Title:	Concurrent Software Development-A Survey of the State of Concurrent Engineering in Software Development
Course:	EMGT 510 NPD
Term:	Spring
Year:	1997
Author(s):	L. Harding
Report No:	P97014

Abstract: Concurrent Engineering is a popular form for improving the new product development process in hardware design This paper addresses the question, "Why has software-engineering process not benefited from CE?" This paper attempts to address the question through a survey of software developers and research into the software development process.

Concurrent Software Development—A Survey of the State of Concurrent Engineering in Software Development

L Harding

EMP-9714

MUIR LEE HARDING

INDIVIDUAL PROJECT REPORT

MAY 28, 1997

CONCURRENT SOFTWARE DEVELOPMENT

A SURVEY OF THE STATE OF CONCURRENT ENGINEERING IN SOFTWARE DEVELOPMENT

TABLE OF CONTENTS

Table of Contents i

Introduction 1

Background 1

Concurrent Engineering 1

Software Development 2

The Capability Maturity Model and the Personal Software Process 3

Methodology 3

Survey Design 3

Survey Distribution 3

Results and Discussion 4

Demographics 4

Questionaire 5

Conclusion 6

Appendix A – Survey Form ii

CONCURRENT SOFTWARE DEVELOPMENT

A SURVEY OF THE STATE OF CONCURRENT ENGINEERING IN SOFTWARE DEVELOPMENT

INTRODUCTION

Software is among the largest and fastest growing industries, with an estimated market value of $350 billion by 1998. Since its genesis in the 50s and 60s, the industry has been fairly stable with regard to process. The early description of the Waterfall Model is still widely applicable today, and is still cited as the most commonly used process framework. This is in contrast with the mechanical and electrical design industries that have shown widespread and rapid process changes over that last decade.

Concurrent Engineering (CE) is a popular form of process rejuvenation in the hardware design industry. Firms successfully implementing CE techniques realize significant improvements in quality and time to market. Improvements in these benchmark areas often directly improve the firm's financial performance.

The question thus arises, "Why has software-engineering process not benefited from CE?" This paper attempts to address the question through a survey of software developers and research into the software development process.

BACKGROUND

CONCURRENT ENGINEERING

The interpretations of 'Concurrent Engineering' are varied and rich. This paper addresses only two of those definitions:

1. Concurrent Engineering as a description of the systems and processes that facilitate the coordinated activities of a group. In this definition, it is assumed that by coordinating effort, an improvement in efficiency will be realized.

2. Concurrent Engineering as an objective which is characterized by maximizing the amount of simultaneous activity in a project. This definition assumes that any duplication of effort caused by the parallel activity is offset by gains in time to market.

In general, CE implementations are recognized as sharing some common characteristics:

1. Cross-functional, autonomous teams.

2. Project ownership.

3. Well defined requirements.

Hardware engineering processes have historically been defined based on the context. It is very difficult to define mechanical design process, and very few concise descriptions have been offered. Processes that exist to day have evolved over a long period, and generally rely heavily on the individual engineer.

SOFTWARE DEVELOPMENT

Developing software appears to be different than developing hardware – unfortunately, the importance differences have been difficult to identify. For the most part, software developers of the 90's employ the same process that developers of the 70's and 80's used. It is unusual that a relatively young industry would not show improvements that are more significant in the realm of process.

In most professions, competent work requires the disciplines use of established practices. The use of plans and procedures brings order and efficiency to any job and allows increased focus on producing superior product. Unfortunately, software professionals today often do not plan or track their work, and software quality is rarely measured.

There are several possible explanations for the relative stagnation of software development processes. Two of them are:

1. As a product of the modern era, software development processes were done right the first time. Having no historical baggage to carry forward, software developers were free to define the processes that were most efficient, and these processes have stood the test of time.

2. Software development processes are distinctly different from hardware development processes, so rapid changes in one do not correlate with changes in the other.

3. Software engineering doesn't require process to be effective.

The answer is probably a combination of all three. In the early days of the computer, programmers were constrained by the power of the machines for which that constructed code. Because of the extreme cost associate with errors in software at this stage, the processes that developed them were very closely considered. The result was the adoption of a number of formal development models. Notable among them was the Waterfall model, which is still commonly used today.

Because of the high degree of intangible deliverables in software development, it is more difficult estimate schedules and measure progress. In addition, software systems tend to dwarf their mechanical counterparts in informational degrees of freedom. The combined effect is that it is difficult to fully understand the entire process.

THE CAPABILITY MATURITY MODEL AND THE PERSONAL SOFTWARE PROCESS

One of the most promising developments in Software development process is the Personal Software Process (PSP). The PSP is a designed and measured framework that helps software engineers plan and track their work and product high-quality products. PSP can be applied to many parts of the development process, including small-program development, requirements definition, document writing, systems test, and maintenance and enhancement of large software systems. The PSP is directed toward the activities of the individual developer rather that the team or organization. It developed from a desire to project the beneficial characteristics of the Software Engineering Institute's Capability Maturity Model to small teams or individuals.

The PSP is distinguished from other SWD frameworks in that it does not require a full understanding of the global process – only the part in which the individual operates, and the *interfaces* with the global process. By reducing the scope of the problem, PSP allows individuals to focus attention on their contribution to the process, but not the process itself.

PSP implementations have so far shown moderate success in enhancing productivity, and exceptional results in increasing the reliability of estimating and scheduling. Since the PSP is new, only a small body of case-study data is available. In addition, SEI is in the process of reviewing and improving the initial guidelines based on feedback received to date.

The PSP is interesting in terms of CE because in provides the enabling infrastructure at the level of the individual engineer. By installing a fine-grained, individually maintained process-tracking framework, PSP can provide feedback about the success of CE.

METHODOLOGY

To determine the extent and character of concurrent engineering in software development, a survey was created and distributed at random via the Internet to software professionals around the world. The survey was implemented as an HTML form on the author's web site, and survey results were automatically accumulated and analyzed in real-time.

SURVEY DESIGN

In designing a survey for Internet distribution, the focus was on compactness. The feeling was that a shorter, more direct survey would elicit a much larger number of responses. In that light, the survey was limited to five general questions regarding the collaboration between functional groups in the production of important deliverables. A more rigorous and robust survey would be an interesting follow-up to this effort.

SURVEY DISTRIBUTION

There were several interesting consequences of the electronic distribution of the survey. First among them was that the distribution was not restricted to a known set of candidates. The respondents could have been anyone from a 10-year-old to seasoned software professionals. Because survey submissions were anonymous, the author did not attempt to filter the responses. However, two precautions were taken to minimize the 'noise' associated with bogus responses.

First, no general announcement of the survey was posted to UUNET. Rather, notification of the survey was sent to a number of moderated computer science and programming email discussion lists under the assumption that the members of these lists are actively involved in the development of software.

A second consequence of the electronic posting was the large number of respondents, and short turn-around time. Within 48 hours of the initial survey announcements, 115 responses had been received. At the time of the writing of this paper, survey results are continuing to arrive at a rate of about two per day, and the total survey tally has reached 171 responses.

The major third consequence of the electronic distribution was the ability to determine the source of the survey response. By recording the sending site, and requesting a 'cookie' from the users browser, it was possible to reduce the possibility of multiple submissions from one person. In addition, since the survey tally was automated via Perl CGI scripts, the results could be queried in real-time. To avoid influencing the survey results, tally information was not provided to survey participants.

RESULTS AND DISCUSSION

DEMOGRAPHICS

Of the 171 survey responses, 23 were incomplete or duplicates. The demographic data for the remaining 148 responses are shown in the figures below. The majority of respondents (55%) appear to work for medium or large firms (50+ employees) with small development organizations (51% with 1-5 SWD personnel). In addition, the majority of the respondents described their role in SWD as Engineering (65%), and their firms were located in the United States (85%).

These results are not in good agreement with the general demographics of the software development community. Of the many possible explanations for this, it appears likely that the survey sample was not randomly distributed. In future surveys, it might be desirable to find a more 'random' method of survey notification.

At the time of this writing, no attempt has been made to examine the fine-grained relationships between the demographics and the results of the questionnaire. It would be desirable to do so in a subsequent work. In particular, it would be desirable to investigate whether the various functional groups have different views on the use of CE.

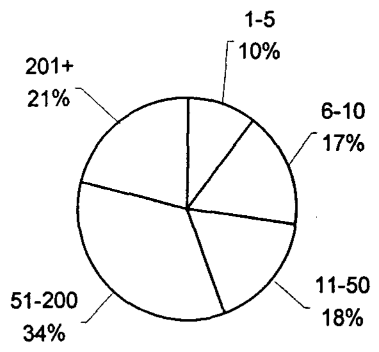

Figure 1: Number of employees in the firm

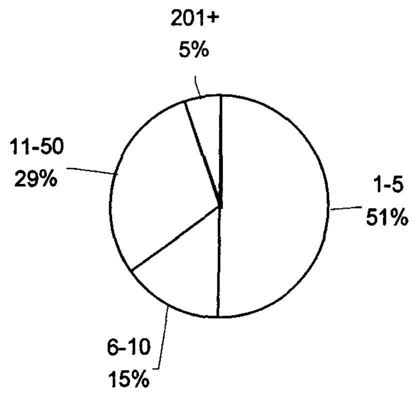

Figure 2: Number of employees in SWD

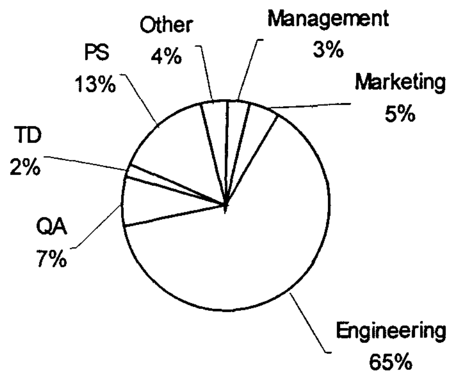

Figure 3: Responant's role in SWD

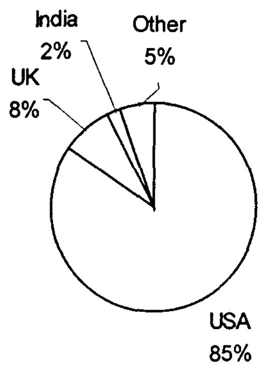

Figure 4: Firm's location

QUESTIONAIRE

The survey results indicate that there is a belief that significant collaboration occurring across many functions of software development organizations. The most common interaction appears to be between marketing and engineering in the production of specifications.

The least common area of interaction is between quality assurance and product support in the development of testing plans. This is disturbing, but may be explained by the demographic data. A large number of respondents were not involved in the production of consumer software, and therefor may not have had a Product Support role in their organization. This flaw in the survey could have been avoided by having respondents identify the roles present in their firm/organization.

If it is true that QA and PS roles do not often collaborate in developing testing strategy, then it is an area for potential improvement. As direct reception points for the 'Voice Of the Customer,' (VOC) Product Support personnel may provide valuable, real-time awareness of customer requirements. This information is directly applicable to the development of test strategies since it will effect the usage of the product once delivered. It is also relevant to the development of specification for subsequent products.

Table 1: Summary of the survey results

Statement	Level of Agreement
Marketing and Engineering work together to develop specifications for new products.	1.9
Quality Assurance participates in the specification of new products.	3.2
Product (Customer) Support participates in the specification of new products.	3.3
Product (Customer) Support participates in the development of Quality Plans.	3.7
Software design activities occur in parallel with specification activities.	2.9

CONCLUSION

At its root, Concurrent Engineering is quest for efficiency – get it done quicker, cheaper and with better quality. Interestingly, software development has been driven by these factors almost since its inception. Until recently, limited computing power required software developers to optimize their work for speed, size and resource requirements.

To achieve good results with CE, software development organizations need to raise awareness of the new bottleneck -- the company's ability to execute. As computer hardware was in the past decades, the corporate infrastructure is the 90s. A good software development process, and the systems to support it, is as important as good product specification, design and marketing. This is because the scope of the organization allows alignment of activities that have heretofore been considered independent.

Management, Marketing, Engineering, Quality Assurance and Product Support have significant interest in all stages of development, and must synchronize their activities. By doing so the organization can maximize its opportunities to catch errors while they are 'cheap,' and accelerate schedule and defect reductions.

BIBLIOGRAPHY

1) M. C. Paulk at al., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Adison Wesely, Reading, MA, 1995.

2) W. S. Humphrey, *Introduction to the Personal Software Process.* Addison-Wesley, Reading, MA, 1997

3) P. Ferguson, W. S. Humphrey, S. Khajenoori, S. Macke & A. Matvya, *Results of Applying the Personal Software Process*, Computer, pp.24-31, May 1997.

4) R. Beltramini, *Concurrent Engineering: Information Acquisition in New Product Development*, Intl. J. Technology Management, Vol. 11, No. 1/2, 1996.

5) B. Boehm & P. Bose, *Humans and Process Frameworks: Some Critical Process Elements.*

6) J. D. Blackburn, G. Hoedemaker & L. N. Van Wassenhove, *Concurrent Software Engineering: Prospects and Pitfalls*, IEEE Transactions on Engineering Management, Vol. 43, No. 2, May 1996.

APPENDIX