What Do We Know about Defect Detection Methods?

Per Runeson, Carina Andersson, and Thomas Thelin, *Lund University*
Anneliese Andrews, *University of Denver*
Tomas Berling, *Ericsson Microwave Systems*

A survey of defect detection studies comparing inspection and testing techniques yields practical recommendations: use inspections for requirements and design defects, and use testing for code.

Detecting defects in software product development requires serious effort, so it's important to use the most efficient and effective methods. *Evidence-based software engineering* can help software practitioners decide which methods to use and for what purpose.[1] EBSE involves defining relevant questions, surveying and appraising available empirical evidence, and integrating and evaluating new practices in the target environment.

This article helps define questions regarding defect detection techniques and presents a survey of empirical studies on testing and inspection techniques. We then interpret the findings in terms of practical use.

The term *defect* always relates to one or more underlying faults in an artifact such as code. In the context of this article, defects map to single faults. Thus, we use the terms *defect* and *fault* interchangeably, as have many of the authors whose work we refer to.

What influences the choice of method?

The choice of defect detection method depends on factors such as the artifacts, the types of defects they contain, who's doing the detection, how it's done, for what purpose, and in which activities. Factors also include which criteria govern the evaluation.

These factors show that many variations must be taken into account. When you search the evidence for the the pros and cons of using some defect detection method, you must choose specific levels of these factors to guide the appraisal of empirical evidence.

Artifact

Which artifact are you assessing? Requirements? Design? Code? Testing requires an executable representation—that is, code—while inspection can apply to any artifact. Most experiments, by necessity, use small, artificial artifacts. Using industrial artifacts improves the study's generalizability but also leads to more confounding factors.

Types of defects

What types of defects do the artifacts contain? There's a big difference between grammatical errors in code and missing requirements in a requirements specification. Testing and inspection methods might be better or worse for different types of defects. In this article, we first classify defects on the basis of their origin: *requirements*, *design*, or *code*. Second, many empirical studies categorize defects along two dimensions, as Victor Basili and Richard Selby proposed.[2] The first dimension classifies defects as either an *omission* (something is missing) or a *commission* (something is incorrect), while the second dimension defines defect classes according to their technical content.[2,3] Other classifications focus on the defect's severity in terms of its impact for the user: *unimportant*, *important*, or *crucial*.[4]

Actor

Who's the reviewer or tester? A freshman student in an experimental setting? An experienced software engineer in industry? What's the incentive for doing a good job in an empirical study, which isn't part of a real development project?[5] To confound matters further, experienced students have been observed to outperform recently graduated engineers.[6]

Technique

Which techniques are you using for inspection and testing respectively? Because there are many techniques for each, we refer to inspection and testing as *families* of verification techniques. For testing, we distinguish between structural (white box) and functional (black box) testing.

Purpose

What's the purpose of the inspection and testing activity? The activity might contribute to *validation*—that is, assuring that the correct system is developed—or to *verification*—that is, assuring that the system meets its specifications—or to both. The primary goal of both inspection and testing is to find defects, but more specifically, is it to *detect* a defect's presence, for later isolation by someone else, or is it to *isolate* the underlying fault? Testing reveals a defect's presence through its manifestation as a failure during execution. Inspections point directly at the underlying fault.

Furthermore, there are secondary purposes for inspection and testing. Inspections might contribute to knowledge transfer, for example, whereas testing in a test-driven design setting produces test specifications that also might constitute a detailed design specification.[7]

Table 1

Defect origins and detection

	Defect detection activity					
Phase of defect origin	Requirements inspection	Design inspection	Code inspection	Unit test	Function test	System test
Requirements	1	2	2	2	2	1
Design	N/A	1	2	2	1	2
Coding	N/A	N/A	1	1	2	2

Defect detection activities

A defect might originate in one development stage and be detected in the same or a later stage. For instance, a missing interface in a design specification could propagate to the coding stage, resulting in missing functionality in the code. You might detect this design defect during a design inspection, code inspection, unit test, function test, or system test. Because defect detection focuses on abstraction levels, we consider that *primary* defect detection activities are at the same level of abstraction and *secondary* defect detection activities are at a different level. For example, for design defects, design inspection and functional testing are primary activities, and code inspection and unit testing are *secondary* defect detection activities.

Table 1 illustrates primary and secondary defect detection activities for defects originating from requirements, design, and coding (listed in column 1). Cells numbered 1 represent primary defect detection activities, and cells numbered 2 classify secondary activities.

Evaluation criteria

What are the criteria for selecting techniques? Should you choose the most effective or the most efficient method? *Efficiency* in this context means the number of defects found per time unit spent on verification, and *effectiveness* means the share of the existing defects found.

Survey of empirical studies

Available sources of empirical evidence are experiments and case studies. Experiments provide good internal validity.[8] They can manipulate the investigation's context and control many parameters. However, achieving high ex-

Table 2

Surveyed empirical studies on inspection versus testing

Study and year	Type	Technique	Artifact	Number of defects of certain types*	Actors	Purpose	Result†
Hetzel,[17] 1976	Experiment	Functional test vs. structural test and inspection	Code modules (PL/1) 3 programs, 64–170 statements each	–	39 students	Detection	Effectiveness: testing > inspection
Myers,[18] 1978	Experiment		Code (PL/1) 63 statements	15 defects	59 professionals	Detection	Effectiveness: inspection = testing; complementary, but different for some classes of defects
Basili and Selby,[2] 1987	Experiment		Code (Fortran, Simpl-T) 169, 145, 147, and 365 LOC)	4 programs, total 34 defects Om/com: 0/2 ini, 4/4 cmp, 2/5 cnt, 2/11 int, 2/1 d, 0/1 cos	32 professionals + 42 advanced students	Detection	Effectiveness and efficiency depend on software type
Kamsties and Lott,[19] 1995	Experiment (replication)		Code (C) 211, 248, and 230 LOC	3 programs, 6/9/7 defects 0/2/0 ini, 0/0/1 cmp, 3/2/3 cnt, 0/3/0 int, 2/1/2 d, 1/1/1 cos	27 and 15 students in replications 1 and 2, respectively	Detection and isolation	Effectiveness: no significant difference Efficiency: testing > inspection
Roper et al.[20] 1997	Experiment (replication)		Code[19]	3 programs, 8/9/8 defects	47 students	Detection	Effectiveness: no significant difference Efficiency: testing > inspection Combination better
Laitenberger,[11] 1998	Experiment	Inspection, then structural testing	Code (C) 262 LOC	13 defects, 2 ini, 4 cnt, 2 cmp, 2 cos, 3 d	20 students	Detection	Not complementary
So et al.,[12] 2002	Experiment	Voting, testing, self-checks, code reading, data-flow analysis, Fagan inspection	Code (Pascal) 8 programs, 1,201–2,414 LOC each	Experiment 1: 270 major faults Experiment 2: 179 major faults	26 and 15 students in experiments 1 and 2, respectively	Detection	Effectiveness: voting > testing > inspection; complementary Efficiency: testing < inspection
Runeson and Andrews,[13] 2003	Experiment	Inspection vs. structural testing	Code (C) 190 and 208 LOC each	9 in each version 0/1 ini, 5/0 cnt, 1/0 int, 0/4 cmp, 1/2 cos, 1/2 d	30 students	Detection and isolation	For detection: testing > inspection For isolation: inspection > testing
Juristo and Vegas,[14] 2003	Experiment (replication)	Functional test vs. structural testing and inspection	Code[19] + one new program	4 programs, 9 defects each Om/com: 1/2 ini, 2/2 cnt, 1/1 cos	196 students	Detection	Effectiveness: different for different fault types Testing > inspection
			Code[19]	3 programs, 7 defects each Om/com: 1/1 ini, 1/1 cnt, 0/1 cmp, 1/1 cos	46 students		
Andersson et al.,[15] 2003	Experiment	Inspection vs. functional testing	Design (text) 9 pages, 2,300 words	2 design document versions, 13/14 defects, 3/4 crucial, 5/6 important, 5/4 unimportant	51 students	Detection	Inspection > testing Different faults found (one version)
Conradi et al.,[16] 1999	Case study	Inspection, desk check, test	Design (code)	1,502 and 6,300 in two projects, respectively	Professionals		Inspection > testing
Berling and Thelin,[3] 2003	Case study	Inspection, unit test, subsystem and system test	Requirements, design, code	244 (45 likely to propagate to code)	Professionals	Isolation	Little overlap between testing and inspection

** om = omission, com = commission, ini = initialization, cmp = computation, cnt = control, int = interface, d = data, cos = cosmetic*

† "x > y" means x is better than y

Table 3

Average values of effectiveness and efficiency for defect detection

		Study	Inspection effectiveness (%)*,†	Inspection efficiency†,‡	Testing effectiveness*,§	Testing efficiency‡,§	Different faults found
Experiments	**Code**	Hetzel[17]	37.3	–	47.7; 46.7	–	–
		Myers[18]	38.0	0.8	30.0; 36.0	1.62; 2.07	Yes
		Basili and Selby[2]	54.1	Dependent on software type	54.6; 41.2	–	Yes
		Kamsties and Lott[19]	43.5 50.3	2.11 1.52	47.5; 47.4 60.7; 52.8	4.69; 2.92 3.07; 1.92	Partly (for some types)
		Roper et al.[20]	32.1	1.06	55.2; 57.5	2.47; 2.20	Yes
		Laitenberger[11]	38	–	9#	–	No
		So et al.[12]	17.9, 34.6	0.16; 0.26	43.0	0.034	Yes
		Runeson and Andrews[13]	27.5	1.49	37.5	1.8	Yes
		Juristo and Vegas[14]	20.0 –	– –	37.7; 35.5 75.8; 71.4	– –	Partly (for some types)
	Design	Andersson et al.[15]	53.5	5.05	41.8	2.78	Yes for one version, no for the other
Case studies		Conradi et al.[16]	–	0.82	–	0.013	–
		Berling and Thelin[3]	86.5 (estimated)	0.68 (0.13)	80	0.10	Yes

* *Percent of the artifact's defects that are detected.*
† *Single entries involve code reading; multiple entries in one cell are reported in this order: code reading, Fagan inspection.*
‡ *Detected defects per hour.*
§ *Single entries involve functional testing; multiple entries in one cell are reported in this order: functional test, structural test.*
# *Testing is conducted in sequence after the inspection.*

ternal validity in experiments is difficult. Case studies have a more realistic context but by nature are subject to confounding factors. A case study's generalizability depends on how similar it is to an actual situation. No single experiment or case study can provide a complete answer, but a collection of studies can contribute to understanding a phenomenon.

Many empirical studies have investigated defect detection techniques, inspections, and testing in isolation. Aybüke Aurum, Håkan Petersson, and Claes Wohlin summarize 25 years of empirical research on software inspections, including more than 30 studies that investigated different reading techniques, team sizes, meeting gains, and so on.[9] Similarly, Natalia Juristo, Ana Moreno, and Sira Vegas summarize 25 years of empirical research on software testing based on more than 20 studies.[10] They compare testing within and across so-called "families" of techniques. Despite the large number of studies, they conclude that our collective knowledge of testing techniques is limited.

We know even less about the relationship between inspection and testing and the effects of combining them. Oliver Laitenberger summarizes five experimental studies comparing inspection and testing methods as well as results from one experiment he conducted that combined inspection and testing.[11] Later, four experimental studies[12–15] and two case studies[3,16] added to the knowledge on inspections versus testing.

We used these 12 studies—nine experiments on code defects, one experiment on design defects, and two case studies on a comprehensive defect detection process—to search for evidence that would help developers choose between defect detection methods. We followed the procedures in the EBSE framework that Tore Dybå, Barbara Kitchenham, and Magne Jørgensen recently presented.[1] Table 2 summarizes empirical studies involving both inspection and testing, and table 3 presents data from the studies. We report the experiments' outcomes and relate them to the results of the case studies conducted. The issues covered include

- requirements defects,
- design defects,
- code defects,
- different defect types, and
- efficiency versus effectiveness.

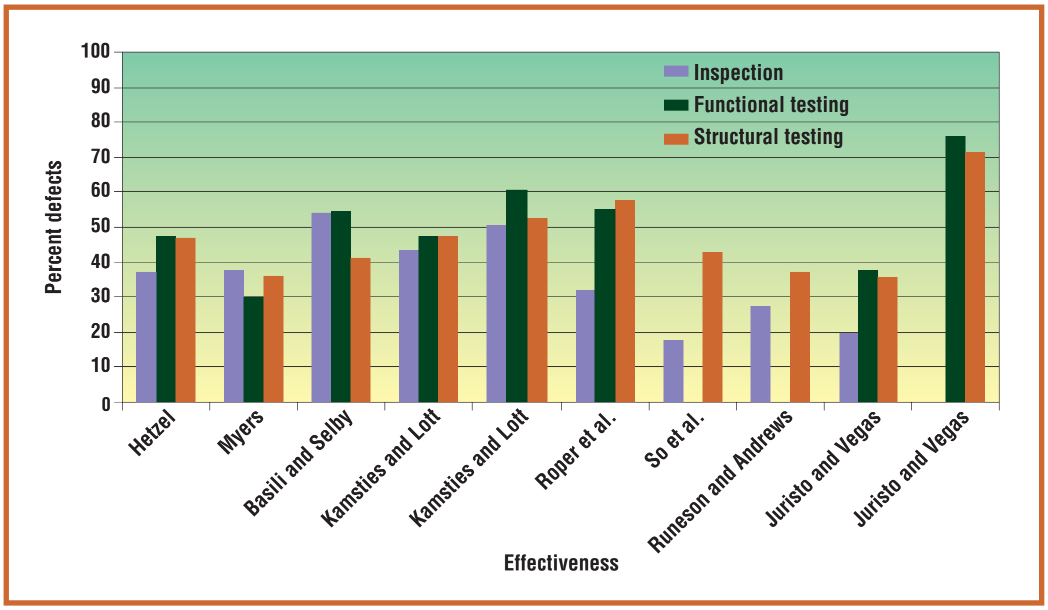

Figure 1. Average effectiveness of techniques for code defect detection.

Requirements defects

Several experiments compared different requirements inspection methods,[9] but none compared one to a testing method. The choice between requirements inspection and system testing is quite obvious and needs no experiments: spending effort up front to establish a good set of requirements is more efficient than developing a system on the basis of incorrect requirements and then reworking it. The case study by Tomas Berling and Thomas Thelin supports this assumption.[3]

Design defects

The key question regarding design defects is whether inspecting design documents or testing the implemented function is more efficient. Carina Andersson and her colleagues addressed this issue in one experiment.[15] They observed defect detection in a design specification and in a log from function test execution; they used experimental groups that varied greatly.

Inspections were significantly more effective and efficient than testing. The study's participants found more than half of the defects (53.5 percent) during inspection and fewer (41.8 percent) during testing (see table 3). Efficiency was five defects per hour for inspection and fewer than three per hour for testing.

The analysis didn't take into account rework costs. A defect detected during design inspection is much cheaper to correct than one detected in function testing, because the latter involves reworking the design and code. This implies even greater efficiency for design inspections compared to functional testing.

Berling and Thelin confirmed these results in their industrial case study, including five incremental project iterations.[3] Inspections detected on average 0.68 defects per hour, while testing detected 0.10 defects per hour (see table 3). For the fraction of faults that they estimated to propagate into code, the rate was 0.13 defects per hour. This case study also reports slightly higher effectiveness for inspections, but the differences are small. Reidar Conradi, Amarjit Singh Marjara, and Børge Skåtevik had similar results in their case study: they reported 0.82 defects per hour in design inspection and only 0.013 defects per hour in function testing.[16] So, the empirical data support design inspections as a more efficient means for detecting design defects.

Code defects

Most of the experiments investigated code defects. However, they revealed no clear answer as to whether code inspection or testing—functional or structural—is preferable.

James Miller[21] attempted to analyze the results from the first five studies (see table 2), but variation among the studies was too large to apply meta-analysis. So, we simply rank the techniques with respect to their effectiveness and efficiency. Figure 1 presents effectiveness—that is, the percentage of total defects that the different techniques found.

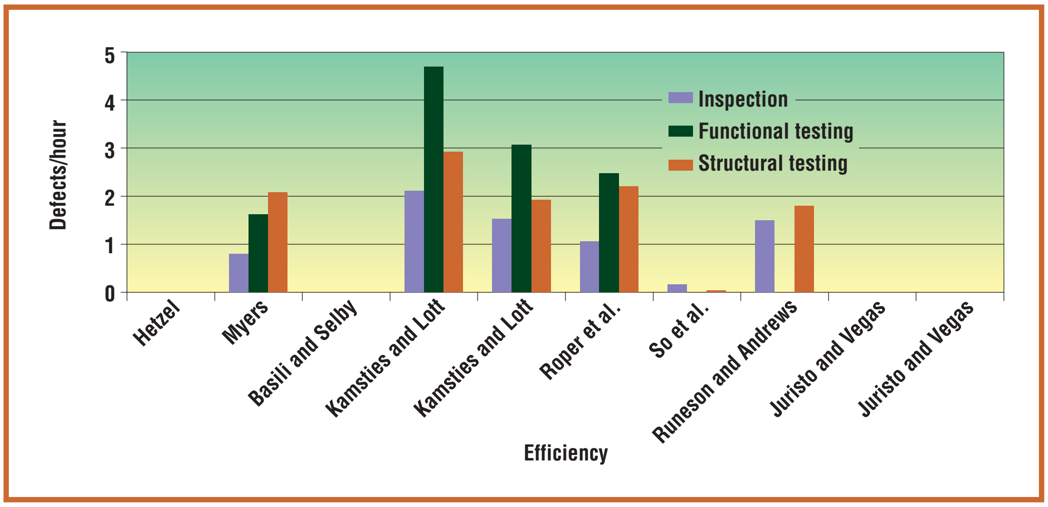

Figure 2. Average efficiency of techniques for code defect detection.

No technique emerges as a clear winner or loser. All three techniques are ranked most effective in at least one study *and* least effective in at least one study. However, except in one case, all researchers who compared the three techniques found the difference between the average effectiveness of the first and second techniques to be smaller than between the second and third. The data doesn't support a scientific conclusion as to which technique is superior, but from a practical perspective it seems that testing is more effective than code inspections.

Figure 2 presents the efficiency ranking—that is, the number of defects found per time unit—for those studies reporting efficiency data. (For the others, the space is left blank.) The efficiency rankings differ from the effectiveness rankings. Glenford Myers' study[18] ranked inspection most effective and least efficient, while a study by Sun Sup So and his colleagues[12] ranked testing most effective and least efficient. The study by Marc Roper, Murray Wood, and James Miller[20] ranked functional testing most effective and structural testing most efficient, but the differences are small.

The studies by Erik Kamsties and Christopher Lott[19] and by Per Runeson and Anneliese Andrews[13] distinguish between *detection* and *isolation* of defects. In the former study, this moved inspections to the second rank. In the latter study, this distinction didn't change the rank between the two studied techniques. The study by Roper, Wood, and Miller[20] also distinguishes between detection and isolation, which improved inspection's performance, but it still ranked third.

We caution that these studies were conducted in isolation and thus didn't take into account secondary issues such as the information-spreading effects of inspections, the value of unit-test automation, cost, and intrinsic values of a test suite or test-driven design.

Different defect types

The studies vary widely in the types of defects the artifacts contain. If that's indeed an important factor, then the absence or presence of different types of defects might affect efficiency and effectiveness. Table 2, column 5 shows the types of defects in each study, if they were known.

Comparing the studies is difficult. First, only five of the studies report defects by the same type scheme and can be used to investigate whether the differences depend on the fault type, not only on the technique. Second, the frequencies of the different defect types vary widely among the remaining studies. Third, only a fraction of defects are found; so, while we might know each artifact's defect type frequencies, we might not know the defect type frequencies of the defects that were found. Given the low proportion of defects found, this makes cross-study comparison difficult, if not impossible. Fourth, classification schemes involve subjective judgment that can confuse classification results.

Few studies investigate the relationship between defect types and inspection versus testing. One study found a statistically significant difference between defect types discovered by inspection and by testing.[13] This implies that the techniques' performance is sensitive to the

The studies indicate pros and cons for the two families of defect detection techniques, but no clear winner.

defect type. The same study indicates that subjects tend to prefer testing over inspection; this is a piece of qualitative information that should be taken into account.

Kamsties and Lott found two significant differences in performance between defect types; inspections gave worse results in finding omissions type faults and control class faults.[19] Juristo and Vegas replicated this experiment twice.[7] In the first replication, cosmetic faults were most difficult to find, irrespective of technique. However, defect type didn't affect code inspection, contradicting an earlier study by Basili and Selby.[2] Functional testing performed better than structural testing for faults of omission, and testing techniques performed better than inspection. Basili and Selby couldn't detect a clear pattern as to which technique detects which defect types more easily.

In summary, the jury is still out as to the effect of defect types on the performance of inspection versus testing.

Effectiveness and efficiency

Absolute levels of effectiveness of defect detection techniques are remarkably low. In all but one experimental study, the subjects found only 25 to 50 percent of the defects on average during inspection, and slightly more during testing (30 to 60 percent). This means that on average, more than half the defects remain! The Berling and Thelin case study reported 86.5 percent effectiveness for inspections and 80 percent effectiveness for testing.[3] However, these are based on an estimated number of defects that the technique could possibly find, not on the total number of defects in the documents.

The experimental studies found 1 to 2.5 defects per hour. The size of the artifacts was at most a few hundred lines of code. This is small from an industrial perspective, where professionals deal with more complex artifacts, struggle with more communication overhead, and so on. Consequently, the efficiency in the industrial case studies is lower: 0.01 to 0.82 defects per hour. The variation is also much larger, which might be due to different company measures of efficiency.[3]

The practical implication of the primary defect detection methods' low effectiveness and efficiency values is that secondary detection methods might play a larger role than the surveyed empirical studies concluded.

Validity of the empirical results

The studies we summarized indicate pros and cons for the two families of defect detection techniques, but no clear winner. But how valid are the results?

We can analyze threats to the validity of empirical studies along four dimensions: *internal*, *conclusion*, *construct*, and *external* validity.[8] From a practitioner point of view, external validity is the most important.[22] From a researcher point of view, internal validity is traditionally considered the key to successful research. However, it's important to balance all dimensions of validity to achieve trustworthy empirical studies.

The surveyed experimental studies' internal validity seems quite high. Established researchers conducted the studies, mostly in student environments with experienced students. On the other hand, the inconclusive results indicate the presence of factors that weren't under experimental control. By using established analysis methods, the studies also seem to limit conclusion validity threats.

For case studies, on the other hand, high internal and conclusion validity is harder to achieve. Confounding factors might interact with the factor under study and threaten internal validity. Because data collection is often set up for purposes other than empirical study, conclusion validity might be threatened.

Both experiments and case studies potentially threaten construct validity, because they use different instantiations of defect detection methods to represent test and inspection techniques. The studies listed in table 2 present large variations within as well as between the families of techniques, both in experiments and case studies.

The major threat for experiments is external validity, because they're conducted on small artifacts, mostly with students as subjects. The case studies suffer less from external threats, although there's a risk that the conditions specific to a particular case greatly influence the outcome.

What's the answer?

Our analysis of existing empirical studies showed no clear-cut answer to the question of which defect detection method to choose. However, from a practical viewpoint, these findings tend to be true:

- For *requirements defects*, no empirical evidence exists at all, but the fact that costs for requirements inspections are low compared to implementing incorrect requirements indicates that reviewers should look for requirements defects through inspection.
- For *design specification defects*, the case studies and one experiment indicate that inspections are both more efficient and more effective than functional testing.
- For *code*, functional or structural testing is ranked more effective or efficient than inspection in most studies. Some studies conclude that testing and inspection find different kinds of defects, so they're complementary. Results differ when studying fault isolation and not just defect detection.
- *Verification*'s effectiveness is low; reviewers find only 25 to 50 percent of an artifact's defects using inspection, and testers find 30 to 60 percent using testing. This makes secondary defect detection important. The efficiency is in the magnitude of 1 to 2.5 defects per hour spent on inspection or testing.

Several studies call for replication and further work on this topic. Factors seem to be at work that aren't measured or controlled but that nonetheless influence defect detection methods' performance. It might be useful to investigate "softer" factors such as motivation and satisfaction[5] and in particular to apply methods in practice, monitor them, and follow up.

Practitioners can use our empirical results in two ways: either as a guideline for a high-level strategy of defect detection or in a full EBSE fashion,[1] finding an answer to a specific question.

In strategy definitions, the summary of the empirical studies can act as a general guide for which defect detection methods to use for different purposes and at different stages of development. Our findings aren't novel or strictly empirically based. However, defining such a strategy would benefit many organizations and projects. Making a defect detection strategy explicit helps communicate values internally, making project members aware of how their work fits into the complete picture. The strategy could also be a starting point for EBSE-based investigations of more specific trade-offs between different methods.

Table 4

Example definition of factor levels

Factor	Example
Artifact	Code modules
Types of defects	Omitted, crucial, and important interfaces
Actor	Novice testers and programmers
Technique	Any feasible technique
Purpose	Design verification
Defect detection activity	Code inspection and unit testing
Evaluation criteria	Effectiveness

To precisely specify which defect detection technique to use, you can apply a full EBSE cycle,[1] along with the variation factors we outlined earlier. Table 4 shows examples of variation factors that are important in searching for a feasible defect detection technique.

These factors help frame the general question into a more precise one: "Which defect detection technique should we use to detect as many critical and important omission defects in interfaces as possible, in code inspection and unit testing, conducted by novice testers and programmers?" Then you can survey the studies in table 2 in detail as a basis for your decision. You might choose a combination of structural and functional testing, for example. Then to anchor the decision, you'd take into account the organization's previous experience. As a final step in the cycle, you must monitor and evaluate the selected technique's performance.

Furthermore, if we feed our industrial evaluations back to the research community, we can increase the body of knowledge about these methods' usefulness.

References

1. T. Dybå, B.A. Kitchenham, and M. Jørgensen, "Evidence-Based Software Engineering for Practitioners," *IEEE Software*, vol. 22, no. 1, 2005, pp. 58–65.
2. V.R. Basili and R. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Trans. Software Eng.*, Dec. 1987, pp. 1278–1296.
3. T. Berling and T. Thelin, "An Industrial Case Study of the Verification and Validation Activities," *Proc. 9th Int'l Software Metrics Symp.*, IEEE CS Press, 2003, pp. 226–238.

About the Authors

Per Runeson is a professor of software engineering at Lund University and leader of the Software Engineering Research Group. He also holds a senior researcher position funded by the Swedish Research Council. His research interests include software development methods and processes, particularly for verification and validation. He received his PhD in software engineering from Lund University. He serves on the editorial boards of *Empirical Software Engineering Journal* and the *Journal of the Association of Software Testing*. Contact him at Dept. of Communication Systems, Lund Univ., Box 118, SE-22100 Lund, Sweden; per.runeson@telecom.lth.se.

Anneliese Andrews is a professor in and chair of the University of Denver's Department of Computer Science. Her research interests include software design, testing, and maintenance as well as quantitative approaches to software engineering data analysis. She received her PhD in computer science from Duke University. She serves on the editorial board of the *Empirical Software Engineering Journal*. Contact her at the Dept. of Computer Science, Univ. of Denver, 2360 S. Gaylord St., John Greene Hall, Rm. 100, Denver, CO 80208; andrews@cs.du.edu.

Carina Andersson is a licentiate in software engineering and a doctoral candidate at Lund University. Her research interests include software development verification and validation processes and software quality metrics and models. She received her MSc in engineering physics with industrial management from Lund University. Contact her at the Dept. of Communication Systems, Lund Univ., Box 118, SE-22100 Lund, Sweden; carina.andersson@telecom.lth.se.

Tomas Berling is a specialist at Ericsson Microwave Systems and a researcher at the IT University of Göteborg. His research and application interests include system verification and validation of complex software systems. He received his PhD in software engineering from Lund University. Contact him at Ericsson Microwave Systems, SE-431 84 Mölndal, Sweden; tomas.berling@ericsson.com.

Thomas Thelin is an associate professor of software engineering at Lund University. His research interests include empirical methods in software engineering; software quality; and verification and validation with emphasis on testing, inspections, and estimation methods. He received his PhD in software engineering from Lund University. Contact him at the Dept. of Communication Systems, Lund Univ., Box 118, SE-22100 Lund, Sweden; thomas.thelin@telecom.lth.se.

4.T. Thelin, P. Runeson, and C. Wohlin, "Prioritized Use Cases as a Vehicle for Software Inspections," *IEEE Software*, vol. 20, no. 4, 2003, pp. 30–33.
5. M. Höst, C. Wohlin, and T. Thelin, "Experimental Context Classification: Incentives and Experience of Subjects," *Proc. 27th Int'l Conf. Software Eng.*, ACM Press, 2005, pp. 470–478.
6. M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Eng.*, vol. 5, no. 3, 2000, pp. 201–214.
7. K. Beck, *Test Driven Development: By Example*, Addison-Wesley, 2002.
8. C. Wohlin et al., *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers, 2000.
9. A. Aurum, H. Petersson, and C. Wohlin, "State-of-the-Art: Software Inspections after 25 Years," *Software Testing, Verification, and Reliability*, vol. 12, no. 3, 2002, pp. 133–154.
10. N. Juristo, A.M. Moreno, and S. Vegas, "Reviewing 25 Years of Testing Technique Experiments," *Empirical Software Eng.*, vol. 9, nos. 1–2, 2004, pp. 7–44.
11. O. Laitenberger, "Studying the Effects of Code Inspection and Structural Testing on Software Quality," *Proc. 9th Int'l Symp. Software Reliability Eng.*, IEEE CS Press, 1998, pp. 237–246.
12. S.S. So et al., "An Empirical Evaluation of Six Methods to Detect Faults in Software," *Software Testing, Verification, and Reliability*, vol. 12, no. 3, 2002, pp. 155–171.
13. P. Runeson and A. Andrews, "Detection or Isolation of Defects? An Experimental Comparison of Unit Testing and Code Inspection," *Proc. 14th Int'l Symp. Software Reliability Eng.*, IEEE CS Press, 2003, pp. 3–13.
14. N. Juristo and S. Vegas, "Functional Testing, Structural Testing, and Code Reading: What Fault Type Do They Each Detect?" *Empirical Methods and Studies in Software Engineering*, R. Conradi and A.I. Wang, eds., Springer, 2003, pp. 208–232.
15. C. Andersson et al., "An Experimental Evaluation of Inspection and Testing for Detection of Design Faults," *Proc. IEEE/ACM Int'l Symp. Empirical Software Eng.*, IEEE CS Press, 2003, pp. 174–184.
16. R. Conradi, A.S. Marjara, and B. Skåtevik, "An Empirical Study of Inspection and Testing Data at Ericsson, Norway," *Proc. 24th NASA Software Eng. Workshop*, NASA, 1999; http://sel.gsfc.nasa.gov/website/sew/1999/topics/marjara_SEW99paper.pdf.
17. W.C. Hetzel, "An Experimental Analysis of Program Verification Methods," doctoral dissertation, Dept. of Computer Science, Univ. of North Carolina, Chapel Hill, 1976.
18. G.J. Myers, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," *Comm. ACM*, Sept. 1978, pp. 760–768.
19. E. Kamsties and C.M. Lott, "An Empirical Evaluation of Three Defect-Detection Techniques," *Proc. 5th European Software Eng. Conf.*, LNCS 989, Springer, 1995, pp. 362–383.
20. M. Roper, M. Wood, and J. Miller, "An Empirical Evaluation of Defect Detection Techniques," *Information and Software Technology*, vol. 39, no. 11, 1997, pp. 763–775.
21. J. Miller, "Applying Meta-Analytical Procedures to Software Engineering Experiments," *J. Systems and Software*, vol. 54, no. 1, 2000, pp. 29–39.
22. A. Rainer, T. Hall, and N. Baddoo, "Persuading Developers to 'Buy into' Software Process Improvement: Local Opinion and Empirical Evidence," *Proc. Int'l Symp. Empirical Software Eng.*, IEEE CS Press, 2003, pp. 326–355.