Improve Productivity by Integrating Automation and Defect Prevention into Your Software Development Process

PARASOFT. We make software work.

This book presents an approach to software management based on a new methodology called Automated Defect Prevention (ADP). The authors describe how to establish an infrastructure that functions as a software "production line" that automates repetitive tasks, organizes project activities, tracks project status, seamlessly collects project data, and sustains and facilitates the improvement of human-defined processes. Well-grounded in software engineering research and in industry best practices, this book helps organizations gain dramatic improvement in both product quality and operational effectiveness.

Ideal for industry professionals and project managers, as well as upper-level undergraduates and graduate-level students in software engineering, **Automated Defect Prevention** is complete with figures that illustrate how to structure projects and contains real-world examples, developers' testimonies, and tips on how to implement defect prevention strategies across a project group.

DOROTA HUIZINGA, PhD, is the Associate Dean for the College of Engineering and Computer Science and Professor of Computer Science at California State University, Fullerton. Her publication record spans a wide range of computer science disciplines and her research was sponsored by the National Science Foundation, California State University System, and private industry.

ADAM KOLAWA, PhD, is the co-founder and CEO of Parasoft, a leading provider of Automated Error Prevention software solutions. Dr. Kolawa, a co-author of *Bulletproofing Web Applications*, has contributed to or written more than 100 commentary pieces and technical papers, and has authored numerous scientific papers.

To learn more about the ADP book or read a sample chapter, visit http://www.parasoft.com/adp_book.

IEEE

IEEE COMPUTER SOCIETY

WILEY-INTERSCIENCE
wiley.com

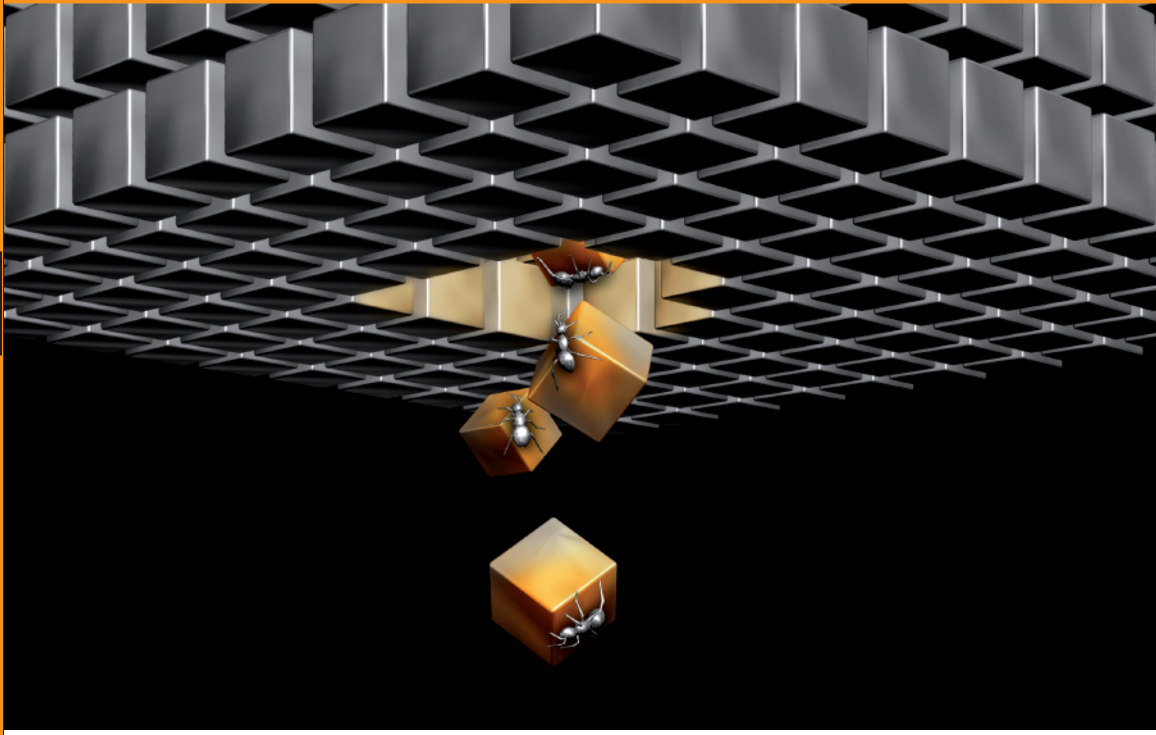
BICENTENNIAL 1807 WILEY 2007 BICENTENNIAL

ISBN 978-0-470-04212-0
90000
9 780470 042120

EXCERPTS FROM

Automated Defect Prevention

BEST PRACTICES IN SOFTWARE MANAGEMENT



Dorota Huizinga ▪ Adam Kolawa

EXCERPTS FROM

Automated Defect Prevention

BEST PRACTICES IN SOFTWARE MANAGEMENT

Dorota Huizinga | Adam Kolawa

WILEY-INTERSCIENCE
A John Wiley & Sons, Inc., Publication

WILEY-INTERSCIENCE
A John Wiley & Sons, Inc., Publication

ISBN: 978-0470-04212-0

List price $95.50

Ordering information:
To order online, go to http://www.wiley.com/go/adp.
For a 20% discount, use the promotion code TSADP.

To learn more about the ADP book or read a sample chapter, visit http://www.parasoft.com/adp_book.

WHAT IS THIS BOOKLET?

This booklet introduces the key concepts featured in the book Automated Defect Prevention: Best Practices in Software Development, which was co-authored by Parasoft CEO and co-founder Adam Kolawa along with Dorota Huizinga, Associate Dean and Professor at CSUF. It is designed to provide you an overview of what Automated Defect Prevention involves and how it can benefit your development team.

COMMENTS ABOUT AUTOMATED DEFECT PREVENTION

The best software bug is the bug that never, ever gets into the code. While we can all try to design zero-defect applications, the reality is that whenever there are programmers involved, you're going to have bugs. No set of best practices can prevent coders from making mistakes, just like no spell check can prevent me from typing "mistake" incorrectly. The real trick, therefore, is to catch defects as soon as they're made – quickly, painlessly, automatically.

In their authoritative new book, Dorota Huizinga and Adam Kolawa have done an admirable job defining a realistic methodology for implementing infrastructure for automatically preventing defects from getting into software. Is it simple? No, of course not. There's no silver bullet. But when the software industry is ready to journey toward zero-defect applications, the road will look like Huizinga and Kolawa's "Automated Defect Prevention."

– Alan Zeichick
Editorial Director, BZ Media's SD Times

Adam Kolawa is a seasoned practitioner of Quality Function Deployment (QFD) – he makes countless trips to every Parasoft client's site, and takes away the essential customer needs which get implemented at an astonishing rate into Parasoft's solutions.

The same level of scrutiny has been applied to "Automated Defect Prevention." This is a practical guide with realistic, useful and immediately applicable techniques that save money while improving developer skills. It's a must read for anyone serious about significant software quality improvement.

– Nigel DeFreitas
Application Architect, Insurance Services Office

WHY THIS BOOK IS IMPORTANT

Many software development managers and team leads want to make the software development process more productive and improve the quality of the software being delivered. Yet, they worry that such efforts will frustrate developers and impact the creativity that is vital for successful software projects. Fortunately, efficiency, quality. and creativity can peacefully coexist within the software development process. When the more mundane aspects of development are automated, developers can focus on the creative tasks they enjoy most…and still deliver better software in less time.

The Automated Defect Prevention (ADP) methodology described in the book reduces waste and inefficiency in the software development process, enabling developers to satisfy business goals without compromising their craft. The book explains how ADP can be applied to establish a continuous process that ensures quality tasks are not only deployed across every stage of the SDLC, but also ingrained into the team's workflow. Additionally, it covers how to leverage an automated infrastructure that drives this process to ensure that it remains on track and does not disrupt the team's workflow. Readers will learn how evolve their own development process into a continuous quality process that delivers greater productivity and significantly fewer software defects.

HOW THIS BOOK CAN HELP YOU

This book introduces ADP: a practical approach to software management through process improvement. This strategy is enabled by an infrastructure that automates repetitive tasks, tracks project status, and provides instant access to the information needed for informed decision making and process improvement. Applying ADP, you can evolve a sustainable quality process that delivers predictable outcomes.

ADP stands out from the current software landscape as a result of two unique features:

- Its approach to quality as a continuous process.
- Its far-reaching emphasis on automation.

It can be applied to any team, regardless of its structure, projects, or development method.

The overall goal of this book is to provide practical advice on how to implement ADP in a modern software development process. This is not a theoretical book on how to build software under ideal circumstances. Rather, it is a hands-on guide to solving real-world software project problems.

We recognize that most readers will not have the luxury of starting a new software process from scratch, so throughout the book, we pay special attention to phasing in ADP to existing projects and processes, and applying it to large complex code bases without overwhelming the team.

The focus throughout is on evolution, not revolution. We do not expect the readers to abandon current processes and workflows in search of a silver bullet solution that will solve all current problems. There is no such perfect development environment or development process that will be a panacea for all software development groups and projects. Consequently, rather than offer step-by-step implementation details for a one-size-fits-all solution, we describe the key practices that have helped many different teams improve their processes.

Reading this book, you will learn how to:

- Control and improve your existing development process.
- Increase agility while facing complexity and change.
- Ensure quality throughout the SDLC – from requirements definition, to design, to construction, to integration, to deployment.
- Collect objective data about the application and the processes used to build it, then leverage it for quality and process improvement.
- Instantly assess project quality and readiness.
- Determine if a release is on target, and how to get it back on track.
- Automate routine and repetitive tasks so the team can focus on more critical and challenging ones.

WHAT'S IN THE BOOK

In **Chapter 1,** the book begins by describing ADP and its goals as a response to the complexities of modern software development. First, it categorizes the goals into those affecting people, product, organization, process, and project (PPOPP). Next, it provides an overview of ADP's principles, practices and policies with a special focus on the role of defect prevention and automation. The last section briefly describes the evolution of software development process models from waterfall to iterative and agile. The objective is not to convince the readers to adopt a new development process, but rather to propose ideas for making existing processes more effective.

In **Chapter 2** introduces the ADP principles: establishment of the infrastructure, application of general best practices, customization of best practices, measurement and tracking of project status, automation, and incremental implementation of ADP's practices. These principles are explained and validated with research studies and realistic examples.

In **Chapters 3 and 5** discuss initial and extended planning together with the minimum and expanded infrastructure required for ADP. Supporting infrastructure is essential for ensuring that ADP practices can be properly measured, tracked, and automated, and thus can be implemented and sustained. Four core infrastructure elements are widely recognized as being critical to making software work: source control system, automated build system, problem tracking system, and automated reporting system. These four integrated elements form the backbone that allows the teams to maintain and improve their software development process. In addition to covering this technology infrastructure, these chapters also define the roles of people, describe basic and extended plan, and elaborate on parametric cost and schedule estimation models.

Chapter 4 addresses the requirements phase. Software defects that originate in the requirements phase and propagate further into the process are often the most costly to remove. Even issues that do not seem to be related to requirements can often be traced back to this phase of the software lifecycle — be it a poorly conceived requirement, an incorrectly implemented requirement, or a missing/incomplete requirement. A key point stressed in this chapter is the importance of determining which requirements will be most difficult to implement, prioritizing them, and building a prototype architecture.

In **Chapter 6** addresses architecture and design. The final product of this phase is a blueprint for implementing requirements with the focus on the most critical features. The chapter then discusses design policies, patterns, and architectures, which are different ways of achieving these goals. Tips on how design mistakes can be prevented and how they relate to requirements are provided throughout the chapter.

Chapter 7 covers construction, with a focus on the many practices that can prevent defects during this extremely error-prone phase. It describes best practices starting from coding standards, through test-driven development for modules and algorithmically complex units, to conducting unit tests before adding the code to the source control system. In addition, basic policies for use of the source control system and automated build system are outlined in this chapter.

Chapter 8 discusses how testing should be used to verify application quality at the various stages of its construction. It describes the recommended practices for testing together with the test progression criteria that help determine when development and testing should progress from level to level: from unit, through integration, system, and acceptance testing. Next, the chapter depicts a defect life cycle in the context of root cause analysis and prevention, and then describes basic policies for use of the problem tracking system and automatic regression testing system.

The first part of **Chapter 9** explains how measures introduced throughout the book can be used to identify emerging problems in the software product and/or process, and how these measurements should be used to help identify the root cause of the problem in question. The second part of this chapter discusses deployment to the staging system and then to the production system. Deployment of today's complex applications is an inherently difficult process.

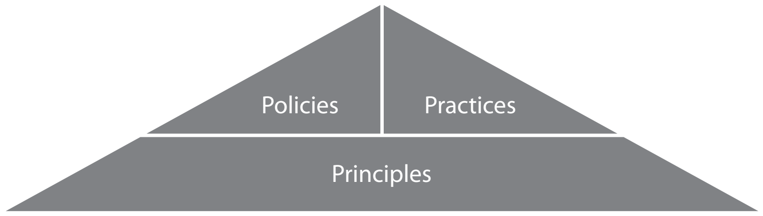
After discussing preventive strategies for making the deployment process less error-prone, the chapter offers guidance on automating it for different types of projects and environments.

Chapter 10 discusses how ADP supports industry initiatives that modern development teams commonly face. It describes the infrastructure and practices put in place for ADP that can be leveraged to support outsourcing initiatives and to address regulatory compliance requirements such as Sarbanes Oxley (SOX). It also shows how the ADP practices, paired with its automation and supporting infrastructure, makes quality improvement initiatives such as SEI – CMMI® practical and sustainable.

Finally, in **Chapter 11**, the book concludes with a case study that shows how industry leaders were able to apply some of the ADP concepts to improve their software development processes.

HOW ADP IS IMPLEMENTED

ADP is implemented by following a set of principles, practices, and policies. The principles are high-level laws that form the basis of ADP methodology, while the policies and practices comprise low-level development and management rules and procedures at varying granularity.



Principles, Policies, and Practices

Principles
Principles are the foundation of the ADP methodology. They are the basic laws that govern structuring and managing software projects through defect prevention. They correspond to ADP's goals at the highest level and they form the basis for the definition of practices and policies, which are directly applicable to software projects.

There are six ADP principles:

1. **Establishment of Infrastructure** – "Build a strong foundation through integration of people and technology"

2. **Application of General Best Practices** – "Learn from the mistakes of others"

3. **Customization of Best Practices** – "Learn from your own mistakes"

4. **Measurement and Tracking of Project Status** – "Understand the past and present to make decisions about the future"

5. **Automation** – "Let the computer do it"

6. **Incremental Implementation of ADP's Practices and Policies** – "Move carefully to minimize resistance and increase the chance of success"

Practices
Practices are functional embodiments of the principles. Depending on their level of granularity, best practices can pertain to entire projects, processes, or even individual tasks and people's daily activities. There are two types of best practices: general, which are based on issues common to all development projects, and customized, which are adopted by the organization to meet the needs of its unique projects. While the body of general best practices is already defined and well accepted by the industry, the body of customized best practices is created by the organization.

Policies
Policies are managerial embodiments of the principles. They mostly pertain to teamwork and define how the team should interact with technology. They are also used to assure that product and process related decisions are consistently applied through the entire team, and usually take the form of written documents. An example is a policy for use of a requirements management system, which should define how individuals and teams should use this system in order to most effectively organize and track product requirements.

SHORTCUT GUIDE TO ADP PRINCIPLES

Principle 1: Establishment of Infrastructure

This principle states that people's roles in a software team need to be defined to include active participation in and control of defect prevention in a project. It also stresses the necessity for technology used to both automate best practices, and to track and measure project status data.

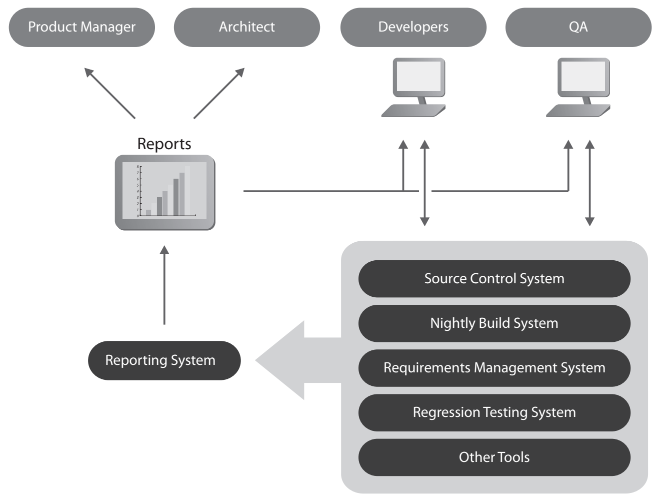
A robust development infrastructure includes the following components:

- **Source control system:** Having a source code repository is a prerequisite for nightly builds. All of the files needed for the build should be in the source control system, including build files, scripts, etc.—not just the source that is being built.

- **Nightly build system:** This system builds an application on a regular basis by automatically executing the required build steps at the scheduled time without any human intervention. This way, any problems are detected the next day (as opposed to a week or more later).

- **Requirements management system (and/or bug tracking system):** This is a repository for storing and tracking defects and issues, as well as the developers' ideas for improving the application. Defects are stored so they can be tracked, analyzed, and ultimately prevented from recurring. The system can also be used for storing feature requests and tracking feature and requirement changes. The end result is a central repository of information related to your software.

- **Regression testing system:** A regression system is any tool or combination of tools that can automatically run the core of the existing tests on the entire code base on a regular basis. Its purpose is to identify when code modifications cause unexpected faults, especially those faults that occur because a developer did not fully understand the internal code dependencies when modifying or extending code that previously functioned correctly. For example, it can be established by running automated testing tools from the command line using scripts or Ant with CruiseControl or Maven.



- **Additional test and analysis tools:** This can include tools for static analysis, code review automation, performance profilers, memory analyzers, coverage analyzers, etc. For example, this could include open source tools such as PMD, JUnit, CppUnit, NUnit, or automated tools such as Parasoft Jtest (for Java), C++test and Insure++ (for C/C++), .TEST (for .NET languages), and SOAtest (for SOA and Web).

- **Reporting system:** This system should be able to gather data from all these components (source control, nightly build, tests, etc) and present them in a visual way to enable analysis of status and trends. Parasoft GRS is an example of such a system.

Principle 2: Application of General Best Practices

Most software development projects share many of the same characteristics and pitfalls, even though each project and organization also encounters its own unique challenges. Consequently, ADP defines general best practices to prevent the software defects and human errors common to most development projects. The basic premise of this principle is to integrate industry well-accepted best practices (such as those identified for requirements change management, configuration management, or coding standards) into the lifecycle.

The set of general best practices is the product of software industry experts examining the most common errors and then developing rules designed to prevent these common errors. They represent a wealth of knowledge that was created through years of experience of many organizations. By adopting and, where possible, automating these „out-of-the-box" general practices, an organization can instantly progress from following the few best practices that it introduced over the years to a comprehensive set of standards that have been developed, tested, and perfected by industry experts. When teams are working on any type of project, it is always prudent to follow the practices and standards that industry experts have designed for the relevant application, language, platform, and technologies.

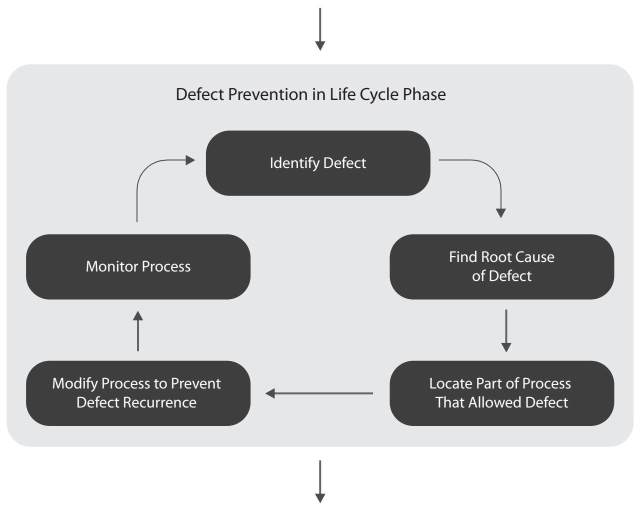
Best practices discussed in this book include:

- Requirements definition and management best practices
- Design best practices
- Static analysis (coding standards, data flow, metrics, etc.)
- Peer code review
- Unit testing
- Regression testing
- Security testing
- Message/protocol testing (SOA, Web, RIA, etc.)
- Load testing
- Deployment best practices

Principle 3: Customization of Best Practices

The customized best practices address the project- or organization-specific problems. Some practices are very valuable in one type of development context, but not applicable to others. Customized practices are necessary because many modern development projects have vastly different needs, depending on the nuances of the product. For example, the team working on embedded C software for a pacemaker, the team working on an e-commerce Web application, and the team developing a fast-action software game, will all have very different concerns. Thus, even though the same core general practices apply in most cases, for thorough defect prevention, these core practices should be supplemented with the appropriate set of specific practices to create a mix of general and customized practices best suited for the project's unique needs.

Thus, there is a need for a mechanism that provides for customization of the best practices in order to prevent project- or organization- specific problems. This mechanism is described by the third principle, which is based on the error prevention concept shown below. Each time a severe defect is discovered, a new customized practice should be defined. Once the new customized practice is defined, it has to become an integral part of the methodology and its application should be, if possible, both automated and seamless. The adherence to the customized practices should be monitored, so consequently, the development methodology would become increasingly defect-resistant.


Defect Prevention in Life Cycle Phase
Identify Defect
Find Root Cause of Defect
Locate Part of Process That Allowed Defect
Modify Process to Prevent Defect Recurrence
Monitor Process

Defect prevention applied to a phase in the software development process.

Principle 4: Measurement and Tracking of Project Status

To make informed decisions, management must be able to analyze measures reflecting project status information. These measures are quantitative representations of product or process attributes. They could be assigned specific, absolute values, such as the number of defects uncovered, or they could characterize the degree to which a system, or its component, possesses a given attribute. In the latter case, they are referred to as metrics. An example of a metric is a percentage of failed tests. These measures could also denote more general statistical indicators, such as confidence factors, derived from many basic measures. Indicators, for example, could provide information such as whether the project progresses according to its schedule and whether the costs are within the planned budget. At a more detailed level, many other project-essential statistics should be available; for example, the number of implemented requirements features, the number of failed tests, coverage of tests, or the number of defects and their severity, etc. Project indicators help in prompt identification of problems, so they can be remedied in a timely manner. Additionally, when observed over an extended period, those indicators can be used to assess product quality and its deployment readiness.

Consequently, continuous data gathering is critical in our methodology and it is achieved by having a seamless, automated reporting system capable of collecting and storing project data in a repository. Based on the data in this repository project status measures and metrics can be calculated, tracked, and plotted for management decisions. Thus, in our methodology, software processes are treated as statistical processes. It is important that each group review the values of these predefined measures, and then use statistical control limits and target average values to assess the status of the project.

Principle 5: Automation

Today's software systems are large, very complex, and typically consist of many sub-systems running on heterogeneous platforms. The trend is for software to continue to become more complex, notwithstanding the advances in modern Integrated Development Environments with automatic code generators, and widespread use of off the shelf components.

With such complex systems and many different cost-driven constraints, automation becomes a necessity for the delivery of quality software systems. Thus, Walker Royce in his top 10 principles of modern software management lists automation as a critical feature that facilitates or even encourages perpetual change in iterative processes. Although many managers hesitate to invest in auxiliary technology due to the high cost of licenses and training, the long-term benefits of automation largely outweigh these drawbacks.

These benefits include:

- Improved job satisfaction and productivity
- Improved product quality
- Facilitating human communication
- Helping to implement and verify best practices and organizational standards
- Facilitating control of the software processes by collecting measurement data

Principle 6: Incremental Implementation of ADP's Practices and Policies

While the ability to embrace change is important to the success of any business, it is a necessary survival skill in the software industry. This is because no other business relies on continuous change through learning, adaptation, and innovation like the software business does.

However, change is unsettling and it can be overwhelming to both the organization and the individuals. Change usually encounters resistance, since people tend to gravitate towards their low energy state called the "comfort zone". In addition, while working in the comfort zone people do not experience a great sense of accomplishment, but instead feel settled and secure.

Thus, to minimize the difficulties induced by yet another change and consequently encounter a resistance, ADP has to be introduced gradually to an organization: group-by-group and practice-by-practice.

PARASOFT—YOUR PARTNER IN IMPLEMENTING ADP
Delivering quality as a continuous process

For 20 years, Parasoft—led by ADP co-author Adam Kolawa—has been empowering organizations to deliver better business applications faster. We achieve this by leveraging ADP to deliver quality as a continuous process throughout the SDLC—not just QA. The result is a sustainable process that delivers greater productivity and significantly fewer software defects.

Parasoft Quality Solutions

Parasoft Quality Solutions are adopted by top companies because they deliver an end-to-end quality process that begins with a requirement and ends with the audit of a business process. By automating quality tasks at all layers of the application stack, the solutions significantly reduce the risks associated with developing new or changing applications. These solutions address multiple layers of the application and address application quality concerns such as:

- Security
- Reliability
- Performance
- Maintainability
- Process visibility and control

Parasoft solutions help development teams prevent errors and continuously test logical units of the application. They also help QA testers or business analysts focus on validating end-to-end business scenarios as an iterative process—not a quality task at the end of a development cycle. The entire quality process is supported by an infrastructure that automates key tasks, tracks project status, and provides instant access to the information needed for smart decision making and process improvement.

Parasoft solutions support the following components:

Error Prevention
Parasoft delivers an automated framework to ensure all software development activities meet uniform expectations around security, reliability, performance, and maintainability. We provide a foundation for producing solid code by exposing structural errors and preventing entire classes of errors. This initiates the continuous quality process, delivering greater productivity and significantly fewer software defects.

Continuous Regression Testing
Parasoft's continuous regression testing immediately alerts you when modifications impact application behavior. By providing a safety net that alerts the team when modifications impact application behavior, it enables rapid and agile responses to business demands, reducing the risk of change.

Functional Audit
Parasoft's continuous quality practices promote the reuse of test assets as building blocks to streamline the validation of changing business requirements. This enables your team to execute a more complete audit of your business application. The result is a reduced risk of business downtime, ensuring business continuity.

Process Visibility and Control
SDLC quality metrics are fragmented across key systems such as requirements, build, and source control management. Parasoft aggregates and correlates this system data, delivering a comprehensive view of your development processes. This process visibility facilitates continuous process improvement, increasing productivity and reducing cost.

Parasoft Automated Infrastructure Services

Parasoft services integrate and automate your SDLC to ensure that quality software can be produced consistently and efficiently. The resulting automated infrastructure improves development productivity and forms the foundation for a sustainable quality process.

Quality Solutions for

- SOA
- Java
- Web 2.0 and RIA
- .NET
- C and C++
- Embedded systems
- Agile, XP, TDD methodologies
- Enterprise integration
- Security
- Outsourcing engagements
- Distributed development
- ALM
- Quality initiatives – CMM-I, Six Sigma, ISO 9001

Compatible Software and Platforms

- Eclipse
- Rational Application Developer
- Microsoft Visual Studio
- Wind River
- ARM
- Borland
- IntelliJ
- Oracle
- BEA
- Software AG/webMethods
- IBM MQ-Series
- TIBCO
- Sonic
- IONA
- HP
- Other leading platforms

ABOUT THE AUTHORS

Dorota Huizinga, PhD, is the Associate Dean for the College of Engineering and Computer Science and Professor of Computer Science at California State University, Fullerton. Her publication record spans a wide range of computer science disciplines and her research was sponsored by the National Science Foundation, California State University System, and private industry.

Adam Kolawa is the co-founder and CEO of Parasoft, leading provider of solutions and services that deliver quality as a continuous process throughout the SDLC. He is considered an authority on the topic of software development and the leading innovator in promoting proven methods for enabling a continuous process for software quality. Kolawa has co-authored two books: Automated Defect Prevention: Best Practices in Software Management (Wiley, 2007) and Bulletproofing Web Applications (Wiley, 2001). He has also written or contributed to hundreds of commentary pieces and technical articles. In 2007, eWeek recognized him as one of the 100 Most Influential People in IT. Kolawa holds a Ph.D. in theoretical physics from the California Institute of Technology.

LEARNING MORE

To learn more about the ADP book or read a sample chapter,
visit http://www.parasoft.com/adp_book.

To learn how Parasoft can help you apply ADP as part of a comprehensive quality and process improvement strategy, **visit http://www.parasoft.com/adpsolutions.**

For 20 years, Parasoft has investigated how and why software errors are introduced into applications. Our solutions leverage this research to deliver quality as a continuous process throughout the SDLC. This promotes strong code foundations, solid functional components, and robust business processes. Whether you are delivering Service-Oriented Architectures (SOA), evolving legacy systems, or improving quality processes—draw on our expertise and award-winning products to increase productivity and the quality of your business applications.