Examining Software Engineering Beliefs about System Testing Defects

Akito Monden, *Okayama University, Japan*
Masateru Tsunoda, *Kindai University, Japan*
Mike Barker and Kenichi Matsumoto, *Nara Institute of Science and Technology, Japan*

Software engineering beliefs—short, attention-getting, practically useful statements—can help to justify process improvements. The authors empirically validate four selected beliefs in relation to the increase or decrease of defects in system testing.

System testing followed by a product release decision are the last guards in assuring software quality—insufficient testing or the wrong release decision can lead directly to the delivery of low-quality software to users. At the same time, relying too much on system testing to guarantee quality is dangerous because it occurs too late to correct poor-quality software. Moreover, previous studies have shown that bug fixing is much costlier during system testing than in earlier phases.[1] Therefore, we must not only be aware of factors that increase defects but also seek possible process improvements to reduce defects before system testing.

To identify and justify process improvements in individual organizations, where processes, data, and context are varied and unique, we explored using a multivariate modeling technique to analyze past development data collected in organizations. However, unlike some academic approaches, we employed a basic linear regression approach with a limited number of independent variables, each associated with what we call *software engineering (SE) beliefs*. These are short statements that are attention-getting, understandable, and obviously practically useful, such as "about 80 percent of the defects come from 20 percent of the modules," or "peer reviews catch 60 percent of the defects."[2]

SE beliefs are a kind of practical hypothesis that

- are related to early problem detection or possible quality assurance actions;

- have been told elsewhere;
- match IT professionals' intuition in a target organization; and
- can be empirically validated using commonly collected metrics in the target organization.

In particular, we explored four basic SE beliefs related to system testing defects, which we discussed with IT professionals in the target organizations to ensure that they matched professionals' intuition. We focused on system testing defects rather than post-release ones because prerelease information is commonly measured and easily collectable, even in small and medium-sized enterprises (SMEs). In future research, we want to investigate the use of post-release defects for improving post-release software quality even for SMEs.

SE Beliefs

We explore four basic SE beliefs that must be empirically confirmed in the individual context in which process improvement takes place. We started by validating the beliefs in each of two software organizations in a midsize Japanese embedded software company. We then did further analysis to clarify why each SE belief is supported or not supported and identified possible actions for process improvement. Although these four SE beliefs are by no means complete—many other factors are involved with system testing defects—we believe these beliefs are still worth validating in a specific organization where review, test, and reuse processes are relatively stable.

SE Belief 1

Spending more effort on design and code reviews can lower the defect density in system testing. For an individual organization, this SE belief is worth confirming to justify increasing review efforts or conducting additional reviews during a troublesome project. It has been pointed out that the most basic target for process improvement is a software review (or inspection) in the early development phases.[3] We adapted the results of a past study, which showed that higher review efforts increased field software quality, to system testing.[4] Indeed, many software companies focus on early defect detection via design or code reviews for long-term software process improvements.[5,6]

SE Belief 2

Low software quality revealed in design and code reviews will result in high defect density in system testing. This SE belief is worth confirming to discover a troublesome project in an early development phase. It is often the case that a troublesome project yields defects throughout a development lifecycle.[7] Researchers have revealed that in many systems, more defects will be found in modules (or subsystems) that yielded more defects in the past.[8] Thus, very low quality revealed in early development phases could imply high defect density in later phases, including system testing. Note that this belief is a little tricky because you will not find many defects unless you spend enough review effort. Also, if this SE belief is not empirically supported in a target organization, it could imply that the organization is already taking proper quality improvement actions (such as additional design or code reviews) before system testing.

SE Belief 3

Larger quantities of reused code from past projects increase the risk of higher defect density in system testing. Although reusing code from past projects can save coding time and resources, it can also raise the cost and quality risks unless the reused code is well designed, documented, tested, and intended for reuse.[9] Even with systematic reuse, which can increase both productivity and software quality,[10] reused code must be properly tested to decrease quality risks because it is not defect-free in general.[11] However, due to the limitations of testing resources and schedules, companies often spend much less effort on reused code than on developed code, which can increase quality risks from reused code. This SE belief is worth confirming to justify adding more testing efforts for reused code.

SE Belief 4

Higher test case density in unit and integration testing can lower the defect density in system testing. Although it is rather obvious that defects overlooked in unit and integration testing will increase defect density in system testing—a problem that is often referred to as defect slippage[12]—this SE belief explicitly focuses on increasing test case density in unit and integration testing. We believe this belief is worth confirming to show that adding more effort to unit and integration testing will actually help improve the low software quality found in design phases (SE belief 2).

Table 1. Statistics for organizations A and B.

Metrics		Organization A			Organization B		
		Average	Median	Standard deviation	Average	Median	Standard deviation
Development size	A: number of pages of architecture design document (new or modified pages)	295	193	246	220	90	331
	B: number of pages of module design document (new or modified pages)	397	312	351	255	82	442
	C: developed thousand lines of code (lines of new or modified functions)	35.7	32.9	20.4	20.4	11.7	26.7
	D: reused thousand lines of code (lines of unmodified functions)	36.0	19.6	45.8	43.0	19.5	50.5
Review effort	E: architecture design review effort (person hours per page reviewed)	0.25	0.22	0.14	0.48	0.39	0.44
	F: module design review effort (person hours per page reviewed)	0.26	0.22	0.16	0.26	0.22	0.18
	G: code review effort (person hours per developed thousand lines of code)	3.89	3.21	2.63	2.70	2.13	2.35
Test case density	H: unit test case density (defects per thousand lines of code)	99.31	86.00	41.93	60.65	48.17	43.24
	I: integration test case density (defects per thousand lines of code)	39.71	30.34	43.80	39.25	42.40	23.04
	J: system test case density (defects per thousand lines of code)	29.80	20.80	18.09	19.53	16.48	14.35
Defect density	K: defect density in architecture design	0.53	0.51	0.21	0.69	0.40	1.01
	L: defect density in module design	0.48	0.49	0.19	0.40	0.33	0.39
	M: defect density in code review	15.79	11.95	14.23	4.43	3.54	2.81
	N: defect density in unit testing	3.94	3.72	1.63	1.96	1.53	1.63
	O: defect density in integration testing	1.97	1.63	1.65	1.06	1.07	0.68
	P: defect density in system testing	1.71	1.52	1.34	0.73	0.54	0.56

Target Organizations and Projects

We obtained a dataset consisting of data from 107 waterfall-style software development projects (52 in organization A and 55 in organization B) undertaken at a midsize software development company from 2009 to 2012. The main business domain of both organizations is embedded software development for wired and wireless communication systems, image processing systems, and public transportation systems. However, the two organizations are separate from each other and have different customers. Most projects are contract-based development to produce software based on requirements given by other companies. Hence, most projects consisted of development phases after requirements analysis—that is, architectural design, module design, implementation, unit testing, integration testing, and system testing. Here, system testing does not include hardware testing.

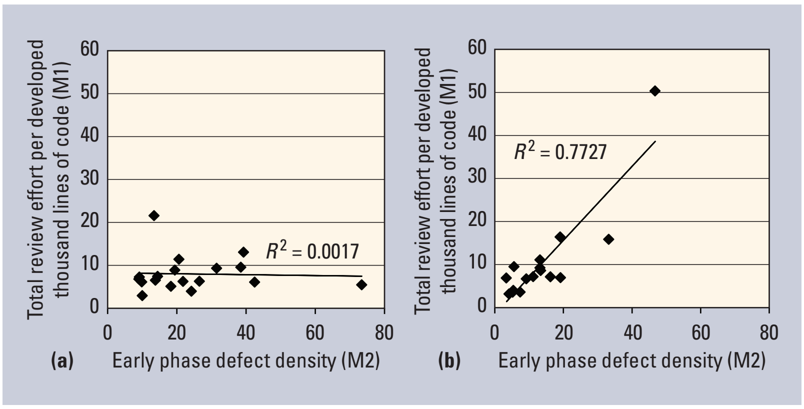

Figure 1. Histograms of defect density in system testing. (a) Organization A has higher defect density projects than (b) organization b, implying that there is more room for process improvement in A.

After basic cleaning of the dataset, such as deleting tiny projects or those with missing values, 34 projects (18 in organization A and 16 in organization B) remained available. As Table 1 shows, statistics for these projects were measured in terms of development size (document pages or noncomment lines of C/C++ source code), review efforts (person hours), test case density, and defect density in various stages of development. Based on these statistics, we could identify several differences between the two organizations.

Regarding development size, projects in organization A were about 2 to 4 times larger than those in organization B in terms of the median values of document pages and developed lines of code, whereas the reused size was almost the same. Regarding the quality assurance effort (that is, review effort and test case density), projects in A had relatively smaller values in architecture design, whereas they had larger values in later phases (code review, unit testing, integration, and system testing). Regarding defect density, projects in A yielded more defects than those in B. This implies that organization A is struggling more with quality assurance than B. Figure 1 shows histograms of the defect density in acceptance testing. Obviously, organization A has higher defect density projects than organization B, which implies that there is more room for process improvement in A.

Initial Validation of SE Beliefs

Table 2 shows the four metrics M1 to M4 used for validating the four SE beliefs. As shown in the table, each metric is associated with the hypothesis for an SE belief. For example, SE belief 1 hypothesizes that projects having a higher M1 value (total review effort per thousand lines of code) will have a lower defect density in system testing. Note that M1 does not include reused code—that is, the denominator is not (C + D)—because the design and code reviews have been done on the new or modified pages or code only (not on the unmodified pages or code).

To validate the four SE beliefs, we employed all four metrics M1 to M4 as independent variables with the defect density as a dependent variable in a multivariate linear regression analysis (Equation 1). Because the metrics are expected to independently increase or decrease the defect density, it is convenient to use linear regression analysis for the validation. By identifying variables significantly related to the defect density of system testing based on the t-test of the coefficients' significance, we can statistically validate SE beliefs:

$$\hat{Y} = k_1M_1 + k_2M_2 + k_3M_3 + k_4M_4 + C, \qquad (1)$$

where $\hat{Y}$ is the estimated defect density, M_i is an independent variable, k_i is the regression coefficient, and C is a constant.

Table 3 shows the results of the regression analysis for organizations A and B. For each independent variable, Table 3 shows the regression coefficient and the p-value of its t-test, which is the estimated probability of rejecting the null hypothesis "a coefficient is zero." The bold, italic p-values indicate that the coefficient is statistically significant ($p < 0.05$), which supports the validity of its associated SE belief.

Analysis beyond the SE Beliefs

As shown in Table 3, M1 was not significant in either organization, whereas M2 was statistically significant only in organization A. This indicates that coping with high defect density in design and code reviews is crucial for organization A, because it currently has a high level of system testing defects in such cases. On the other hand, for organization B, the results suggest that even if a high defect density was found in design and code review, it has a low level of system testing defects. For further analysis, we analyzed the relationship between M2 (early phase defect

Table 2. Metrics related to software engineering (SE) beliefs.

SE belief	Metrics	Definition	Hypothesis in system testing
1	M1: total review effort per developed thousand lines of code	Total review effort/C	Higher M1 has a lower defect density
2	M2: early phase defect density	K + L + M	Higher M2 has a higher defect density
3	M3: reuse ratio	D/(C + D)	Higher M3 has a higher defect density
4	M4: average test case density of unit testing and integration testing	(H + I)/2	Higher M4 has a lower defect density

Table 3. Results of regression analysis.

	Organization A		Organization B	
Independent variable	Regression coefficient	*p*-value	Regression coefficient	*p*-value
M1	0.0226	0.768	–0.0217	0.355
M2	0.0394	***0.029***	0.0031	0.885
M3	2.0649	0.086	1.0512	***0.006***
M4	–0.0037	0.647	–0.0006	0.893
(Constant)	0.0906	0.914	0.3450	0.309

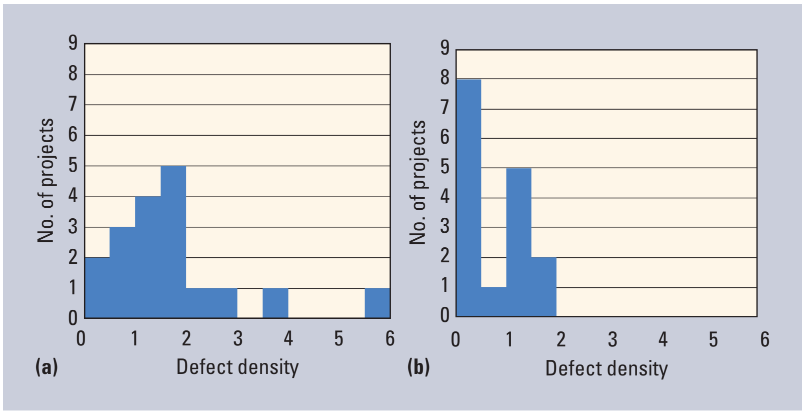

Figure 2. Review effort analysis for (a) organization A and (b) organization B. We analyzed the relationship between M2 (early phase defect density) and M1 (total review effort per developed thousand lines of code).

density) and M1 (total review effort per developed thousand lines of code), shown in Figure 2. Obviously, organization B spent more review effort (indicated by M1) on high defect density projects (indicated by M2), which demonstrates why organization B has a lower level of system testing defects for such troublesome projects. On the other hand, even if organization A found a high defect density in early phases, it did not spend additional review effort, which demonstrates that this is a necessary target for process improvement in organization A. This result also demonstrates that using the same baseline of review effort per size for all projects is a bad habit in quality assurance.

Regarding M3, in organization B, it was statistically significant, which means that SE belief 3—that is, more reused code has a higher level of defect density—was confirmed in organization B. In further analysis, to estimate how many defects are introduced by 1,000 lines of reused code, we conducted an additional regression analysis using the number of defects found in system testing as a dependent variable, and obtained a model $\hat{Y} = 0.233 \cdot$ (developed thousand lines of code) + $0.085 \cdot$ (reused thousand lines of code). Note that $\hat{Y}$ is the number of defects, not the density, and that we confirmed beforehand that these two variables were really independent. The correlation coefficient between developed thousand lines of code and reused thousand lines of code was –0.004. Based on this analysis, it can be estimated that 1,000 lines of reused code introduces 0.085 defects in system testing, whereas 1,000 lines of developed code introduces 0.233 defects. This indicates that about 27 percent of the total defects came from reused code. This number, 27 percent, is larger than the 16 percent in system and acceptance testing found in a previous study.[11] This indicates that reducing reuse-related defects is a crucial task for process improvement in organization B.

Looking back at Table 3, we see that M4 was not significant in organization B, which implies that unit and integration testing are currently not enough to eliminate the risk of reused code. To understand the current integration testing strategy of organization B, we analyzed the relationship between the number of test cases and developed or reused code sizes (Figure 3). Obviously, the number of test cases is proportional to the developed thousand lines of code, and not at all related to the reused thousand lines of code. This demonstrates the need for additional test cases on reused code in integration testing in organization B.

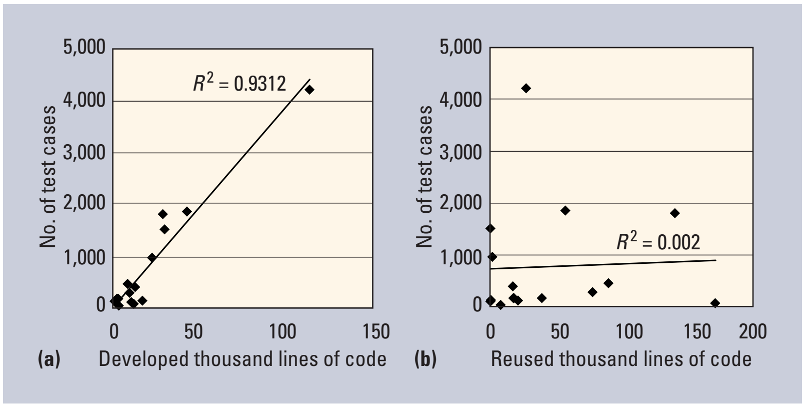

Figure 3. Analysis of integration testing for organization B. We analyzed the relationship between the number of test cases and (a) developed or (b) reused code sizes.

Our study confirmed that focusing on a small number of SE beliefs that match IT professionals' intuition in a target organization is a good starting point for such an analysis. Regardless of whether the SE beliefs are confirmed, we can then proceed to a further analysis on why they were confirmed or not, and identify possible process improvements.

This approach provides a bridge between SE beliefs and the practical need of development organizations to identify targets for process improvements that are suited to the individual organization. Based on our study, even SMEs can use this approach to improve their processes, which will result in better products. By using available metrics and linear regression analysis to confirm whether these SE beliefs apply in an individual organization, then further analyzing data related to SE beliefs with statistically significant results, we can provide recommendations for tailored process improvements that are attention-getting, easily understandable, and practically useful.

The main limitation of this approach is that it cannot improve system testing itself, given that we lack defect information after system testing (that is, the post-release defects). We recommend using post-release defects and related SE beliefs in cases where an organization has enough data on post-release defects. Also, a larger dataset is obviously preferable to provide more confidence in and better drive process improvements.

What does this mean for IT professionals? Instead of just claiming that SE beliefs seem reasonable, using this basic set of metrics and analysis allows IT professionals to check whether these beliefs really work for them in their organizations. From those results, IT professionals can then develop their own set of recommendations tailored to their organization. These aren't just SE beliefs that are generally true—they are ones that have been tested and proven in your organization.

References

1. T. Matsumura et al., "Analyzing Factors of Defect Correction Effort in a Multi-Vendor Information System Development," *J. Computer Information Systems*, vol. XLIX, no. 1, 2008, pp. 73–80.
2. B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," *Computer*, Jan. 2001, pp. 135–137.
3. L. Harjumaa, I. Tervonen, and P. Vuorio, "Using Software Inspection as a Catalyst for SPI in a Small Company," *Proc. 5th Int'l Conf. Product Focused Software Process Improvement*, LNCS 3009, 2004, pp. 62–75.
4. Y. Takagi et al., "Analysis of Review's Effectiveness Based on Software Metrics," *Proc. 6th Int'l Symp. Software Reliability Eng.* (ISSRE), 1995, pp. 34–39.
5. O. Mizuno and T. Kikuno, "Empirical Evaluation of Review Process Improvement Activities with Respect to Post-Release Failure," *Proc. Empirical Studies on Software Development Eng.*, 1999, pp. 50–53.
6. N. Honda and S. Yamada, "Empirical Analysis for High Quality Software Development," *Am. J. Operations Research*, vol. 2, no. 1, 2012, pp. 36–42.

7. E. Yourdon, *Death March*, 2nd ed., Prentice Hall, 2003.
8. M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating Defect Prediction Approaches: A Benchmark and an Extensive Comparison," *Empirical Software Eng.*, vol. 17, nos. 4–5, 2012, pp. 531–577.
9. W. Tracz, "Software Reuse: Motivators and Inhibitors," *Proc. 32nd IEEE CS Int'l Conf.* (COMPCON), 1987, pp. 358–363.
10. P. Mohagheghi and R. Conradi, "Quality, Productivity and Economic Benefits of Software Reuse: A Review of Industrial Studies," *Empirical Software Eng.*, vol. 12, no. 5, 2007, pp. 471–516.
11. W.M. Thomas, A. Delis and V.R. Basili, "An Analysis of Errors in a Reuse-Oriented Development Environment," *J. Systems and Software*, vol. 38, 1997, pp. 211–224.
12. V. Basili et al., "Bridging the Gap between Business Strategy and Software Development," *Proc. Int'l Conf. Information Systems* (ICIS), 2007, article no. 25.

***Akito Monden** is a professor in the Graduate School of Natural Science and Technology at Okayama University, Japan. His research interests include software measurement and analytics, and software security and protection. Monden received a Doctor of Engineering in information science from Nara Institute of Science and Technology. He is a member of IEEE, ACM, the Institute of Electronics, Information, and Communication Engineers, the Information Processing Society of Japan, and the Japan Society for Software Science and Technology. Contact him at monden@okayama-u.ac.jp.*

***Masateru Tsunoda** is a lecturer in the Department of Informatics at Kindai University, Japan. His research interests include software measurement and human factors in software development. Tsunoda received a Doctor of Engineering in information science from the Nara Institute of Science and Technology. He is a member of IEEE, the Institute of Electronics, Information, and Communication Engineers, the Information Processing Society of Japan, the Japan Society for Software Science and Technology, and the Japan Society for Information and Systems in Education. Contact him at tsunoda@info.kindai.ac.jp.*

***Mike Barker** is a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include software measurement, research methods, and the software development process. Barker has almost 40 years of software engineering experience in both industry—with companies such as RCA and BBN—and academia, at MIT. He has been the steering committee chair for the Conference on Software Engineering Education and Training since 2012, is a long-time member of ACM and IEEE, and is a PMP-certified member of the Project Management Institute. Contact him at mbarker@mit.edu.*

***Kenichi Matsumoto** is a professor in the Graduate School of Information Science at Nara Institute of Science and Technology, Japan. His research interests include software measurement and software process. Matsumoto received a PhD in information and computer sciences from Osaka University, Japan. He is a fellow of the Institute of Electronics, Information, and Communication Engineers and the Information Processing Society of Japan, a senior member of IEEE, and a member of ACM and the Japan Society for Software Science and Technology. Contact him at matumoto@is.naist.jp.*