The Relationship Between Software Process, Context and Outcome

Dag I.K. Sjøberg[1,2(✉)]

[1] Department of Informatics, University of Oslo, Oslo, Norway
dagsj@ifi.uio.no
[2] SINTEF ICT, Trondheim, Norway

Abstract. Most practitioners and researchers agree that when developing software, process affects product, and the usefulness of a process depends on the context. However, which processes are most useful for a specific company or project is generally unknown. When studying the relation between context, process and product, one challenge is that experiments often lack realism, which makes the transfer of results to industry difficult. In contrast, most of the important factors vary beyond the researcher's control in case studies, which makes it difficult to identify cause and effect relationships. This paper reports a study where realism was combined with control over certain context and process factors. Four companies developed the same system, and the price varied by a factor of six. Certain patterns of relationships were expected (expensive company, low cost, schedule overrun); others were unexpected (cheap company, maintainable system because of small code). The community needs to identify the most important relationships among process, context and outcome.

Keywords: Software process improvement · Controlled multiple-case study · Software industry · Theory · Software engineering folklore · Measurement

1 Introduction

I am regularly contacted by various organizations for help regarding their software processes. They range from small, private companies to large, public sector agencies, some of whose projects failed to the order of hundreds of millions of euros. These organizations are not interested in general, overall principles regarding process; they are interested in what would work for them. They expect us, as researchers in the field, to know "what works for whom, where, when, and why" [1].

In a few cases, I have immediate suggestions for improvement, such as introducing automated testing if the number of defects is out of control. But in most cases, I cannot propose anything without working with the organization for some length of time. Identifying and measuring the factors that should be taken into account when proposing process changes are far from trivial; the software engineering literature does not give much help in concrete settings. The current body of knowledge is mostly too general or too specific; see Fig. 1.

P. Abrahamsson et al. (Eds.): PROFES 2016, LNCS 10027, pp. 3–11, 2016.
DOI: 10.1007/978-3-319-49094-6_1

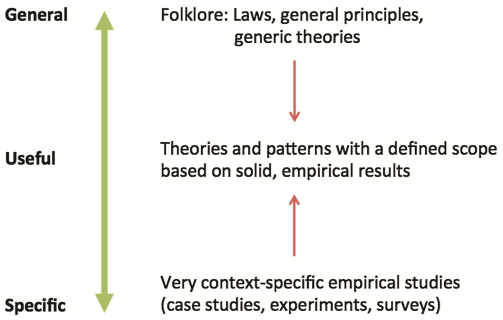

Fig. 1. Useful theories and patterns

Software engineering folklore guides processes to some extent. For example, a collection of "laws" and principles that have emerged over the years can be found in [2–4]. One of the laws stated by Endres and Rombach [4], attributed to the work by Boehm [5], is: "Errors are most frequent during the requirements and design activities and are more expensive the later they are removed." This "law" encourages processes that emphasize the analysis and design phases. However, what does this mean in practice? Should one spend, say, 10 to 30 percent of the total effort of a development project in these phases? Because of varying contexts, software engineering folklore is often contradictory. For example, much effort has been devoted to developing process models that include a large number of activities, practices and roles together with formal, detailed project documents. In contrast to such heavy processes are the light processes recommended in agile development.

Few empirical studies in software engineering discuss the contexts to which the results may be generalizable. Experiments in software engineering generally have few subjects and almost all of them use convenience sampling [6]. Most case studies are of single cases, and few attempt to generalize the results through theories with a well-defined scope of validity [7]. Surveys collect people's subjective opinions, which are based on knowledge and experience gained in specific contexts. The results of surveys also need to be related to theories to become generally useful.

Nevertheless, a premise in software engineering is that there is a relationship between software processes and success of a project or task. The success is typically described in terms of the quality of the delivered software, how long it takes to develop it and how much it costs. It is also commonly agreed that this relationship is moderated by the context of the processes, as illustrated in Fig. 2. It is reasonable to assume that an optimal process varies with context; for example, a small team may not benefit from activities designed to help large teams.

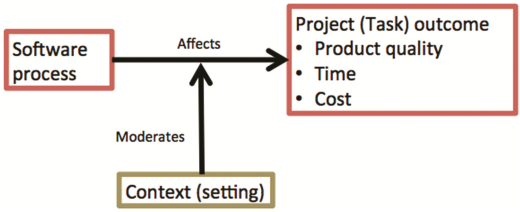

Fig. 2. Relationship between process, context and outcome

The ideal model would be a deterministic one, in which a set of given context and outcome parameters would determine the optimal process, and a given process and context would determine the outcome. However, it is unlikely that we will manage to develop such models given that software development is a mostly human activity and we are unable to describe human behaviour deterministically in general, even though certain theories describe human behaviour in specific situations, for example, the prospect theory [8].

Although we are far from a scenario where we can fully determine which process gives which result in a specific context, our community can improve in identifying patterns and proposing theories for the relationships among process, context and outcome. This paper reports on a study that is an ongoing attempt in that direction.

2 Design of Study

The empirical software engineering community conducts both controlled experiments, which focus on cause and effect, and case studies, which focus on realism. How to identify cause–effect relationships in realistic settings is a challenge. What if we hire several companies to develop the same system and see what happens? Some years ago, our research group had such an opportunity. We needed a web application to store information about all the empirical studies of the group. We developed a requirement specification and sent a call for tender to 81 consultancy companies and received bids from 35 of them. A study of this bidding process was reported in [9].

The striking difference in the bids, given that we provided a well-defined 11-page requirement specification, led us to use price as the selection criteria. We wanted to study the effect of price on process and outcome. Thus, in four price segments, we selected the company that appeared most likely to develop a good system based on the quality of the bid documents. The companies are named *A* to *D* in this paper, in the order of bid price; see Fig. 3.

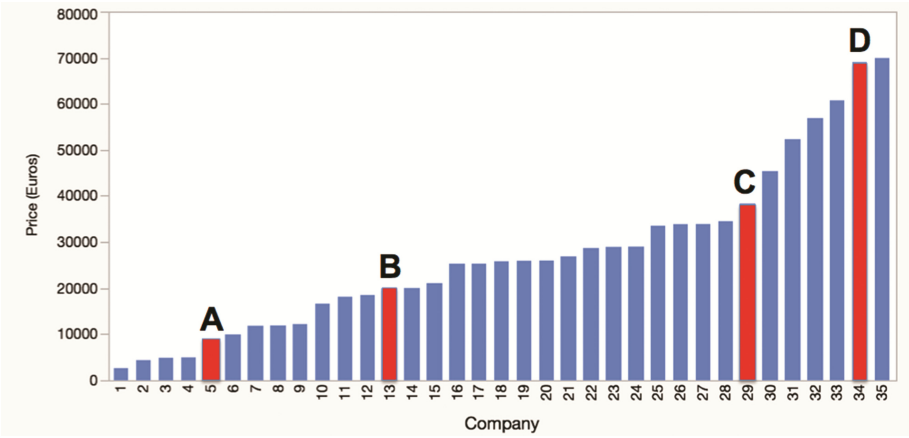

Fig. 3. Four out of 35 companies selected for development

The data sources in this study are comprehensive. They include daily time sheets on tasks and subtasks of each developer, weekly snapshots of all documents including source code produced during the projects, full history provided by the configuration management and issue tracker tools and other information collected from defect logs, e-mail communication and team interviews.

From this study, we published an investigation on reproducibility and variability in software engineering [10] and a study of effort estimation based on use case points [11]. The code developed by the four companies has also been used in follow-up studies on maintenance metrics [12] and effects of code smells [13]. (In this paper, the companies are named according to bid price, while in the papers already published, the order was alphabetic. Company C is now Company A, Company A is now Company B, and Company B is now Company C. Company D remains Company D.)

A detailed investigation of the effect of process and context has not yet been published. Initial results are reported here.

3 Context

We controlled parts of the context to make them the same for all the companies; other context factors were specific to each company. The controlled ones included:

- Requirement specification
- Application domain (web document management)
- Functional size of the system (57 unadjusted use case points [11])
- Low complexity of system
- Customer (our research department)
- Programming language (Java, Javascript and SQL)

- Tools (IDE: Netbeans or Eclipse, Build & Deploy: Ant, Configuration management: CVS; note that these tools were selected by the companies themselves but they happened to be the same by accident)
- Team composition (1 project manager and 2 developers, except in Company B, which had 1 developer and 1 interaction designer)
- Uniform interaction between development team and customer (e.g., use of same issue tracker, acceptance tests by the same customer team)
- Intermediate skill level of the developers

Regarding skill level, we selected the developers on the basis of their CVs. All of them had at least three years of formal education in programming and three years of industrial experience with the technology to be used. Ideally, we should have tested the developers using a validated skill evaluation instrument [14], but in the absence of such an instrument at that time, the developers were tested for their Java skills by taking part in a one-day exercise in which they performed the same Java programming tasks used in a former experiment [15]. Their performance was thus compared with that of 77 other Java programmers. Similarly, the developers were tested for their design skills by taking part in a half-day UML exercise where they performed the same tasks used by 28 persons in a former experiment on use cases and class diagrams [16]. We did not observe any clear relationship between the skills of the team and project outcome.

Table 1 shows context factors that varied among the companies. Some factors were specific to the development organization; others were specific to this development project. Note that the bid by Company D, of 69,000 euros, shown in Fig. 3, was negotiated down to the 56,000 euros, shown in Table 1.

Table 1. Varying context

Aspect	Variable	Company A	Company B	Company C	Company D
Development organization	**Size (# employees)**	Appr. 8	Appr. 100	Appr. 25	Appr. 13,000 worldwide
	Nationality	Domestic	Domestic	Domestic	International
	Ownership	By employees	Private	By employees	Listed exchanges
	Location	Bergen	Oslo	Oslo	Oslo, 20 countries
	Process models	Light	Intermediate	Intermediate	Heavy
Project	**Firm price**	€8,750	€20,000	€45,380	€56,000
	Agreed time schedule	41 days	55 days	73 days	62 days
	Estimated effort	100 hours	220 hours	341 hours	650 hours
	Allocation	Part-time	Part-time	Part-time	Full-time
	Co-location	No	No	No	Yes

The table shows several internal relationships among the factors. Company A is small and can therefore only run fairly small projects with small teams. Their organization-level process models are therefore light. The low price offered to build the system is followed by an expectation of a short lead time, a low number of effort hours, and the need for the developers to work on several projects in parallel. At the other end, Company D is large and has a heavy organization-level process model. The high bid allows higher estimated effort and allows developers to work full time on this project.

4 Processes

As an example of process data, Fig. 4 shows the number of hours the companies spent on various development activities. Note the one-to-one correspondence between the effort spent on the activities "Implementation" and "Analysis and Design". There is no indication here that much effort spent on analysis and design reduces the effort needed on implementation, or vice versa. Remember that the amount of functionality is fixed. Figure 5 shows the hours spent on the activities as the projects were running.

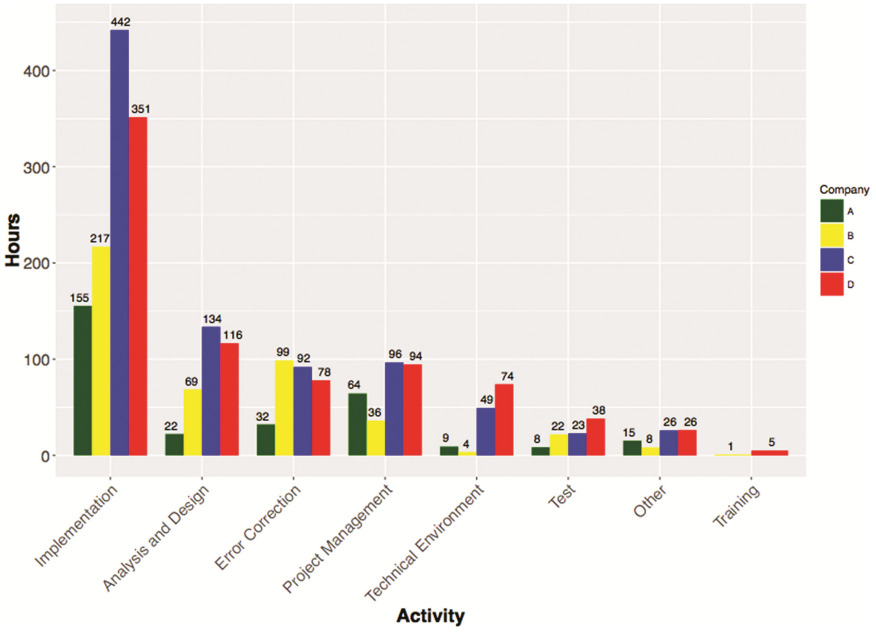

Fig. 4. Effort spent on various activities

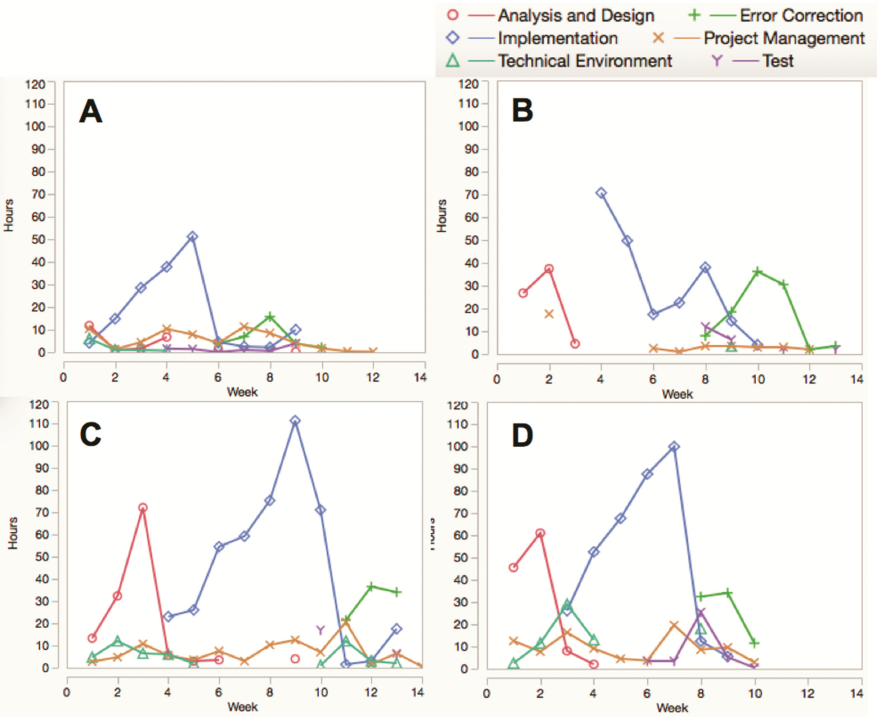

Fig. 5. Effort on activities along the way

5 Outcome

The outcome of a process or project may be measured along many dimensions. For the four systems, we assessed reliability, usability and maintainability. Reliability was measured by investigating the defects found over a period of two years when the systems were operational [10]. The usability was measured through a dedicated experiment [17] and maintainability in a follow-up experiment [13]. Table 2 shows the results and also includes measurements of effort and lead time.

Table 2. Outcome variables

Aspect	Variable	Company A	Company B	Company C	Company D
System	Reliability	Poor	Good	Good	Fair
	Usability	Fair	Good	Fair	Good
	Maintainability	Good	Fair	Poor	Fair
Effort	Actual effort	315 hours	562 hours	894 hours	796 hours
	Overrun effort	215 %	155 %	74 %	22 %
Lead time	Actual lead time	79 days	87 days	90 days	65 days
	Overrun lead time	93 %	58 %	23 %	5 %

6 Relationships

Did the rather extreme price differences, of a factor from 1 to 6, lead to corresponding differences in outcome? Generally not. Company A had given the lowest bid and accordingly spent the least effort on the whole development, particularly on analysis and design and on testing. In a sense, this company developed a "quick and dirty" solution, but the small size of their Java code led to the most maintainable system. The low number of lines of Java code trumped other maintainability metrics [12]. On the other hand, the low focus on testing resulted in the least reliable system and required us as a customer to spend much more effort on testing than we did for the other companies. In total, we spent almost twice the number of effort hours on Company A as we did on the other companies, which to some extent reduces the cost savings of hiring Company A. Furthermore, we were a competent customer; an incompetent customer might have resulted in a failed project.

Company B scored best on the system quality dimensions on average. Given the next lowest price, one may consider their project as the best value for money.

Company C over-designed their system, which resulted in excess code size, which in turn resulted in poor maintainability. But they scored top on reliability.

Company D had relatively heavy processes and a highly competent project manager. The developers worked full-time and were co-located. Their project seemed to have full control all the way and resulted in the lowest lead time and very little overrun.

We have observed many other relationships among context, process and outcome, but much analysis remains. We hope to reveal interesting patterns that may shed new light on existing theories or be the basis for new theories.

References

1. Dybå, T., Sjøberg, D.I.K., Cruzes, D.S.: What works for whom, where, when, and why? On the role of context in empirical software engineering. In: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. ACM (2012)
2. Brooks Jr., F.P.: The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley Publishing Company, Reading (1975)
3. Glass, R.L.: Facts and Fallacies of Software Engineering. Addison-Wesley Professional, Reading (2002)
4. Endres, A., Dieter Rombach, H.: A Handbook of Software and Systems Engineering: Empirical Observations, Laws, and Theories. Pearson Education, New York (2003)
5. Boehm, B.W., McClean, R.K., Urfrig, D.E.: Some experience with automated aids to the design of large-scale reliable software. IEEE Trans. Softw. Eng. **1**, 125–133 (1975)
6. Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.K., Rekdal, A.C.: A survey of controlled experiments in software engineering. IEEE Trans. Softw. Eng. **31**(9), 733–753 (2005)
7. Sjøberg, D.I.K., Dybå, T., Anda, B.C., Hannay, J.E.: Building theories in software engineering. In: Shull, F., et al. (eds.) Guide to Advanced Empirical Software Engineering, pp. 312–336. Springer, London (2008)
8. Kahneman, D., Tversky, A.: Prospect theory: an analysis of decision under risk. Econometrica **47**(2), 263–291 (1979)

9. Jørgensen, M., Carelius, G.J.: An empirical study of software project bidding. IEEE Trans. Softw. Eng. **30**(12), 953–969 (2004)
10. Anda, B.C.D., Sjøberg, D.I.K., Mockus, A.: Variability and reproducibility in software engineering: A study of four companies that developed the same system. IEEE Trans. Softw. Eng. **35**(3), 407–429 (2009)
11. Anda, B., Benestad, H.C., Hove, S.E.: A multiple-case study of software effort estimation based on use case points. In: International Symposium on Empirical Software Engineering, pp. 407–416 (2005)
12. Sjøberg, D.I.K., Anda, B.C., Mockus, A.: Questioning software maintenance metrics: a comparative case study. In: ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (2012)
13. Sjøberg, D.I.K., Yamashita, A., Anda, B.C., Mockus, A., Dybå, T.: Quantifying the effect of code smells on maintenance effort. IEEE Trans. Softw. Eng. **39**(8), 1144–1156 (2013)
14. Bergersen, G.R., Sjøberg, D.I.K., Dybå, T.: Construction and validation of an instrument for measuring programming skill. IEEE Trans. Softw. Eng. **40**(12), 1163–1184 (2014)
15. Arisholm, E., Sjøberg, D.I.K.: Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE Trans. Softw. Eng. **30**(8), 521–534 (2004)
16. Anda, B., Sjøberg, D.I.K.: Investigating the role of use cases in the construction of class diagrams. Empirical Softw. Eng. **10**(3), 285–309 (2005)
17. Følstad, A., Anda, B.C.D., Sjøberg, D.I.K.: The usability inspection performance of work-domain experts: An empirical study. Interact. Comput. **22**(2), 75–87 (2010)