Factors Explaining External Quality in 54 Case Studies of Software Development Projects

Chris Thomson and Mike Holcombe

Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP
{c.thomson, m.holcombe}@dcs.shef.ac.uk

Background: Confounding factors can easily make research hard to interpret and generalise. But there is currently no standard list of factors that should always be measured when conducting empirical investigations. **Objective:** To measure the explanatory power of eight simple metrics (two different pre-tests, number of members, total working time reported, development method used, test method used, formal specification method used, and programming language used) to explain external software project quality as measured by the project client. **Method:** We collected data on 54 software development teams over a five year period. A univariate analysis was used to calculate the explanatory power of the metrics and check for interaction effects between the categorical data. **Results:** Two of the proposed metrics (a pre-test based on a development project and the total time spent per team) led to significant explanation of the quality measurement. It was also noted that the differences between the Java and PHP programming languages did not explain the variation in quality, but some limited data available for JSP indicated this may not be the case for all languages. **Conclusion:** We recommend that any empirical investigations into external quality at least records the total time spent in man hours and an assessment of the competence of the developers. In addition future work is needed to determine if other programming languages explain variance in external quality.

External quality, empirical, multiple case study, factors, software development

1. INTRODUCTION

A reoccurring problem in the analysis of data from software engineering projects is the existence of confounding factors. These are factors that may not be measured as part of the data collection process but may nevertheless influence the outcome. Our investigative focus has been on collecting data from a large number of comparable case studies where some, but not all, of the factors are controlled. We have used the data collected to look at the development procedures used by the teams to try to identify the more successful techniques (Macias, 2004, Syed-Abdullah, 2005). Throughout this research we have assumed, based on the evidence available, that a number of factors had little or no impact on the performance of the teams as a whole. In this paper we make a retrospective investigation using summary data collected from 54 complete development projects which were similar in nature (CS-09-01, 2009). As there is variation in the projects undertaken we will treat this data as a set of case studies. The use of these case studies allows us to look at a large number of projects that were only controlled in some of the aspects to identify which factors affected the final outcome.

For each of the teams we have a measurement of external software quality, the *grade*, as assessed by the commissioning client of that system (Macias, 2004). The quality measurement was fairly simplistic with the client assessing the software for a few hours and under a small number of criteria. Therefore the measurement is weighted towards the identification of easily found problems and the assessment of the features the client requested rather than metrics such as robustness.

In this paper we investigate eight easily measured factors and their interactions, each per team: two different pre-tests, number of members, total working time reported, development method used, test method used, formal specification method used, and programming language used. The teams were instructed or chose to use various combinations of these, thus there is a mixture of literally and theoretically replicated cases (Yin, 2009). Not all combinations were covered and the data set was limited in various ways as described below. This approach was explored by Macías (a researcher at the University Of Sheffield), in particular he defines external validity of our data as being determined by multiple repetitions that allow for a greater number of novel experimental conditions to be encountered (Macias, 2004).

2. LITERATURE REVIEW

There are a wide range of data sources available to researchers in empirical software engineering; however where controlled experiments or multiple case studies are often used, three survey papers have reported problems with the tasks undertaken. Segal et al, reported on seven years of papers in the Journal of Empirical Software Engineering (Segal, 2003). One conclusion made was that empirical investigations do not factor out the context of the process. They emphasise that natural controls can be used instead of a formal experiment, by holding some variables constant (Lee, 1989). They argue that such studies can only be performed by observing practitioners in their workspace. Sjøberg et al. surveyed 103 papers that concerned controlled experiments. They found that the period of the experiment had a median length of 1 hour where specific times were recorded to two hours where the

available time was recorded, although the longest experiment lasted 55 hours. Sjøberg et al. highlighted that external validity was a problem in 16 experiments due to task size and in 34 due to the nature of the experimental material. Höfer and Tichy reported on 133 empirical papers (Höfer and Tichy, 2007). One of their conclusions is that “Long-term studies of programming methods, such as agile methods were missing, too.”

All three surveys call for deeper, longer and more contextual studies of software engineering practice. Due to the nature of long experiments it is both harder to enforce controls and hard to observe a team for the whole period. Furthermore to build up sufficient evidence not all experimentation or case observation will occur at the same time, which may lead to differences between the teams. In order to carry out research efficiently it is necessary to consider which factors might be important. One factor that is at least partly understood is the relationship between productivity per team member and team size. Conte reports that productivity exponentially declines in relation to team size (Conte, 1986), the consequence of this is that the productivity of a team only increases marginally with the number of team members.

3. METHOD

All of the case studies were sourced from the “Software Hut”, a undergraduate student module where self selected teams of developers work on real industrial projects (CS-09-01, 2009). The clients were sourced each year from a pool of contacts that approached us with projects. In each case the client graded the teams that worked on his projects on the basis of their final delivered software. The data was collected over a five year period and whilst the clients were different each year, in individual years multiple teams work on identical projects concurrently in competition with each other. The teams were organised according to the randomised complete block experimental design, with three to four teams per client and a balanced distribution of treatments between teams working with each client (Ostle and Malone, 1988). Each project lasted for 12 weeks, with each individual developer expected to spend 15 hours working on the project per week.

Over the period of data collection used in this analysis the conditions on the course changed from a comparison between teams following Extreme Programming (XP) (Beck and Andres, 2004) and plan driven approaches (Simons, 2009) to a comparison between XP teams following test first and other testing practices.

Throughout the data collection period the developers were instructed to follow one of the methods mentioned and record a weekly timesheet. The programming language, formal specification technique and number of developers per team were identified by inspecting the product and project records. Where more than one programming language was used the most common was selected. The only formal specification technique used was the Extreme X-Machine technique (Holcombe, 2008), other teams did not use a technique (Thomson and Holcombe, 2005). The pre-test scores were obtained from the students grades on two modules in the previous year; we found that these modules over our whole data set explained the most variance in the assessment of quality made by the client. The details of the data collected, and the summary data can be found in a technical report (CS-09-01, 2009).

The data was processed with SPSS 13 using a univariate general linear model to calculate the variance in the client grades explained by the measured factors. Principally two analyses were made due to the nature of the data collected: namely only the XP teams used a formal specification and different test methods, thus the first analysis compared the design methods of XP and plan driven approaches and the second analysis just the teams that used XP were included but also the additional data for the formal specification and test methods. The model was arranged such that the client grade was entered as the dependant variable, the number of team members, hours worked and pre-test results as covariates. The remaining factors were entered as fixed factors.

3.1 Threats to Internal Validity

The teams working with different clients developed different software products, but all had equal motivation to follow the process. In order to address the issue of interaction the teams were aware that this was a competition; equally the client was told not to share ideas between teams. To mitigate the effects of the teams' maturing we allowed the members to self select, hence some members in each team had worked together before. The team members may have learnt, becoming more experienced over the course of the development period, but we have no obvious way to measure this and by viewing the projects each as a whole.

Lastly the measurements of design method, test method and formal specification are binary in nature, that is to say these techniques were measured as being followed or not followed. This is clearly a poor measurement as in all three cases it would be possible to partially follow the full method, and this could effect the potency of that method to deliver the result which is claimed. Thus it is possible that the significance of any results could be unrepresentative if the method was used to its fullest extent in all cases.

3.2 Threats to External Validity

The developers were novice users of XP although they had completed a previous team software development project. The task was representative of other small web based development projects where there is a development team of 4-6 members taking up to 120 man hours per person (Tichy, 2000). The developers worked at home and in a university laboratory, having access to a range of professional software tools.

4. RESULTS

The variability of the data set used was found to be fairly limited, of the 140 projects in the Observatory archive 67 were found to have recorded details of all the factors that were considered in this analysis. Of these we discarded

a further thirteen as we found that for programming languages other than Java and PHP there were only a handful of cases (JSP was the next largest with 4 cases). Given the experimental design (where there were 3-4 teams working with each client, and the number of other factors) having such a small number of cases could lead to a biased result. Within this set of data all of the XP teams were recorded as following test first or test last, and either as having used the Extreme X-Machine formal specification technique or not. Table 1 shows the cases of theoretical and literal replication that were present in the data set. The team size, client grade and number of hours worked varied on a continuous scale.

TABLE 1: Overview of replications within the multiple case studies analysed.

Project factors	Number of teams
Plan-driven, PHP	1
Plan-driven, Java	7
XP, Extreme X-Machine, TF, PHP	18
XP, TF, PHP	4
XP, Extreme X-Machine, TL, PHP	5
XP, TL, PHP	2
XP, Extreme X-Machine, TF, Java	8
XP, TF, Java	6
XP, Extreme X-Machine, TL, Java	3

The result of the analysis of the effect of the design method is shown in table 2. Of the factors considered the mean value of pre-test 2 was significant, as was the total time spent by the development team in hours ($p < .05$). Pre-test 2 is a measurement based on the developers' performance in a module on requirements engineering which included a practical element, in contrast with pre-test 3 which was based on java programming; however this only explained less than 10% of the overall grade. This suggests that developers who understand the development process consistently do better than other developers. The fact that around 17% of the variation in grade can be attributed to the variation in time spent is not unexpected. Equally the lack of any explanation based on the number of developers reflects the previous results where it was found that productivity had a negative exponential relationship with the number of developers. The fact that design method did not explain grade reflects our previous results where differences found between the XP and plan driven approaches appeared to be small (Macias, 2004, Syed-Abdullah, 2005).

Of perhaps most useful significance is that programming language did not have an effect, this is clearly important when comparing multiple cases studies as it means this factor can be ignored. However we also looked at the results including JSP, and this caused language to be significant ($p = .042$, eta squared = .124). This is difficult to interpret due to the small number of cases where JSP was used, which had a mean value that was less than Java and PHP, therefore we recommend that future research collects more data about other languages to investigate this result

TABLE 2: Analysis of variance between the design methods of XP and plan driven.

Source	Type III Sum of Squares	df	Mean Square	F	P Sig.	Partial Eta Squared
Pre-test 2	116.654	1	116.654	4.859	.033	.096
Pre-test 3	16.147	1	16.147	.673	.416	.014
Team size (number of members)	6.658	1	6.658	.277	.601	.006
Total time in hours	230.593	1	230.593	9.604	.003	.173
Design Method	15.992	1	15.992	.666	.419	.014
Programming Language	19.857	1	19.857	.827	.368	.018
Design Method * Programming Language	11.861	1	11.861	.494	.486	.011
Error	1104.458	46	24.010			
Total	75129.000	54				
Corrected Total	1720.093	53				

Table 3 contains the result of the analysis on the XP teams, where the test method and use of formal specification factors were included. For this data only one factor, the time spent in hours, is significant ($p < .05$), once again explaining around 17% of the variance. As with the previous analysis of design method no interaction effects were found between the other factors.

5. DISCUSSION AND CONCLUSIONS

We used a historical archive of data collected from 54 software development projects of similar size and context to investigate the relationship between eight factors and external quality as assessed by the projects' client as a grade. Five of the factors were measure directly, one was self reported by the developers (the total time spent on the project), and two were determined by the techniques the teams were told to use. The teams were composed of second year undergraduate students with previous experience of software development in teams.

The data was analysed by creating a univariate analysis of variance. This showed that a pre-test based on java programming aptitude (pre-test 3), the team size, design method, programming language, test method and

formal specification did not have a significant effect on the grade awarded by the client. However a pre-test based on the development project aptitude (pre-test 2) and the total time spent on the project in hours did have an effect. We can therefore conclude that these factors have a bearing on the ability of a developer to complete a project, although with this test we cannot conclude that this was an independent effect as both of these factors were treated as covariates in the analysis. We recommend that empirical investigations into the external quality of software development projects should measure at least these factors.

TABLE 3: Analysis of variance between test first, test last and the use of X-Machines in teams following an XP method.

Source	Type III Sum of Squares	df	Mean Square	F	P Sig.	Partial Eta Squared
Pre-test 2	87.119	1	87.119	3.141	.085	.082
Pre-test 3	15.296	1	15.296	.552	.463	.016
Team size (number of members)	.433	1	.433	.016	.901	.000
Total time in hours	195.833	1	195.833	7.061	.012	.168
Programming Language	.520	1	.520	.019	.892	.001
Test Method	6.188	1	6.188	.223	.640	.006
Formal Specification	20.356	1	20.356	.734	.397	.021
Programming Language * Test Method	3.721	1	3.721	.134	.716	.004
Programming Language * Formal Spec	14.301	1	14.301	.516	.477	.015
Test Method * Formal Spec	1.088	1	1.088	.039	.844	.001
Programming Language * Test Method * Formal Spec	.000	0	.	.	.	.000
Error	970.679	35	27.734			
Total	64091.000	46				
Corrected Total	1633.935	45				

With reference to the discussion validity, this analysis was limited due to the lack of a measurement of fidelity or extent that the methods were applied. This may have reduced the ability of the factors to explain variance in the client grade. In particular it seems likely that not all the teams applied extreme programming well based on our previous observations (Syed-Abdullah, 2005). Therefore we plan to revisit our archive data and attempt to quantify these before reanalysing the data.

The result that showed that the programming language used did not have a significant effect on the clients' grades is potentially useful as it allows projects to be more easily compared. However the analysis only took account of two languages, Java and PHP, furthermore the limited data available for JSP suggested that there maybe an effect. Therefore we recommend that whilst Java and PHP could be treated as delivering equivalent external quality, further data is collected about other languages.

This work was supported by an EPSRC grant: EP/D031516 - the Sheffield Software Engineering Observatory.

REFERENCES

- BECK, K. & ANDRES, C. (2004) *Extreme Programming Explained: Embrace Change (2nd Edition)*, Addison-Wesley Professional.
- CONTE, S. D. (1986) *Software engineering metrics and models*, Menlo Park, California, The Benjamin/Cummins Publishing Company.
- HÖFER, A. & TICHY, W. (2007) Status of Empirical Research in Software Engineering. *Empirical Software Engineering Issues. Critical Assessment and Future Directions*.
- HOLCOMBE, M. (2008) *Running an agile software development project*, John Wiley.
- LEE, A. S. (1989) A scientific methodology for MIS case studies. *MIS Q.*, 13, 33-50.
- MACIAS, F. (2004) *Empirical Assessment of Extreme Programming*. PhD Thesis, Department of Computer Science, University Of Sheffield.
- OSTLE, B. & MALONE, L. (1988) *Statistics in Research: Basic Concepts and Techniques for Research Workers*, Iowa State Press.
- SEGAL, J. (2003) *The nature of evidence in empirical software engineering*. Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on. IEEE Computer Society.
- SIMONS, A. (2009) *The Discovery Method for Object-Oriented Software Engineering* [Online]. University of Sheffield. Available from: <http://www.dcs.shef.ac.uk/~ajhs/discovery/>
- SYED-ABDULLAH, S. (2005) *Empirical Study on Extreme Programming*. PhD Thesis, Department of Computer Science, University Of Sheffield.
- CS-09-01, (2009), The Sheffield Software Engineering Observatory Archive: Six Years of Empirical Data Collected from 73 Complete Projects, Department of Computer Science, University of Sheffield, Sheffield, UK.
- THOMSON, C. & HOLCOMBE, W. (2005) *Using a formal method to model software design in XP projects. 2nd South-East European Workshop on Formal Methods*. South-East European Research Centre.
- TICHY, W. (2000) Hints for Reviewing Empirical Work in Software Engineering. *Empirical Software Engineering*, 5, 309-312.
- YIN, R. K. (2009) *Case Study Research: Design and Methods*, London, Sage.