For example, a team working on a consumer Web site might focus one sprint on catalog search. Without a working checkout process, the site would be incomplete and not actually shippable or publicly deployable. However, if the catalog search were usable and exercised the product database, business logic, and display pages, it would be a reasonable *potentially shippable* increment. Both stakeholders and the team can assess the results of the sprint, provide feedback, and recommend changes before the next sprint. Based on these changes, the product owner can adjust the product backlog, and the team can adjust its internal processes.

Increasing the Flow of Value in Software

Central to Agile Consensus is an emphasis on *flow.* The flow of customer value is the primary measure of the system of delivery. David J. Anderson summarizes this view in *Agile Management for Software Engineering:*

> Flow means that there is a steady movement of value through the system. Client-valued functionality is moving regularly through the stages of transformation—and the steady arrival of throughput—with working code being delivered.[18]

In this paradigm, you do not measure planned tasks completed as the primary indicator of progress; you count units of value delivered.

Scrum introduced the concept of the *product backlog*, "a prioritized list of everything that might be needed in the product."[19] This is a stack-ranked list of requirements maintained by the product owner on the basis of stakeholder needs. The product backlog contains the definition of the intended customer value. The product backlog is described in depth in Chapter 3, "Product Ownership."

The product backlog provides the yardstick against which flow of value can be measured. Consistent with Scrum, Visual Studio 2010 offers an always-visible product backlog to increase the communication about the flow of customer-valued deliverables. The product backlog is the current agreement between stakeholders and the development team regarding the next increments to build, and it is kept in terms understandable to the stakeholders. Usually, product backlog items are written as *user stories*, discussed more in Chapter 3. The report in Figure 1-4 shows product backlog and the test status against the product backlog. This bird's eye view of progress in the sprint lets the team see where backlog items are flowing and where they are blocked. More detailed examples of a common dashboard, showing both progress and impediments, are discussed in Chapter 4, "Running the Sprint."

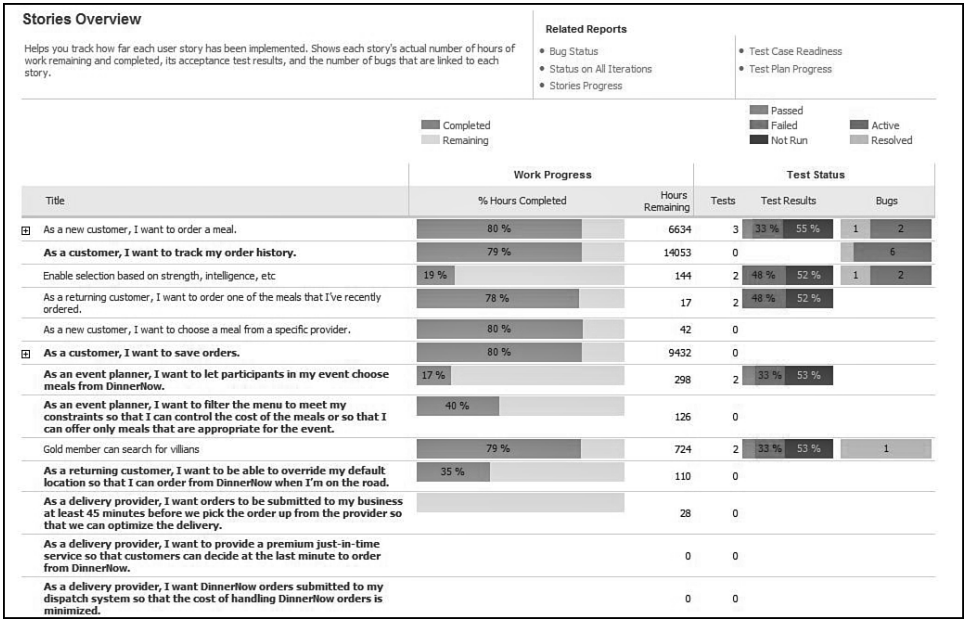


FIGURE 1-4: The Stories Overview report shows each product backlog Item on a row, with a task perspective under Work Progress, a Test Results perspective reflecting the tests run, and a Bugs perspective for the bugs actually found.

Reducing Waste in Software

The enemy of flow is waste. This opposition is so strong that reduction of waste is the most widely recognized aspect of Lean. Taiichi Ohno of Toyota, the father of Lean, developed the taxonomy of *muda* (Japanese for "waste"), *mura* ("inconsistency"), and *muri* ("unreasonableness"), such that these became common business terms.[20] Ohno categorized seven types of *muda* with an approach for reducing every one. Mary and Tom Poppendieck introduced the *muda* taxonomy to software in their first book.[21] Table 1-1 shows an updated version of this taxonomy, which provides a valuable perspective for thinking about impediments in the software development process, too.

TABLE 1-1: Taiichi Ohno's Taxonomy of Waste, Updated to Software Practices

Muda 無駄	**In-Process Inventory**	Partially implemented user stories, bug debt and incomplete work carried forward. Requires multiple handling, creates overhead and stress.
	Overproduction	Teams create low-priority features and make them self-justifying. This work squeezes capacity from the high-priority work.
	Extra Processing	Bug debt, reactivations, triage, redundant testing, relearning of others' code, handling broken dependencies.
	Transportation	Handoffs across roles, teams, divisions, and so on.
	Motion	Managing tools, access rights, data transfer, lab setup, parallel release work.
	Waiting	Delays, blocking bugs, incomplete incoming components or dependencies.
	Correction	Scrap and rework of code.
Mura 斑	**Unevenness**	Varying granularity of work, creating unpredictability in the flow.
	Inconsistency	Different definitions of done, process variations that make assessment of "potentially shippable" impossible.
Muri 無理	**Absurdity**	Stress due to excessive scope.
	Unreasonableness	Expectations of heroic actions and commitments to perform heroic actions.
	Overburden	Stress due to excessive overhead.

Consistent with Ohno's taxonomy, *in-process inventory*, *transportation*, *motion*, and *waiting* often get overlooked in software development. Especially when many specialist roles are involved, waste appears in many subtle ways. As Kent Beck observed, "The greater the flow, the greater the need to support transitions between activities."[22] Some of the transitions take seconds or minutes, such as the time a developer spends in the cycle of coding and unit testing. Other transitions too often take days, weeks, or unfortunately, months. All the little delays add up.

Transparency

Scrum and all Agile processes emphasize self-managing teams. Successful self-management requires transparency. Transparency, in turn, requires measurement with minimal overhead. Burndown charts of work remaining in tasks became an early icon for transparency. VS takes this idea further, to provide dashboards that measure not just the tasks, but multidimensional indicators of quality.

VS enables and instruments the process, tying source code, testing, work items, and metrics together. Work items include all the work that needs to be tracked on a project, such as scenarios, development tasks, test tasks, bugs, and impediments. These can be viewed and edited in the Team Explorer, Team Web Access, Visual Studio, Microsoft Excel, or Microsoft Project.

Technical Debt

In 2008, the plight of the financial sector plunged the world economy into the steepest recession of the past 70 years. Economists broadly agree that the problem was a shadow banking system with undisclosed and unmeasured financial debts hidden by murky derivatives. Fortunately, this crisis has led legislators to remember the words of U.S. Supreme Court Justice Louis Brandeis, "Sunlight is said to be the best of disinfectants; electric light the most efficient policeman."[23]

For software teams, the equivalent of these unknown liabilities is *technical debt.* Technical debt refers to work that needs to be done to achieve the *potentially shippable* threshold, such as fixing bugs, unit testing, integration testing, performance improvement, security hardening, or refactoring for sustainability. Technical debt is an unfortunately common form of waste. Unanticipated technical debt can crush a software project, leading to unpredictable delays, costs, and late cancellation. And similar to the contingent financial liabilities, technical debt is often not disclosed or measured until it is too late.

Among the problems with technical debt is the fact that it prevents the stakeholders from seeing what software is actually in a potentially shippable state. This obstacle is the reason that Scrum prescribes that every product backlog item must be delivered according to a definition of *done*

agreed by the team. This is discussed more in Chapter 2, "Scrum, Agile Practices, and Visual Studio." Think of the transparency like Louis Brandeis's electric light: It makes the policeman less necessary. Together, the common definition of *done* and transparent view of progress prevent the accumulation of technical debt, and thereby enable the team and its stakeholders to assess the team's true velocity.

An Example

Consider the effort spent in making a new build available for testing. Or think about the handling cost of a bug that is reported fixed and then has to get reactivated. Or consider writing specs for requirements that ultimately get cut. All of these wastes are common to software projects.

VS 2010 has focused on reducing the key sources of waste in the software development process. The build automation in VS Team Foundation Server allows continuous or regularly scheduled builds, and with "gated check-in" can force builds before accepting changed code. Lab Management can automatically deploy those builds directly into virtualized test environments. These are discussed in Chapter 7, "Build and Lab."

An egregious example of waste is "Bug Ping-Pong." Every tester or product owner has countless stories of filing bugs with meticulous descriptions, only to receive a "Cannot reproduce" response from a programmer. There are many variants of this "No repro" response, such as "Need more information" or "Works on my machine." This usually leads to a repetitive cycle that involves every type of *muda* as the tester and programmer try to isolate the fault. And the cycle often leads to frustration, blame, and low morale.

Bug Ping-Pong happens not because testers and developers are incompetent or lazy, but because software bugs are often truly hard to isolate. Some bugs may demonstrate themselves only after thousands of asynchronous events occur, and the exact repro sequence cannot be re-created deterministically. Bugs like this are usually found by manual or exploratory testing, not by test automation.

When a tester files a bug, VS 2010 automatically invokes up to six mechanisms to eliminate the guesswork from fault isolation:

1. All the tester's interactions with the software under test are captured in an *action log,* grouped according to the prescribed test steps (if any).
2. A *full-motion video* captures what the tester sees, time-indexed to the test steps.
3. *Screenshots* highlight anything the tester needs to point out during the sequence.
4. *System configurations* are automatically captured for each machine involved in the test environment.
5. An *IntelliTrace log* records application events and the sequence of code executed on the server, to enable future debugging based on this actual execution history.
6. *Virtual machine snapshots* record the state of all the machines in the test environment in their actual state at the time of failure.

Eliminating Bug Ping-Pong is one of the clearest ways in which VS 2010 reduces work in process and allows quick turnaround and small batches in testing. Another is test impact analysis, which recommends the highest-priority tests for each build, based both on completed work and historical code coverage. This is shown in more detail in Chapter 8, "Test."

Self-Managing Teams

A lot of ink has been used in the past 20 years on the concept of governance with regard to software development. Consider this quote from an IBM Redbook, for example:

> Development governance addresses an organization-wide measurement program whose purpose *is to drive consistent progress assessment* across development programs, as well as the use of *consistent steering mechanisms.* [Emphasis added.][24]

Most of the discussion conveys a bias that problems in software quality can be traced to a lack of central control over the development process. If only we measured developers' activities better, the reasoning goes, we could control them better. The Agile Consensus takes a very different attitude to command and control. Contrast the preceding quote with the following analysis:

> Toyota has long believed that *first-line employees* can be more than cogs in a soulless manufacturing machine; they *can be problem solvers, innovators, and change agents.* While American companies relied on staff experts to come up with process improvements, Toyota gave every employee the skills, the tools, and the permission to solve problems as they arose and to head off new problems before they occurred. The result: Year after year, Toyota has been able to get more out of its people than its competitors have been able to get out of theirs. Such is the power of management orthodoxy that it was only after American carmakers had exhausted every other explanation for Toyota's success—an undervalued yen, a docile workforce, Japanese culture, superior automation—that they were finally able to admit that *Toyota's real advantage was its ability to harness the intellect of "ordinary" employees.*[25]

The difference in attitude couldn't be stronger. The "ordinary" employees—members of the software team—are the ones who can best judge how to do their jobs. They need tools, suitable processes, and a supportive environment, not command and control.

Lean turns governance on its head, by trusting teams to work toward a shared goal, and using measurement *transparency* to allow teams to improve the flow of value and reduce waste themselves. In VS, this transparency is fundamental and available both to the software team and its stakeholders. The metrics and dashboards are instruments for the team to use to inspect its own process and adapt its own ways of working, rather than tools designed for steering from above.

Back to Basics

It's hard to disagree with Lean expert Jim Womack's words:

> The critical starting point for lean thinking is value. Value can only be defined by the ultimate customer.[26]

Similarly for software, the Agile Consensus changes the way we work to focus on value to the customer, reduce the waste impeding the flow, and transparently communicate, measure, and improve the process. The auto industry took 50 years to absorb the lessons of Lean, until customer and investor patience wore out. In mid-2009, on the day General Motors emerged from bankruptcy, CEO Fritz Henderson held a news conference in Detroit and said the following:

> At the new GM, we're going to make the customer the center of everything. And we're going to be obsessed with this, because if we don't get this right, nothing else is going to work.[27]

Six months later, when GM had failed to show suitable obsession, Henderson was out of a job. It may be relatively easy to dismiss the woes of Detroit as self-inflicted, but we in the software industry have carried plenty of our own technical debt, too. That technical debt has cost many a CIO his job, as well.

Summary

For a long time, Scrum creator Ken Schwaber has said, "Scrum is all about common sense," but a lesson of the past decade is that we need supportive tooling, too.[28] To prevent the practice from diverging from common sense, the tools need to reinforce the flow of value, reduce the waste, and make the process transparent. These Agile principles have been consistently reflected in five years of customer feedback that are reflected in VS 2010.

In practice, most software processes require a good deal of manual work, which makes collecting data and tracking progress expensive. Up front, such processes need documentation, training, and management, and

they have high operating and maintenance costs. Most significantly, the process artifacts and effort do not contribute in any direct way to the delivery of customer value. Project managers in these situations can often spend 40 hours a week cutting and pasting to report status.

In contrast, the business forces driving software engineering today require a different paradigm. A team today needs to embrace customer value, change, variance, and situationally specific actions as a part of everyday practice. This is true whether projects are in-house or outsourced and whether they are local or geographically distributed. Managing such a process usually requires an Agile approach.

And the Agile Consensus requires supportive tooling. Collecting, maintaining, and reporting the data without overhead is simply not practical otherwise. In situations where regulatory compliance and auditing are required, the tooling is necessary to provide the change management and audit trails. Making the handoffs between different team members as efficient as possible becomes more important than ever, because these handoffs happen so much more often in an iterative process. VS 2010 does that and makes Agile practices available to any motivated software team. The rest of this book describes the use of VS to support this paradigm.

In the next chapter, I look at the implementation of Scrum and other processes with VS. This chapter focuses on how VS represents the timeboxes and cycles. Chapter 3 pulls the camera lens a little further out and looks at product ownership broadly and the grooming of the product backlog, and Chapter 4 puts these topics together to discuss how to run the sprint using VS.

End Notes

1 James P. Womack and Daniel T. Jones, *Lean Thinking: Banish Waste and Create Wealth in Your Corporation* (New York: Free Press, 2003), 150.

2 James P. Womack, Daniel T. Jones, and Daniel Roos, *The Machine That Changed the World: How Japan's Secret Weapon in the Global Auto Wars Will Revolutionize Western Industry* (New York: Rawson Associates, 1990).

[3] "G.M. and Ford Channel Toyota to Beat Toyota," *New York Times,* March 7, 2010, BU1.

[4] www.agilemanifesto.org

[5] "5th Annual State of Agile Survey" by Analysis.Net Research, available from http://agilescout.com/5th-annual-state-of-agile-survey-from-version-one/.

[6] Dave West and Tom Grant, "Agile Development: Mainstream Adoption Has Changed Agility Trends in Real-World Adoption of Agile Methods," available from www.forrester.com/rb/Research/agile_development_mainstream_adoption_has_changed_agility/q/id/56100/t/2, 17.

[7] Available from www.pmi.org/Resources/Pages/Library-of-PMI-Global-Standards-Projects.aspx.

[8] Ken Schwaber, adapted from Ralph. D. Stacey, *Strategic Management and Organisational Dynamics, 2nd Edition* (Prentice Hall, 2007).

[9] Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (Upper Saddle River, NJ: Prentice Hall, 2001).

[10] Ken Schwaber and Jeff Sutherland, *Scrum: Developed and Sustained* (also known as the *Scrum Guide*), February 2010, www.scrum.org/scrumguides.

[11] Henrik Kniberg and Mattias Skarin, "Kanban and Scrum - making the most of both," InfoQ, 2009, www.infoq.com/minibooks/kanban-scrum-minibook.

[12] Eliyahu M. Goldratt, *The Goal* (North River Press, 1986).

[13] Gerald M. Weinberg, *Quality Software Management, Volume I: Systems Thinking* (New York: Dorset House, 1992).

[14] Kent Beck and Cynthia Andres, *Extreme Programming Explained: Embrace Change* (Boston: Addison-Wesley, 2003).

[15] Donald G. Reinertsen, *The Principles of Product Development Flow: Second Generation Lean Product Development* (Redondo Beach, CA: Celeritas Publishing, 2009).

[16] West, *op cit.*, 4.

[17] This variation of the diagram is available from http://msdn.microsoft.com/.

[18] David J. Anderson, *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results* (Upper Saddle River, NJ: Prentice Hall, 2004), 77.

[19] Schwaber and Sutherland, *op. cit.*

[20] Taiichi Ohno, *Toyota Production System: Beyond Large-Scale Production* (Cambridge, MA: Productivity Press, 1988).

[21] Mary B. Poppendieck and Thomas D. Poppendieck, *Lean Software Development: An Agile Toolkit* (Boston: Addison-Wesley, 2003).

[22] Kent Beck, "Tools for Agility," Three Rivers Institute, 6/27/2008, www.microsoft.com/downloads/details.aspx?FamilyId=AE7E07E8-0872-47C4-B1E7-2C1DE7FACF96&displaylang=en.

[23] Louis Brandeis, "What Publicity Can Do," in *Harper's Weekly,* December 20, 1913, available from www.law.louisville.edu/library/collections/brandeis/node/196.

[24] IBM IT Governance Approach: Business Performance through IT Execution, February 2008, www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg247517.html, 35.

[25] Gary Hamel, "The Why, What, and How of Management Innovation," *Harvard Business Review* 84:2 (February 2006), 72–84.

[26] Womack and Jones (2003), *op. cit.,* 16.

[27] All Things Considered, National Public Radio, July 10, 2009, www.npr.org/templates/story/story.php?storyId=106459662.

[28] Schwaber and Sutherland, *op. cit.*

Index

D

E

F

G

H-I

K-L

M

N-O

P

Q-R

S

T

U

V

W-X-Y-Z