Probing Software Engineering Beliefs about System Testing Defects: Analyzing Data for Future Directions

[a, c]Akito Monden [b, c]Masateru Tsunoda [c]Mike Barker [c]Kenichi Matsumoto
[a]Graduate School of Natural Science and Technology, Okayama University
monden@okayama-u.ac.jp
[b]Department of Informatics, Faculty of Science and Engineering, Kindai University
tsunoda@info.kindai.ac.jp
[c]Graduate School of Information Science, Nara Institute of Science and Technology
mbarker@is.naist.jp, matumoto@is.naist.jp

Research findings are often expressed as short startling sentences or software engineering (SE) beliefs such as "about 80 percent of the defects come from 20 percent of the modules" and "peer reviews catch 60 percent of the defects" [2]. Such SE beliefs are particularly important in industry, as they are attention-getting, easily understandable, and thus practically useful. In this paper we examine the power of such SE beliefs to justify process improvement through empirical validation of selected beliefs related to the increase or decrease of defects in system testing. We explore four basic SE beliefs in data from two midsize embedded software development organizations in Japan, and based on this information, identify possible process improvement actions for each organization. Based on our study, even small and medium-sized enterprises (SMEs) can use this approach to find possible directions to improve their process, which will result in better products.

Keywords
D.2.19 Software Quality/SQA, D.2.8.c Process metrics

1. INTRODUCTION

System testing followed by a product release decision are the last guards to assure software quality, since insufficient testing and/or the wrong release decision can directly cause delivery of low quality software to users. At the same time, relying too much on system testing to guarantee quality is dangerous because it is too late to correct bad software quality when it is discovered in system testing. Also, previous studies have shown that bug fixing costs in system testing are much larger than that in earlier phases [6]. Therefore, it is necessary not only to be aware of factors that increase defects but also to seek possible process improvement actions to reduce defects before system testing.

To identify and justify process improvement actions in an individual organization where the process, data, and context are different and unique, we explored using a multivariate modeling technique to analyze past development data collected in the organization. However, unlike some academic approaches, we employed a basic linear regression approach with a limited number of independent variables each associated with what we have called software engineering (SE) beliefs. These are short statements that are attention-getting, understandable, and obviously practically useful. They are a kind of practical hypothesis that (1) are related to early problem detection or possible quality assurance actions, (2) have been told elsewhere, (3) meet IT professionals' intuition in a target organization, and (4) can be empirically validated using commonly collected metrics in the target organization. In particular, we explored the following four basic SE beliefs related to system testing defects, which we have discussed with IT professionals in the target organizations to make sure these match their intuition.

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 the review effort or conducting additional reviews in a troublesome project. It has been pointed out that the most basic target for process improvement is a software review (or inspection) in early development phases [4]. We adjust the result of a past study, which showed higher review effort increased field software quality, to system testing [9]. Indeed, many software companies focus on early defect detection by design/code reviews for long-term sotware process improvements [4][7].

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 [12]. Researchers have revealed that in many systems, more defects will be found in modules (or subsystems) that yielded more defects in the past [3]. Thus, very low quality revealed in early development phases may 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, this implies that the organization is already taking proper quality improvement actions (such as additional design/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.

While 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 [11]. Even in systematic reuse, which can increase both productivity and software quality [8] reused code must be properly tested to decrease the quality risks because reused code is not defect-free in general [10]. However, due to the limitation of testing resources and schedule, companies often spent much less effort on reused code than on developed code, which can raise quality risks from reused code.

This SE belief is worth confirming to justify adding more test effort for reused code.

SE Belief 4: Higher test case density in unit and integration testing can lower the defect density in system testing.

While it is rather obvious that defects overlooked in unit and integration testing will increase defect density in system testing, a problem which is often referred to as defect slippage [1], this SE belief explicitly focuses on increasing test case density in unit and integration testing. We believe this SE 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).

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

The reason why we focus on the system testing defects and not on the post release ones is that pre-release 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.

2. TARGET ORGANIZATIONS AND PROJECTS

We obtained a data set consisting of data from 107 waterfall-style software development projects (52 in organization A, and 55 in organization B) conducted in a midsize software development company from 2009 to 2012. The main business domain of both organizations is embedded software development for wired/wireless communication systems, image processing systems, and public transportation systems. However, the two organizations are separated 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 consist of development phases after requirements analysis, i.e. architectural design, module design, implementation, unit testing, integration testing and system testing. Here, system testing does not include hardware testing.

After basic cleaning of the data set, such as deleting projects of tiny size or having missing values, 34 projects (18 in organization A, and 16 in organization B) remained available. As shown in Table 1, statistics for these projects were measured in terms of development size (document pages or non-comment 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, while the reused size is almost the same. Regarding the quality assurance effort (i.e. review effort and test case density,) projects in A have relatively smaller values in architecture

Table 1. Statistics for Organizations A and B

Metrics		Organization A			Organization B		
		Average	Median	Std.Dev.	Average	Median	Std.Dev.
Development size	(A) # of pages of archtecture design document (new or modified pages)	295	193	246	220	90	331
	(B) # 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) Archtecture 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 archtecture 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

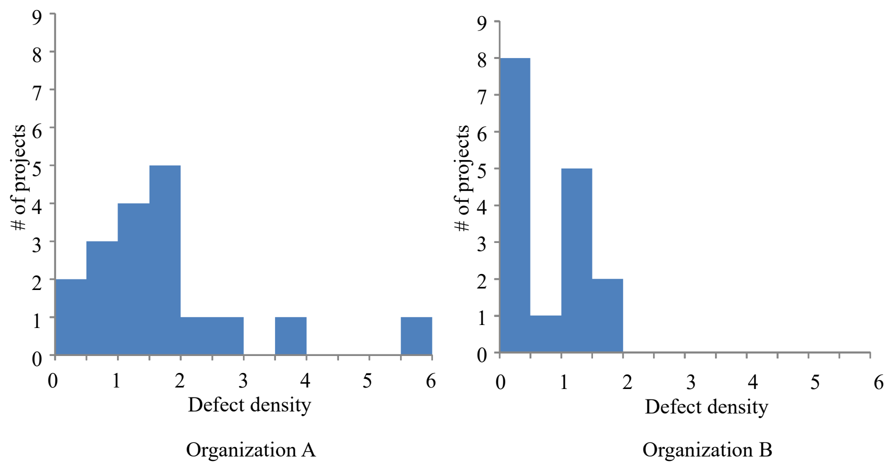


Figure 1. Histograms of defect density in system testing.

Table 2. Metrics related to 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

design while they have larger values in later phases (code review, unit testing, integration and system testing). Regarding defect density, projects in A yield more defects than those in B. These imply 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 there is more room for process improvement in A.

3. INITIAL VALIDATION OF SE BELIEFS

Table 2 shows the four metrics M1 to M4 used for validating the four SE beliefs. As shown in Table 2, each metric is associated with the hypothesis for a 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, i.e., the denominator is not (C+D), because the design/code reviews have been done on the new or modified pages/code only (not on the un-modified pages/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)). Since 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.

$$\hat{Y} = k_1 M_1 + k_2 M_2 + k_3 M_3 + k_4 M_4 + C \quad \text{....} \ (1)$$

$\hat{Y}$: Estimated defect density

M_i : Independent variable

k_i : Regression coefficient

C : Constant

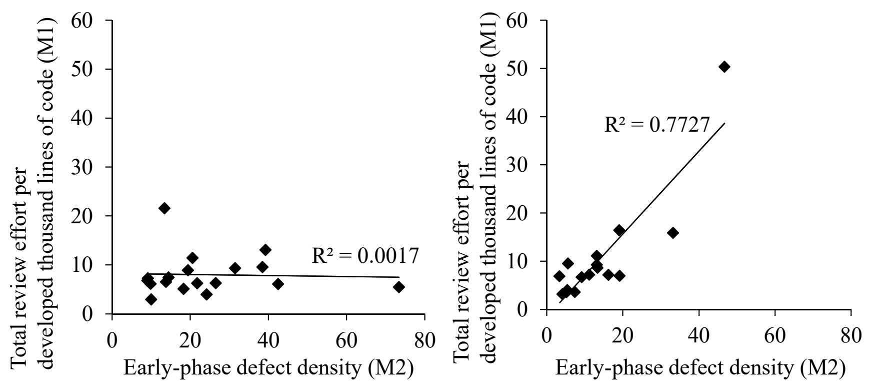
Table 3 shows the results of the regression analysis for organizations A and B. For each independent variable, Table 3 shows the regression coefficient and the p-value of its *t*-test, which is the estimated probability of rejecting the null hypothesis "a coefficient is zero." The bold italic p-values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

4. FURTHER ANALYSIS BEYOND THE SE BELIEFS

As shown in Table 3, M1 was not significant in either organization, while 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, as they currently have a high level of system testing defects in such a case. On the other hand, for organization B, the results suggest that even if a high defect density was found in design and code review, they have a low level of system testing defects. For further analysis, we analyzed the relationship between M2 (early-phase defect 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, they did not spend additional review effort,

Table 3. Result 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



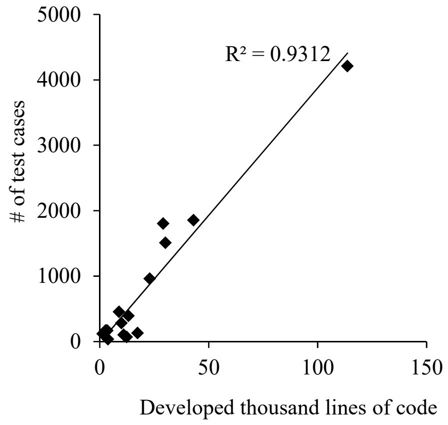
Organization A Organization B

Figure 2. Analysis on 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 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 1000 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 1000 lines of reused code introduces 0.085 defects in system testing, while 1000 lines of developed code introduces 0.233 defects. This indicates that about 27% of the total defects came from reused code. This number, 27%, is larger than the 16% in system and acceptance testing found in a previous study [10]. This indicates that reducing the 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/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.



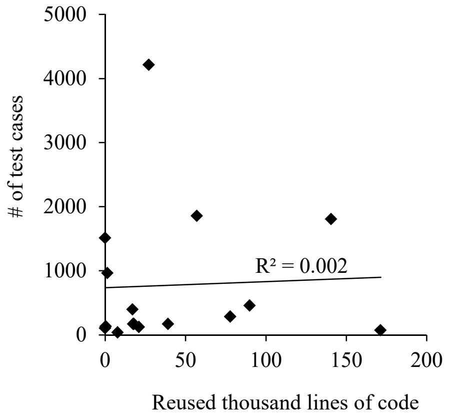


Figure 3. Analysis of integration testing for organization B.

5. SUMMARY

This article explored the use of an SE belief-based defect factor analysis to identify possible quality improvement actions in an individual organization where the process, data, and context are different and unique. We confirmed that focusing on a small number of SE beliefs, which meet IT professionals' intuitions in a target organization, is a good starting point for such an analysis. Whether or not the SE beliefs are confirmed, we can then proceed to a further analysis on why they are confirmed or not confirmed, and to identify possible process improvement actions.

This approach provides a bridge between SE beliefs and the practical need of development organizations to identify targets for process improvements suited to individual organizations. Based on our study, even SMEs can use this approach to improve their process, 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, and then further analyzing data related to SE beliefs with statistically significant results, we can provide recommendations for tailored process improvement actions for individual organizations that are attention-getting, easily understandable, and practically useful.

The main limitation of this approach is that, it cannot improve system testing itself, as we lack defect information after system testing (i.e. the post release defects.) We recommend using post-release defects and related SE beliefs in case an organization could have enough data on post release defects. Also, a larger data set is obviously preferable to give more confidence to drive the process improvement actions.

What does that 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 individual 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!

6. REFERENCES

[1] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, C. Seaman, and A. Trendowicz, "Bridging the Gap between Business Strategy and Software Development," Proc. International Conference on Information Systems (ICIS2007), No. 25, 2007.

[2] B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," Computer, pp.135-137, Jan. 2001.

[3] M. D'Ambros, M. Lanza, R. Robbes, "Evaluating Defect Prediction Approaches: a Benchmark and an Extensive Comparison," Empirical Software Engineering, Vol.17, Issue 4-5, pp.531-577, 2012.

[4] L. Harjumaa, I. Tervonen, P. Vuorio, "Using Software Inspection as a Catalyst for SPI in a Small Company," 5th International Conference on Product Focused Software Process Improvement (Profes2004), Lecture Notes in Computer Science, pp.62-75, Springer, 2004.

[5] N. Honda and S. Yamada, “Empirical analysis for high quality sotware development,” American Journal of Operations Research, Vol.2, No.1, pp.36-42, 2012.

[6] T. Matsumura, A. Monden, S. Morisaki, and K. Matsumoto, “Analyzing factors of defect correction effort in a multi-vendor information system development,” Journal of Computer Information Systems, Vol.XLIX, No.1, pp.73-80 2008.

[7] O. Mizuno and T. Kikuno, “Empirical evaluation of review process improvement activities with respect to post-release failure,” Proc. Empirical Studies on Software Development Engineering, pp.50-53, 1999.

[8] P. Mohagheghi and R. Conradi, “Quality, productivity and economic benefits of software reuse: a review of industrial studies,” Empirical Software Engineering, Vol.12, Issue 5, pp.471-516, 2007.

[9] Y. Takagi, T. Tanaka, N. Niihara, K. Sakamoto, S. Kusumoto, T. Kikuno, “Analysis of review's effectiveness based on software metrics,” Proc. 6th International Symposium on Software Reliability Engineering (ISSRE’95), pp.34-39, 1995.

[10] W. M. Thomas, A. Delis and V. R. Basili, “An analysis of errors in a reuse-oriented development environment,” Journal of Systems and Software, Vol.38, pp.211-224, 1997.

[11] W. Tracz, “Software reuse: motivators and inhibitors,” Proc 32nd IEEE Computer Society International Conference (COMPCON’87), pp.358-363, 1987.

[12] E. Yourdon, “Death march (2nd Edition),” Prentice Hall, Nov. 2003.