

# Best practices: what testers need to know

Sources of useful, established methodologies and advice are not limited to standards organisations. Dr Adam Kolawa of Parasoft introduces some you might not have seen

*Testers can benefit from understanding both development and testing best practices.*

## Development best practices

When developers follow these, testers can spend more time focusing on high-level testing tasks and less time chasing after problems that developers should have prevented or fixed.

### 1. Defensive programming

Defensive programming is the practice of anticipating where failures can occur and then creating an infrastructure that tests for errors, reports when anticipated failures occur, and performs specified damage-control actions -- such as stopping program execution, redirecting users to a backup server, enabling debugging information that can be used to diagnose the problem, and so on. These defensive programming infrastructures are typically built by adding assertions to the code, implementing Design by Contract, developing software defensive firewalls, or simply adding code that validates user inputs. By applying defensive programming techniques, developers can detect problems that might otherwise go unnoticed, prevent minor problems from growing into disasters, and save themselves a lot of debugging and maintenance time in the long run.

*Hint: If the developers are performing defensive programming, they will be able to compile the code in two ways - with or without the defensive programming infrastructure.*

### 2. Code review

A code review is the process where the developers and architects meet and discuss code. Its purpose is to exchange ideas about how code is written, and to establish a consistent interpretation of code throughout the group. During these reviews, developers should be given the opportunity to explain their code to one another. Often, simply explaining the code helps developers identify problems and envision new solutions for previously troubling dilemmas. When the group members discuss the code, their discussion should focus on important issues such as algorithms, object-oriented programming, and class design.

*Hint: If developers are not complaining about code reviews, they probably are not performing them.*

### 3. Coding standard compliance

Coding standards are language-specific programming rules that greatly reduce the probability of introducing errors into applications. Coding standards originated from the intensive study of industry experts who analyzed how bugs were generated when code was written and correlated these bugs to specific coding practices; they took these correlations between bugs and coding practices and came up with a set of rules that prevent coding errors from occurring. In a team environment or group collaboration, coding standards ensure uniform coding practices, reducing oversight errors and the time spent in code reviews. When work is outsourced to a third-party contractor, having a set of coding standards in place ensures that the code produced by the contractor meets all quality guidelines mandated by the client company.

*Hint: If there is no system set up to scan code on a regular basis, the developers probably are not following coding standards.*

### Development best practices reading list

- Hunt, Andrew and David Thomas, *The Pragmatic Programmer*, 1999 (defensive programming, code review)
- Eldridge, Geoff. Java and Design by Contract (defensive programming). [www.elj.com/eiffel/feature/dbc/java/ge/](http://www.elj.com/eiffel/feature/dbc/java/ge/)
- Kolawa, Adam, Wendell Hicken, and Cynthia Dunlop, *Bulletproofing Web Applications*. 2001 (defensive programming, coding standards)
- Maguire, Steve, *Writing Solid Code*, 1993 (defensive programming)
- McConnell, Steve. *Code Complete*, 1993 (defensive programming, code review, coding standards)
- Meyer, Bertrand. *Object-Oriented Software Construction*, 2000 (defensive programming)

- Payne, Jeffrey E., Michael A. Schatz, and Matthew N. Schmid. *Implementing Assertions for Java*. Dr. Dobbs's Journal January 1998 (defensive programming)
- Plessel, Todd. *Design by Contract: A Missing Link in the Quest for Quality Software*: [www.elj.com/eiffel/dbc/](http://www.elj.com/eiffel/dbc/) (defensive programming)
- Meyers, Glenford J., *The Art of Software Testing*, 1979 (code review)
- Kernighan, Brian and P.J. Plauger. *The Elements of Programming Style*, 1988 (coding standards)

### Testing best practices

These are designed to help verify that the product really works and is solid.

#### 1. Understand the product architecture before you start testing the product

If you do not understand the architecture and inner workings of the product you are testing, you will not be able to anticipate where it is most error prone. As a result, you could overlook easy opportunities to uncover a large amount of errors in a small amount of time.

As testers gain experience, they learn that there are some parts of programs that are more error-prone than other. Generally, the most error-prone parts of a program are the interfaces between different modules. Why? Because different groups of developers work on different modules. These groups often misunderstand one another's intentions and assumptions, and errors typically result when their code interacts. Consequently, a lot of bugs are usually hidden in program interfaces.

Another trick is to learn which development groups worked on which program segments, and use those development groups' track records to anticipate which parts of the program are most error-prone. For example, if you know that Group C worked on a certain part of the program and that Group C usually produces code with a lot of errors, you might want to focus a large percentage of your testing efforts on the part

## ISEB Software Testing Certificates

These qualifications provide visible and objective evidence of an individual's knowledge and ability to perform software testing.

It incorporates the following:

- **The Foundation Certificate** – is for anyone with an interest in testing. It provides visible evidence that the individual understands the basics of software testing.
- **The New Practitioner Certificate** – is for experienced testing practitioners. This certificate demonstrates a depth of knowledge of testing topics and the ability to perform testing activities in practice.

[www.iseb.org.uk/pt](http://www.iseb.org.uk/pt)

For information contact Customer Support:

TEL +44 (0)1793 417542

Email: [isebenq@hq.bcs.org.uk](mailto:isebenq@hq.bcs.org.uk)

quoting reference number: 710/1003

THE BCS IS A REGISTERED CHARITY: NUMBER 292786



of the program created by Group C. Likewise, if you know that Group A almost always delivers clean, functional code, it is probably safe to spend a smaller percentage of your time testing the parts of the program that Group A worked on.

Just like a police detective needs to understand the entire situation before he can solve a murder mystery, you need to understand the entire situation to solve the mystery of "does this software really work"?

### 2. Anticipate potential misuses and verify how the software responds in those cases

Don't think that your job is done once you have verified that the software does what it is supposed to do. Users inevitably try to use software in unexpected ways—sometimes because they misunderstand how to use the product, sometimes because they see additional usages for the product, and sometimes because they want to launch security attacks through the product.

The specification is the best starting point for testing unexpected usages. If you don't have a specification, find one or write one yourself if needed. For each feature in the specification, try to imagine what unexpected paths could be taken by a new user exploring the program, an experienced user trying to maximize the program, and a hacker trying to manipulate the program. For example, what happens if the user tries to apply a tool to an unexpected type of source? If the user does not provide critical information? If the user designs and sends unexpected inputs in an attempt to gain access to privileged data or to gain control of a program?

The appropriate response to these unexpected situations depends on the program and the situation. In all cases, the response should be intelligent. For example, if the user does not provide critical information, it is better to have the program display a helpful dialog explaining the problem than simply to fail to perform the requested action.

### 3. Record clearly the procedure to reproduce each error found

For each problem that you detect, be sure to record detailed, unambiguous instructions for reproducing that problem, as well as a detailed description of the environment and context in which the problem occurred. If your bug report documentation is incomplete or confusing, two problems could occur.

One problem is that developers might not be able to reproduce the error and thus probably will not be able to fix the error. If the developer does manage to reproduce the error without proper instructions, he or she will have probably wasted a lot of time in the process.

Another problem is that you will not be able to verify effectively whether the developer's modification corrected the problem. If you don't test the repair using the exact same environment and procedures that produced the error in the first place, a passed test will not necessarily mean that the error was corrected.

### 4. Help the team prevent errors

Typically, when testers find errors, they add a report to the bug tracking system, the responsible developer tries to reproduce and repair the problem, then the tester must verify that the modification corrected the reported problem and did not introduce any new problems. This approach is not only time-consuming and costly, but also inefficient because it doesn't help prevent the same types of errors from recurring. Moreover, it causes the team to waste a significant amount of time, effort, and resources on the same types of errors thousands of times over.

Various methods have been proposed to help ensure that once an error is discovered it, and often other, similar errors, are prevented from recurring by an improvement to the development and/or testing

process. The key to these is that developers or testers should find each type of error only once. The knowledge the team gains from finding and analyzing errors should be used to improve the process so that you never encounter repeat occurrences of errors similar to those you have already found.

This process benefits the entire team by improving quality and reducing costs, but it is especially beneficial to testers because if developers are performing the required error prevention practices, testers won't need to repeatedly chase after errors that developers could have easily found or prevented and will have more time to dedicate to higher level verification tasks.

You can further error prevention not only by finding bugs, but also by helping the team architect, manager, and developers pinpoint the source of each error you uncover and by suggesting ways to prevent recurrences of that error.

For example, suppose that some of your load tests reveal that a heavy load stops the system. One course of action is to report the problem in the bug tracking system and hope that someone else figures out why it was caused and how to prevent it. A better course of action is to work with the development team to pinpoint the source of the error and reach a group consensus on how to prevent the error from recurring.

After the problem and resolution are identified, the architect and manager should determine how to implement and enforce the error prevention measure; this should not be responsibility of the tester.

However, if your testing indicates that a required error prevention measure is not being followed correctly, it's important to notify the architect or manager so that he or she can address the problem. **PT**