Effects of Software Engineering Practices on the Scalability of Firms' Software Development Output

Mika Ahokas, Jyrki Kontio, Markus M. Mäkelä, Päivi Pöyry, Aki Lassila
Helsinki University of Technology, Software Business Laboratory
P.O. Box 5500, 02015 HUT, Finland
firstname.lastname@tkk.fi, http://www.sbl.tkk.fi/

Abstract

Due to the dynamic nature of the global software industry, many software firms experience rapid growth and numerous subsequent changes to demands for the management of business. Software firms' ability to scale up the output of their operations to meet increased demand in a flexible and cost-effective way is necessary for their sustained growth. In this paper, we explore how software engineering practices affect the scalability of firms' software development output. We provide definitions for scalability and for a related quantity that we label relative scalability. Focusing on software development, we present early empirical findings that spearhead research into such scalability and describe factors that affect relative scalability. Our tentative findings indicate that key factors that improve relative scalability are modular architecture and platforms, automated tools, effective requirements engineering, smallness of teams, flexibility of structures, productization of total offering, parametrization of software products, and outsourcing of non-core activities.

1 Introduction

The software industry has become an increasingly important business sector that also bears a notable impact on the remainder of the economy [4, 8, 9]. Rapid growth of the sector, increasing economic importance connected with the special characteristics of software itself and the dynamics of the software industry make software's technological, process, and business aspects an interesting and challenging area to study [9, 2]. Software engineering activities typically have a major influence on software firms' overall economic performance, growth potential, and capabilities to efficiently scale their output, e.g. [10, 11, 1].

Challenges in software engineering practice, coupled with the dynamism of the software industry, form the rationale for conducting the research reported in this paper: there is a need to understand how to increase the scalability of the software firm and how to increase the ratio of change in outputs to change in inputs in order to meet the needs of growing business. In this context, the scalability of the software development function, is a major component of general business scalability.

Our purpose is to integrate existing software engineering principles and practices with empirical findings in order to develop an understanding on how the various aspects of software engineering practice affect the firm's *scaling capability* (the firm's ability have high scalability: its ability to transform its resource base to allow scaled-up output – see e.g. [6] on such dynamic capabilities). Our research question is *how the firm can increase its scalability from a software engineering viewpoint.* Furthermore, we aim to identify candidate factors of software engineering practices and principles that affect scalability of the software firm. Our goal is to study how good software development practices uphold and improve software firms' capabilities to scale up their business.

2 Scalability of the Software Firm

Recent accounts in both academic [8] and practitioner-oriented [4] research discuss economies of scale in relation to software business. Yet, the literature does not propose any suitable definition for the scalability of the software firm's output.

For the context of the software firm, we define *absolute scalability* as the firm's ability to adjust the quantity of its output. Output could be measured, e.g., as the amount of functionality delivered, number of product variants, number of customer deliveries, or level of quality of the product – or, in a more business-oriented way, as revenue growth. However, as absolute scalability only relates to the changes in output, we believe that the scalability of a software firm is most

relevant in terms of how the output changes with respect to input. We refer to 'input' as resources needed in producing output, such as human resources, financial resources, organizational knowledge, and social capital. Examples of 'input' in the software engineering context include engineers and their knowledge, third party licenses, subcontractors, partners, platforms, tools and technology, facilities and infrastructure, and financial resources.

We define *relative scalability* as change in output in relation to change in input. The relation between output and input can be non-linear (i.e., marginal changes in input result in different changes in output, depending on the input levels) and it may have points of discontinuity (e.g., acquisition of a company or building a new facility).

Relative scalability is high when the software engineering output increases substantially (capability to deliver in higher volumes), compared to inputs (i.e., resources) used. Respectively, relative scalability is low when output increases less than resources. In this paper, we are interested in factors in software development function that are relevant to scalability. High relative scalability of the software engineering function refers to ability to deliver higher volumes of functionality, product variations, and customer installations with relatively small changes in the resources that are used to create this output. With the high level of scalability the firm can scale up the output without increasing the input at the same rate.

3 Research Methods

We have used both inductive and deductive research approaches. We initially used the grounded theory approach [5, 12] to form initial insights inductively on the basis of multiple case study data. This approach is suitable since we were studying a rarely investigated phenomenon for which existing theories appeared to provide only limited answers. We used the replication logic [13] in which the cases are treated like a series of independent experiments that confirm or disconfirm emerging conceptual insights.

3.1 Empirical Study Design and Procedure

To implement case study approach, we studied three firms, who participate in our ongoing research project on internationalization. Each firm is located in Finland but also has foreign sales. Case company one is publicly held and the other two are privately held. These companies compete in different market segments. The publicly held company is large (more than 250 employees) and the other two are medium sized (less than 250 employees) companies.

We collected the empirical data through interviews, observations, and documentation (company fact sheets, the Internet, and other secondary data sources). The primary source were the semi-structured interviews that partly comprised standard interviews with individual respondents and partly auditing company-wide workshops with professionals working in various levels and positions.

We conducted a total of 31 semi-structured interviews that were recorded using laptop microphones, transcribed, and analyzed. First, we gathered data by interviewing 5-6 respondents from each company. The interviews lasted 60-120 minutes each, averaging 90 minutes. Second, we analyzed the data from each company separately, following Eisenhardt's [5] within-case analysis procedure. This phase helped develop a basic understanding about the relevant phenomena and the current state in the case companies.

During our analysis, by the grounded-theoretical means of constantly iterating between data, literature, and emergent theoretical framework, we reached our first understanding of scalability and relative scalability.

We needed to understand that how companies could improve their relative scalability. Hence, during our iterative analysis procedure, we built a conceptual framework that integrated findings from literature, case-specific insights such as bottlenecks identified in the case companies, and a synthesis for future improvements. For each part of the software process, we conducted the literature review. In accordance with the iterative research process, we then gathered more empirical data by interviewing key people from the firms' software development function. In two firms, we arranged half-day workshops where we discussed the phenomenon with key people from software development and top-management of the firm.

During the analysis process, we also wrote detailed descriptions of each case, in relation to our analysis employing the conceptual framework. Our research led to a synthesis that defined key factors affecting the scalability of the firm. For each case company, we made case specific recommendations for future improvement. In this paper, we present six key factors (presented in Section 4) as candidates to describe management of the scalability of the firm.

3.2 Data Analysis and Generalizability

The data analysis was carried out by first analyzing the individual case studies and then conducting a cross-case analysis in order to construct a conceptual framework [5]. The cross-case analysis was done

relying on methods suggested by Yin [13] and Eisenhardt [5]. In each case, we incorporated into the analysis of this study the impact of company- and industry-level professionals, such as software engineers, software company top-managers such as CEOs and CTOs, informants, professors, and researchers.

Although this preliminary research concentrated on only three particular software companies and their relative scalability, the results of this case study can be said to be generalizable on the analytic level (generalization to theory, typical to case study research, see [13]).

4 Findings and Discussion

In this section, we present our findings on factors affecting the relative scalability of the software firm. These tentative findings are based mainly on our empirical case data.

Modular architecture and technology platforms. Modular, component-based architecture and reuse are critical enablers for relative scalability, but require up-front investments. Platforms can influence relative scalability due to the increased compatibility and modules by making it easier to manage the software system. Both platforms and modules can provide high level reusability that increases the level of productivity and relative scalability. Modular architecture connected to a viable platform can also make it easier to outsource parts of the system to third parties. Our case studies imply that firms often overestimate the level of their software system architecture and modularity. Typically problems occur, when firms try to outsource parts of the system or when they try to design software in tight collaboration with customers or suppliers. In these cases, platforms and modular architecture could provide increased possibilities for collaboration with suppliers and customers in order to gain higher relative scalability. See also [8].

Automated tools. Software development is usually seen as a design activity and automation is typically connected only to manufacturing activities. However, our case data implies that automated tools can increase relative scalability. For example, testing automation can increase relative scalability by decreasing the level of bugs, which decreases the need for time-consuming bug-fixing and customer support. In addition, our case data indicates that testing automation can enhance release management process and product quality, which have positive effects on relative scalability.

Effective requirements engineering. Effective requirements elicitation can enhance relative scalability. We found that the right selection of key features have notable effects on relative scalability. In one of our case companies, the primary bottleneck in software development was insufficient requirements engineering due to the lack of requirement prioritization. The key issue seemed to be how to collect and analyze requirements from multiple sources. In conclusion, when the number of customers increase to a certain point, there is a need for systematically defined methods to collect, analyze, and prioritize requirements. See also [10].

Small teams and flexible structures. In our cases, we found that source of flexibility and adaptability to scale up software development in a flexible way is based on sufficient smallness and effectiveness of teams working in parallel. Large teams are disadvantageous as group size correlates negatively with performance after a certain point [7]. Teams larger than ten are violating the ‘small is beautiful rule’ [3]. In general, our case companies followed the ‘small is beautiful rule’: in large projects, development was distributed to several small sub-teams, typically no more than 6-8 developers. The case firms also reported that it is an effective approach to grow a software organization by small increments, small teams, in manageable units. Based on case studies, we can conclude that firms can scale up their software development organization, while ensuring advantages of small firms by dividing systematically and regularly the large units into smaller ones.

Productization and parametrization. Our case data implies that ‘productization’ of the total offering connected with an effective software product ‘parametrization’ can increase relative scalability by reducing human level intervention needed to develop and deliver products and services to customers and to provide support with them. Examples of productization include clear product and service offering, well-packaged product releases, standardized development methods, documentation, frequently asked questions lists, support databases and other support material. In addition, parametrization makes it easier to customize software products for customers in an efficient and cost-effective way without excessive software development work. Furthermore, two of our case companies reported that parametrization reduced significantly their product delivery time to customer.

Outsourcing of non-core activities. Contrary to general assumptions, outsourcing was not seen as an effective way to increase relative scalability as generally thought, if it concerned complex core activities of a software product. However, the case companies had many positive experiences when outsourcing small and well specified parts of the software products. The problem was that complex core

activities need high levels of understanding, documentation and communication between software developers. Finding a subcontractor that has such high-level knowledge to produce desired high quality output is challenging. For example, one case firm tried to outsource part of the software development function but due to the complexity of the products this attempt led to efficiency losses and other difficulties. However, all case firms reported positive experiences when outsourcing small and well-specified parts of the software products. Our case studies also imply that increased level of modularity supports outsourcing activities.

5 Conclusion

In this paper, we have shed light on the software firms' ability to scale up the output of their operations, especially with regard to a quantity that we labeled relative scalability. As a result of our case research we outline factors that affect the software firm's relative scalability. The factors identified in this study may apply to software engineering organizations in general. However, at this stage, we have presented factors associated with the relative scalability. Furthermore, we also tentatively identified other potential factors including process capabilities. However, they are not reported here due to their incompleteness and inconclusiveness in the context of our dataset. To better understand how software firms can scale up the output of their operations, we must focus on both software development and the dynamics of business. Due to the special characteristics of software as an industrial output and economic good, software engineering practices has major influence on the software firm's scaling capability.

We found that relative scalability of the software firm's output is a challenging and understudied phenomenon. Positive feedback both from the academic research community and the global software industry has strengthened our expectation that the scalability of the software firm's output is an industrially relevant research area. Hence, we hope to have opened avenues for continued research on scalability. The study of the area can be taken deeper for instance in the study in software development and the study of software marketing.

6 Acknowledgements

We would like to thank Matti Heikkonen, Manager, Strategic Alliances and Partnering, Nokia Corporation, and researchers Olli-Pekka Mutanen, Jani-Pekka Jokinen, Nilay Oza and Maija Perttula from the Software Business Laboratory of Helsinki University of Technology for fruitful discussion and comments.

7 References

[1] Boehm, B. W. (1981). *Software Engineering Economics*, Prentice Hall, New Jersey.

[2] Brown, S. L., Eisenhardt, K. M. (1997). The art of continuous change: Linking complexity theory and time-paced evolution in relentlessly shifting organizations, *Administrative Science Quarterly*, Vol. 42(1), pp. 1-34.

[3] Carmel E., Bird B. J. (1997). Small is beautiful: A study of packaged software development teams, *Journal of High Technology Management Research*, Vol. 8, No 1, pp. 129-148

[4] Cusumano, M. (2004). *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*, Free Press, New York.

[5] Eisenhardt, K. M. (1989). Building Theories from Case Study Research, *Academy of Management Review*, Vol. 14, No 4., pp. 532-550.

[6] Eisenhardt, K. M., Martin, J. A. (2000). Dynamic capabilities: What are they?, *Strategic Management Journal*, Vol. 21(10/11), pp.1105-1121.

[7] Littlepage, G.E. (1991). Effects of Group size and task characteristics on group performance: A test of Steiner's model. *Personality and social psychology bulletin*, 17(4), 449-456

[8] Messerschmitt, D., Szyperski, C. (2003) *Software Ecosystem*, MIT Press, London.

[9] Mowery, D. (1996). *The International Computer Software Industry: A Comparative Study of Industry Evolution and Structure*, Oxford University Press, New York.

[10] Schach, S. R. (2002). *Object-Oriented and Classical Software Engineering*, McGraw-Hill, New York.

[11] Sommerville, I. (1992). *Software Engineering*, Addison-Wesley.

[12] Strauss, A., Corbin, J. M. (1998). *Basics of Qualitative Research. Techniques and Procedures for Developing Grounded Theory*, Sage Publications.

[13] Yin, R. (2003). *Case Study Research: Design and Methods*, Sage Publications, Thousand Oaks.