RBCS

TIME TESTED.
TESTING IMPROVED.
www.RBCS-US.com

The Use of Checklists in Design and Code Reviews

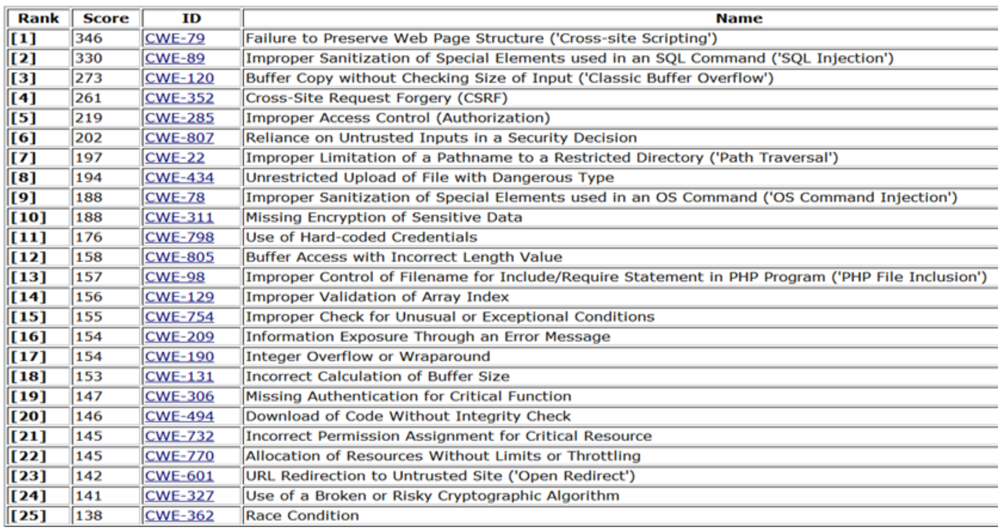
Rex Black and Jamie Mitchell

[The following is adapted from Rex Black and Jamie Mitchell's soon-to-be-released second edition of *Advanced Software Testing: Volume 3*, a book for ISTQB Advanced Technical Test Analyst candidates and other technically oriented testers.]

Technical test analysts have a role in defining, applying, and maintaining design and code review checklists. Why are checklists important? For one thing, checklists also serve to ensure that the same level and type of scrutiny is brought to each author's work. There can be a tendency of review participants to defer to a senior person, and thus that person's work, when in fact everyone is fallible and we all make mistakes. Conversely, a less-senior or more-insecure person might feel threatened by the review. Regardless of the individual author and his or her skills, there is nothing personal about locating potential problems and improvements for their work from a checklist. The checklists serve as a valuable leveling and depersonalizing tool, both factually and psychologically.

More importantly, though, checklists serve as a repository of best practices—and worst practices—that can help the participants of a review remember important points during the review. The checklist frees the participants from the worry, "What if I forget some critical issue or mistake we've made in the past?" Instead, the checklist gives general patterns and anti-patterns to the participants, allowing them to ask instead, "How could this particular item on the checklist be an important consideration for this work product that we're reviewing?"

This idea of including not only best practices but also worst practices is important. Let's illustrate with an example. Consider the *Common Vulnerabilities and Exposures* list from MITRE, the Federally-funded non-profit, shown in Figure 1. If you are involved in reviewing code where security is important—and security almost always is important now—the items here can be incorporated into your code checklist and/or your static code analysis tools to ensure that dangerous coding mistakes are not happening.



Rank	Score	ID	Name
[1]	346	CWE-79	Failure to Preserve Web Page Structure ('Cross-site Scripting')
[2]	330	CWE-89	Improper Sanitization of Special Elements used in an SQL Command ('SQL Injection')
[3]	273	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	261	CWE-352	Cross-Site Request Forgery (CSRF)
[5]	219	CWE-285	Improper Access Control (Authorization)
[6]	202	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[7]	197	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[8]	194	CWE-434	Unrestricted Upload of File with Dangerous Type
[9]	188	CWE-78	Improper Sanitization of Special Elements used in an OS Command ('OS Command Injection')
[10]	188	CWE-311	Missing Encryption of Sensitive Data
[11]	176	CWE-798	Use of Hard-coded Credentials
[12]	158	CWE-805	Buffer Access with Incorrect Length Value
[13]	157	CWE-98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
[14]	156	CWE-129	Improper Validation of Array Index
[15]	155	CWE-754	Improper Check for Unusual or Exceptional Conditions
[16]	154	CWE-209	Information Exposure Through an Error Message
[17]	154	CWE-190	Integer Overflow or Wraparound
[18]	153	CWE-131	Incorrect Calculation of Buffer Size
[19]	147	CWE-306	Missing Authentication for Critical Function
[20]	146	CWE-494	Download of Code Without Integrity Check
[21]	145	CWE-732	Incorrect Permission Assignment for Critical Resource
[22]	145	CWE-770	Allocation of Resources Without Limits or Throttling
[23]	142	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[24]	141	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[25]	138	CWE-362	Race Condition

Figure 1: List of top 25 security vulnerabilities from CVE website

Generally, what should go into a checklist? If there are common templates, style-guides, variable naming rules, or word-usage standards in the company for certain types of work products, the checklists should reflect those guidelines. Of course, the verification of some of these guidelines can be implemented in static analysis tools. In that case, the checklist or the review entry criteria can simply mention that successful completion of the static analysis is a pre-requisite to the review.

Often it is the case that, for particular organizations or for particular systems, there are specific quality characteristics such as security and performance that are important and should be verified. So, checklists can include specific sections for each such characteristic, along with good and bad things to watch for. If these sections are applicable only to certain projects or products, the checklist should give guidance on when the items in a given section are important to consider.

While there are many useful examples of checklists for various kinds of work products, some of which we'll present below, you should plan to maintain and evolve your organization's checklists over time. For example, the study of defects found in testing and in production can reveal that some of these defects could have been found during an earlier review, had the checklist contained appropriate items to look for.

When building on known best practices and worst practices in checklists you've adopted, keep in mind the following factors:

- Product: Different products have different needs. As an extreme example of these contrasts, a smartphone application that allows users to play an entertaining and engaging game is a very different proposition than a mainframe application that handles thousands or even millions of mission-critical transactions per week.
- People: Your team will have different strengths and weaknesses, which are often amplified by the nature of the product. If your team tends to make the same types of mistakes over and over, these should be part of your checklist, regardless of whether these mistakes are common in software engineering in general. In addition, your checklist should mention the value of diverse viewpoints in reviews. People with the same viewpoints will tend to find the same defects.
- Process: The level of formality of your processes will influence the way checklists are documented and used. If you are using lightweight, informal processes, a checklist obtained from a company that follows formal, documentation-heavy processes must be adopted accordingly. In addition, the software development lifecycle matter.
- Tools: If you are able to use static analysis tools to detect certain kinds of defects prior to the review, then the checklist can simply note whether the tools have been applied.
- Priorities: What matters most to technical stakeholders, business stakeholders, users, customers, and testers should be considered as the checklist evolves.
- History: You should look for anti-patterns in your defects. These should be part of your checklist. You should also look for successful examples—patterns—in your past releases. What does your organization do well? What does your organization do poorly?
- Quality characteristics: You should look for ways in which your organization has successfully delivered various technical and domain quality characteristics. Can you see common elements in those successes? How about in the failures?
- Testability: You should consider factors that make it easier, harder, or plain impossible to evaluate whether a requirement, user story, use case, design element, architectural attribute, or other attribute is or is not satisfied. While testability as a quality characteristic is not often the highest priority, it should be considered as part of any checklist. If you can't determine whether something works or not, that's going to be a problem during dynamic testing. Not only will there be a potentially large number of tests for a requirement with testability issues, it will also be hard to distinguish false positives (and false negatives) from actual defects.

Finally, a caveat: While checklists are very useful, a risk of a checklist is that it will focus attention on verification (“are we building the product right?”) versus validation (“are we building the right product?”). Checklists will tend to bring your thinking into a verification mindset. Remember to think outside of the checklist to look at whether the product will actually solve the end users’ problems.