Empir Software Eng (2012) 17:18–22
DOI 10.1007/s10664-011-9184-6

The difficulties of building generic reliability models for software

Brendan Murphy

Published online: 12 November 2011

Editors: Martin Shepperd and Tim Menzies

Abstract The Software Engineering research community have spent considerable effort in developing models to predict the behaviour of software. A number of these models have been derived based on the pre and post behaviour of the development of software products, but when these models are applied to other products, the results are often disappointing. This appears to differentiate Software from other engineering disciplines that often depend on generic predictive models to verify the correctness of their products. This short paper discusses why other engineering disciplines have managed to create generalized models, the challenges faced by the Software industry to build these models, and the change we have made to our process in Microsoft to address some of these challenges.

Keywords Software Engineering · Models · Reliability · Risk · Development

1 Introduction

The reliability of a software product is important for a number of reasons such as customer satisfaction, the cost of servicing the product following its release, etc. It is therefore important for software developers to predict the reliability of the finished product as early as possible during its development. Other engineering disciplines, have built models which use metrics collected during the products development to predict the reliability of the finished product. These models can also be used to change the development process to improve the reliability of the released product.

The computer industry has attempt to replicate other engineering disciplines through building models to predict the reliability of a software product based on the metrics captured during the development process. These models have been derived based on the pre- and post- release behaviour of individual products (such as Windows Vista) following its release. But, when these models are applied to other products (such as Windows 2008)

B. Murphy (✉)
Microsoft Research, 7 J J Thomson Avenue, Cambridge CB3 0FB, UK
e-mail: bmurphy@microsoft.com

they often struggle to achieve the same level of accuracy (Kitchenham et al. 2007; Zimmerman et al. 2009). This differs from most other engineering disciplines where model repeatability is fundamental to their verification of products.

Over the past 8 years, in combination with, researchers at Microsoft, I have analyzed the behavior of Windows software products. During this time we have generated a number of risk models based on multiple development metrics. Over the last year we have expanded the work to monitor products from a number of different Microsoft organizations and, like other researchers in other studies, have been unsuccessful in generalize the Windows models for these other products.

This article discusses why other engineering disciplines have managed to create generalized models, the challenges faced by the software industry to build these models, and the changes we have made to our process in Microsoft to address some of these challenges.

2 Generalized Models in Other Engineering Disciplines

In other engineering industries, such as the railways, models are used to verify changes to the development process during the product's design and implementation (Chen et al. 1999). This is possible because the models are generic to that industry and can be applied to defined classes of product. These models have been built over many years. They are the result of decades and sometimes hundreds of years of monitoring the successes and more importantly the failures of their industry. This monitoring process is often formalized through the creation of independent safety bodies that have responsibility for determining the root cause analysis of failures. Their failure analysis involves considering all aspects of the causes of failures, both technical and human. These safety bodies are powerful and their recommendations often have to be implemented by their industry.

To understand the ability of these safety bodies to consider the total system, and also to understand their power, it is best to look at an example from the railways industry in 1840 (Rolt 1998). At that time a number of accidents occurred due to managing the train systems based on time through the 'time interval system'. In 1840 a board of train inspectors identified that the root cause of the accident was that time is measured locally, whereby if the time in London was 12 pm it would be 11:35 am in Bristol. The board realized that the problems where not technical, rather human related. They realized the average person would struggle to manage trains when time was so variable, so they made the decision to change time and for the whole railway system to run on London time (hence why we now have centralized time systems the world over). Other changes introduced to address this issue where not always perfect but they continually learned from their mistakes. Over the last 170 years various safety bodies have improved the safety of railway travel by continually enforcing stricter regulations on its development, management, and maintenance.

These restrictions inevitably have meant that these industries do not have the flexibility to try out new ideas. All changes needs to be verified through a rigorous system (including models) which compares the changes against the current system. But these models are only accurate if the products are designed and implemented following a pre-defined set of rules. Making significant changes to the design process will invalidate these models which significantly increases the cost of verifying the product. The few deaths that occur on trains and planes validate this as an excellent process; the issue is whether it is applicable to the software industry.

3 Making Software Fault Tolerant

The software industry has been making fault tolerant products for decades Bartlett et al. which are used to manage safety critical products such as airplanes. To achieve a safety critical product it is necessary to place significant restrictions on both the hardware the software that runs on it (its fault model must be fully understood) and the interfaces to the software. Where the software is not embedded the people installing and using the software must be fully certified. The processes used to develop fault tolerant software are inflexible and strictly controlled but do allow for predictive models to be generalized to all other software following this process.

So it is possible for the software industry to produce fault tolerant software using repeatable development processes but the reason why these processes are only applied in niche markets is the cost of applying these methods, both in terms of time to market, headcount and the inflexibility it imposes on the end product.

The software industry knows how to generalize predictive models, by applying the same techniques as all other industries, specifically to freeze and strictly control the product requirements, its development process and ensure its users are highly trained. So how can solve the problem of generalizing predictive models when the users are not trained, the requirements are not set in stone and the development process evolves over time.

4 Experiences of Re-Applying Predictive Models

In Microsoft, I and a number of other researchers have spent over 8 years analyzing the relationship between software development and its subsequent behavior on customer sites. During this time we focused on Windows, analyzing all of its products since the release of Windows 2000 (Levidow and Murphy 2000). For Windows we found that predictive models can be re-applied to different products as long as the products are of similar type (Nagappan et al. 2008). Specifically models developed from Windows Vista can be applied to Windows 7, but Windows Vista models cannot be applied to Windows 2008 server release or any Service Pack releases. We did not overly investigate the reasons for this lack of generalization, assuming it was due to differences in product characteristics.

Recently we expanded our monitoring process to capture the development metrics from a number of other organizations. The majority of these organizations used the same set of development tools as Windows, but their usage of these tools varied greatly. To verify that we had interpreted the data correctly we attempted to characterize the product behavior using the same analytics techniques as applied to Windows. Initially, this was very unsuccessful as the results did not match the product group quality metrics or their perceptions of the behavior of the product developments.

To resolve these differences we worked with the different product groups to understand the discrepancies between our metrics and their own. In most cases, the differences were due to our processes misinterpreting the software metrics, due to differences in the way the individual product group developed software when compared to the Windows group. A number of these differences reflected the changes in the product characteristics. The short term solution was to fork our data collection and analysis process for each of the product groups. The processes used by the product groups allowed us to obtain historical information regarding previously released products which we used to develop predictive models for their current products. This resulted in a number of product specific risk models.

This short term solution of forking the current process provided unique knowledge of the differences in the development methods across diverse product groups. We determined that we needed to merge these collection and analysis processes together, but to do that we needed to understand why the same data collection and analysis techniques cannot be applied across different products.

5 Normalizing Metrics Across Different Products

Through analysis of the data from multiple products across multiple organizations it became obvious that the collected metrics were often impacted by domain specific issues. Two examples of this are

1. Differentiating between common and functional code
   The majority of the monitored products consisted of multiple binaries and these binaries consisted of a combination of common code that existed in all binaries (e.g. libraries) and code unique to the individual binary. The objective, of the development of the common code, is independent of the functionality of the binary, and is also developed outside of the organization that has ownership of the binary. Churn to common code has a disproportionate impact on the product as it results in all binaries undergoing churn. Whereas individual development groups focus on changes to code that impacts the binaries functionality, and ignores changes to common code.
   To normalize development metrics for binaries, the process separately measured the churn of common code and binary specific code.
2. Managing code ownership during organizational churn
   The Windows organization is very stable during product development, with the majority of organizational churn occurring between product releases. In our initial analysis we did not have to take into account engineers moving into and out of an individual product. Other products have much greater organizational churn, for instance where teams exist for specific sub-projects, on completion they may move onto completely different products.
   To normalize code ownership metrics it is necessary to derive project specific rules to define how code ownership is transferred when engineers or managers move between teams.

Within our process we are attempting to remove the domain specific attributes from the metrics and through this we are starting to see commonality in development behavior across products and across organizations. This is allowing us to start developing generic analytic models.

We have not yet been able to derive generic predictive models, but we do see the extraction of domain specific information from the development statistics as the solution to this problem. Specifically, our belief is that to develop generic predictive models you need to normalize the development metrics.

6 Summary

The computer industry is capable of producing generic predictive models if they are willing to apply the same restrictions as other engineering disciplines. Unfortunately, the limitations imposed on the development process to produce these models are too great for the majority

of software development projects. The unique challenge for the software industry is to generate generic models while placing no restrictions on the products development process. This short discussion argues that this is only possible if your collection process understands the development process well enough to remove domain knowledge from the process and thereby normalizing the development metrics.

In Microsoft, we are analyzing multiple products across multiple product groups and through separating out domain knowledge and development statistics we are starting to build generic analytic models. While we have not achieved the goal of developing generic predictive models we believe we are heading in the right direction.

References

Bartlett J, Gray J, Horst B (1987) Fault tolerance in tandem computer systems. In: Avizienis A (ed) The evolution of fault-tolerant computing. Springer Verlag

Chen WF, Duan L, Chen W (1999) Bridge engineering handbook. CRC Press, Boca Raton. ISBN: 0-8493-7434-0

Kitchenham B, Mendes E, Travassos GH (2007) Cross- vs. within company cost estimation studies: A systematic review. IEEE Trans Software Eng 33:316–329

Levidow B, Murphy B (2000) Windows 2000 dependability. IEEE International conference on dependability systems and networks, New York, NY

Nagappan N, Murphy B, Basilli V (2008) The influence of organizational structure on software quality preceedings of the 30th international conference on software engineering, Leipzig, Germany, 10–18 May 2008

Rolt LTC (1998) Red for danger, Sutton publishing limited, UK (first published 1955). ISBN: 0-7509-2047-5

Zimmerman T, Nagapppan N, Gall H, Giger E, Murphy B (2009) Cross-project defect prediction a large scale experiment on data vs domain vs process, ESEC/SIGSOFT FSE 2009. Amsterdam, Netherlands, pp. 91–100

Brendan Murphy is a Principal Researcher at the Microsoft Research Centre in Cambridge UK. Brendan works on the Empirical Software Engineering and Measurement (ESM) activities in Microsoft focusing on software reliability, dependability, quality and process issues. Prior to his current position at Microsoft, Brendan was at Compaq Corporation (previously Digital), Ayr Scotland till August 1999, where he ran the DPP program which collected and analyzed dependability data from customer sites. Prior to working in Scotland, Brendan worked for Digital in Galway Ireland, UNISYS (Scotland and US) and ICL (West Gorton, Manchester). Brendan graduated from Newcastle University.