Myths in Software Engineering: From the Other Side

Nachiappan Nagappan

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
nachin@microsoft.com

Abstract. An important component of Empirical Software Engineering (ESE) research involves the measurement, observation, analysis and understanding of software engineering in practice. Results analyzed without understanding the contexts in which they were obtained can lead to wrong and potentially harmful interpretation. There exist several myths in software engineering, most of which have been accepted for years as being conventional wisdom without having been questioned. In this talk we will deal briefly with a few popular myths in software engineering ranging from testing and static analysis to distributed development and highlight the importance of context and generalization.

Keywords: Empirical Software Engineering, Code coverage, Failures, People, development teams, Software Inspection, Distributed development, Software assertions.

1 Introduction

The ISERN[1] community (a group made up of different research labs, universities and industrial partners) states its purpose as, *"software engineering is a relatively new and immature discipline. In order to mature, we need to adopt an experimental view of research allowing us to observe and experiment with technologies, understand their weaknesses and strengths, tailor technologies to the goals and characteristics of particular projects, and package them together with empirically gained experience to enhance their reuse potential in future projects"*.

In empirical software engineering it is important to contextualize the environment in which the results are obtained to generalize results across studies. In general practitioners become more confident in a theory when similar findings emerge in different contexts [1]. Towards this end, empirical software engineering presents special emphasis on replication of experiments to learn and generalize results across different contexts, environments and domains.

Empirical Software engineering also focuses on evaluating oft held opinions in software engineering as some of these opinions are purely opinions and are not backed up by scientific data. In this keynote, I would like to shed some light on interesting case studies from industry that show some myths to be false, explain the importance of context and show the utility of building empirical bodies of knowledge.

[1] http://isern.iese.de/network/ISERN/pub/isern.manifesto.html

2 Case Studies

Some of the myths I plan to address in this talk with appropriate citations for further reading are:

Code coverage: Does higher code coverage mean better quality. Can we stop testing when we reach 100% code coverage? Are there better measures to quantify test efficacy? Should the goal be to maximize coverage? [2]

Static analysis: Static analysis tools are easy to use and low cost to deploy in software development teams. The downside being static analysis bug finding tools produce a lot of false positives. Are there better techniques for fault fix prioritization? When should we stop fixing static analysis bugs? What is the cost-benefit of utilizing static analysis defects? [3]

Unit testing: Is it really beneficial to do unit testing? Is there empirical evidence to show that teams adopting unit testing product better quality products? [6]

Test Driven Development: Test Driven Development (TDD) is a technique proposed as part of the Agile software development practices like Extreme Programming (XP). Does TDD produce better quality code? What is the cost-benefit trade-off of doing TDD? [7]

Inspections: Inspections have been widely practiced in industry. An important fact associated with inspections is that they are only as good as the inspector involved in the process. How does the human element influence the result of inspections? [8]

Assertions: Software assertions have often been discussed as an important best practice in software development. Has there been any empirical evidence of using assertions in software development that improved the efficacy of the project? [9]

Distributed development: We are increasingly observing large commercial projects being built by teams distributed across the world. We see professionals on either side of the spectrum who strongly argue for or against distributed software development. Does distributed software development affect quality? Do distributed teams work on simpler tasks and involve more communication overhead? [4]

Minor contributors: Software development is a task that involves a variety of individuals working together. Do certain individuals have a higher likelihood of causing failures/problems in the system than others?

Socio-technical networks: On a related note, software development is an activity that involves teams of developers coordinating with each other. In this regard there are multiple dependency structures which exist (due to the code dependencies/people dependencies etc). Does the intersection of these dependencies predict failures? [5]

Organizational metrics: Conway's law (organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations [10]) has been used often in software organizations to organize teams. How important is Conway's law and what are its implications on software quality? [11]

References

1. Basili, V.R., Shull, F., Lanubile, F.: Building Knowledge Through Families of Experiments. IEEE Transactions on Software Engineering 25(4), 456–473 (1999)
2. Mockus, A., Nagappan, N., Dinh-Trong, T.T.: Test Coverage and Post-verification Defects: A Multiple Case Study. In: Empirical Software Engineering and Measurement (ESEM), Orlando, FL (2010)
3. Nagappan, N., Ball, T.: Static Analysis Tools as Early indicators of Pre-Release Defect Density. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, Springer, Heidelberg (2006)
4. Bird, C., et al.: Does Distributed Development affect Software Quality? An empirical Case Study of Windows Vista. In: International Conference on Software Engineering (ICSE), Vancouver, Canada (2009)
5. Bird, C., et al.: Putting It All Together: Using Socio-technical Networks to Predict Failures. In: International Symposium on Software Reliability Engineering (ISSRE), Mysore, India (2009)
6. Williams, L., Kudrjavets, G., Nagappan, N.: On the Effectiveness of Unit Test Automation at Microsoft. In: International Symposium on Software Reliability Engineering (ISSRE), Mysore, India (2010)
7. Nagappan, N., et al.: Realizing Quality Improvement Through Test Driven Development: Results and Experiences of Four Industrial Teams. Empirical Software Engineering (ESE) 13(3), 289–302 (2008)
8. Carver, J.C., Nagappan, N., Page, A.: The Impact of Educational Background on the Effectiveness of Requirements Inspections: An Empirical Study. IEEE Transactions in Software Engineering 34(6), 800–812 (2008)
9. Kudrjavets, G., Nagappan, N., Ball, T.: Assessing the Relationship between Software Assertions and Faults: An Empirical Investigation. In: International Symposium on Soft-ware Reliability Engineering (ISSRE), Raleigh, NC (2006)
10. Conway, M.E.: How Do Committees Invent? Datamation 14(4), 28–31 (1968)
11. Nagappan, N., Murphy, B., Basili, V.: The Influence of Organizational Structure on Software Quality: An Empirical Case Study. In: International Conference on Software Engineering, Leipzig, Germany (2008)