Determination of Concurrent Software Engineering Use in the United States

by

David L. Richter

ISBN: 1-58112-065-6

DISSERTATION.COM

1999

Copyright © 1999 David L. Richter
All rights reserved.

published by
Dissertation.com
USA • 1999

ISBN: 1-58112-065-6

www.Dissertation.com/library/1120656a.htm

NORTH CENTRAL UNIVERSITY

DETERMINATION OF CONCURRENT SOFTWARE ENGINEERING USE IN THE UNITED STATES

A Dissertation submitted to the graduate faculty of the

Department of Business and Management

In candidacy for the degree of

DOCTOR OF PHILOSOPHY

by

David L. Richter

Prescott, Arizona
March 1999

Copyright 1999

David L. Richter

DISSERTATION ABSTRACT

Name: David L. Richter

Degree: Doctor of Philosophy

Major Field: Technology Management

Awarded: March, 1999

Institution: North Central University

Location: Prescott, Arizona

Title of Dissertation: DETERMINATION OF CONCURRENT SOFTWARE ENGINEERING USE IN THE UNITED STATES

Scope of Study: This dissertation summarizes the current use of concurrent software engineering (CSE) by information technology (IT) organizations in the United States and its effectiveness in improving software delivery time, quality, and cost. From a total population of 7,173 IT organizations, a one-third sample of 2,391 were surveyed. A net valid response of 142 organizations was received, which represents a valid return rate of 6.2 percent. The responses were then analyzed against software development time, quality, and cost metrics according to the software development methodologies used.

Findings and Conclusions: This study shows the extent to which pure CSE and CSE in combination with the traditional system development life cycle (SDLC) are used in the United States. There are strong indications that CSE improves software development time and cost, but this could not be statistically proven from the data. There is no indication that CSE improves software quality.

Chair's Approval: _____ Date: _____ 1999

APPROVAL

We, the undersigned, certify we have read this dissertation and approve it as adequate in scope and quality for the degree of Doctor of Philosophy.

Learner: David L. Richter

Title of Dissertation: DETERMINATION OF CONCURRENT SOFTWARE ENGINEERING USE IN THE UNITED STATES

Dissertation Committee:

Judith L Forbes — 8 April 1999

Chair: J. Forbes, Ph.D. — Date

R. Gray — 4/8/99

Member: R. Gray, Ph.D. — Date

Richard S. Yellen — April 5, 1999

Member: R. Yellen, Ph.D. — Date

TABLE OF CONTENTS

TABLE OF CONTENTS

(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

Page

LIST OF FIGURES
(Continued)

Page

LIST OF TABLES

Page

Chapter I

INTRODUCTION

Concurrent engineering (CE) is a design and manufacturing technique that increases product quality, decreases manufacturing costs, and decreases product development time. This is accomplished by organizing a product's life-cycle experts, including design, process, production, and service engineers, into a single design team. Since 70 percent of a product's quality and costs are fixed in the design phase, the multidisciplinary team approach directly addresses improved product quality and decreased manufacturing costs [Kusiak and Park, 1990]. This is because quality, manufacturability, and serviceability are included in the concurrent engineering design process. Decreased development time is achieved by performing work in parallel, or concurrently. Every task is started at the earliest possible moment, which mandates concurrent communication to all members of the project to minimize rework.

The concurrent engineering method is in contrast to the historical design and development approach where a product's development life-cycle is performed serially. In the traditional method, each expert performs his or her function, then passes the product along to the next function, where product developers are expected to follow the design although they have little input to that design.

Because of the clear and proven advantages of concurrent engineering, CE is now a standard product development approach for manufacturing in Japan, Europe, and the United States.

The concepts of concurrent engineering can also be applied to software development. Although the processes for current software development are essentially the same as for manufactured products, the term "concurrent software engineering (CSE)" is used for this activity. Whereas concurrent engineering is now a standard technique in most Western and Japanese manufacturing industries, concurrent software engineering has not been fully accepted by U.S. information technology (IT) organizations.

Problem Statement

This research answers the following questions concerning concurrent software engineering:

1. To what extent is CSE used in the United States?
2. What are the benefits of CSE in terms of quality, cost, and schedule, if any?
3. What are the drawbacks of CSE, if any?

The answers to these questions provide general conclusions as to the current state of CSE deployment in the United States and the potential CSE benefits that can be achieved. Important information on CSE limitations and drawbacks are also identified. Finally, the following is tested although not statistically proven:

Research hypothesis - The use of concurrent software engineering reduces software development time, decreases software development costs, and improves software quality.

The information determined by this research is important because of the enormous size of the problem. The current U.S. market for software products and services exceeds $200 billion per year [Baker, McWilliams, and Kripalani, 1997]. At the same time, the general state

of software quality and development in the United States is poor. For every six new large-scale systems that are put into operation, two are canceled. The average software project overshoots its schedule by half, and three-quarters of all large systems deployed are "operating failures" that do not function as intended or are not used at all [Gibbs, 1994].

A key reason for the excessive costs, schedule overruns, and poor quality of U.S. software is that the software development process still closely resembles craft production rather than true engineering product development. To exacerbate the U.S. software quality, cost, and project schedule problems, there is a chronic shortage of IT engineers. There are 1,200,000 IT jobs in the United States, and there are over 190,000 IT job vacancies [Cone, 1997; King, 1997b]. At the same time, annual computer and information science bachelor degree graduates declined from 42,000 per year in 1986 to 24,200 per year in 1994 [Cone, 1997]. This has put significant upward pressure on IT salaries, adding an additional $15 billion in annual U.S. software development costs. Additionally, King (1997b) projects that U.S. business will lose an additional $500 billion a year in revenue because of uncompleted information systems projects. King (1997b) further projects that the IT labor shortage will persist beyond 2003 while surveys by Fabris (1998) indicate that 80 percent of large and midsized companies expect to add to their IT workforces each year.

The IT engineer labor shortage has prompted major IT companies to push Congress for an increase in the annual number of high-tech foreign worker visas from 65,000 per year to 115,000 per year [Doyle, 1998]. The net result of this labor shortage is that less-experienced engineers are developing software, and they frequently are not knowledgeable in their

application domains [Curtis, Krasner, and Iscoe, 1988]. These individuals can usually program, but they often do not understand the problem to be solved.

In conclusion, by determining the extent to which CSE is used by U.S. IT organizations, along with CSE benefits and limitations, it is possible to better understand the quantitative and qualitative impacts CSE can have. Given the enormous size of the software development problem and expense, even a one percent increase in software development productivity through the use of this proven technique could result in billions of dollars per year in savings as well as improved software quality in the United States.

Definition of Terms

There are two key terms used in this research, and they are defined as follows:

1. **Traditional Software Development Life Cycle (SDLC)**: In the traditional SDLC, software engineering projects go through sequential phases moving from one phase to the next only after the requirements of the preceding phase are satisfied. For example, the Requirement Specification, Functional Specification, Design, Coding, Testing, and Implementation Phases are performed one after the other for each project.

2. **Concurrent Software Engineering (CSE)**: CSE involves a multi-disciplinary team that performs software design, coding, and testing simultaneously rather than sequentially. The team assembled at the beginning of the project consists of users, software designers, programmers, computer operations staff, software quality assurance staff, technical writers, and others as appropriate. Programming and testing begin concurrently at the earliest possible moment rather than after design completion.

Brief Overview of the Literature

There is relatively little literature focused on CSE, but CSE is an almost one hundred percent subset of concurrent engineering. Thus, CE techniques and concepts relate directly to CSE.

Concurrent engineering is defined by RACE (1998), University of Tennessee (1998), Shina (1991), and Blackburn, Hoedemaker, and Van Wassenhove (1996) as an engineering approach and set of techniques that focus on product quality while simultaneous engineering work is done at the earliest possible moment. The essence of CE is the concurrent rather than serial execution of product development phases.

A key element of concurrent engineering is the consideration during the design phase of factors associated with the life-cycle of the product, including manufacturability, assembly, testing, maintenance, reliability, and quality [O'Grady and Young, 1991]. A second key element of CE is that there must be a team approach and that the team members should be located together [Ho, Poh, and Auzare, 1995; Calvano, 1998; Institute of Management and Administration, 1997].

In summary, the theoretical aspects and methods of CSE deployment are clearly defined, and there is little or no controversy associated with these theoretical methods. The literature supporting the theory is presented in detail in Chapter II. There is, however, little published information on the actual use of CSE, and there appears to be no published study reports focused directly on CSE's use and benefits. That is a knowledge gap this research hopes to at least partially fill.

Methods

To reiterate, the purpose of this research is to answer the following questions concerning concurrent software engineering:

1. To what extent is CSE used in the United States?
2. What are the benefits of CSE in terms of quality, cost, and schedule, if any?
3. What are the drawbacks of CSE, if any?

The research method is a survey (Appendix B) utilizing a subset of the 7,173 element U.S. IT organization commercial database maintained by Applied Computer Research, Inc., in Phoenix, Arizona. The survey sample is 2,391, which is one-third the population.

The survey participants were guaranteed that their identities and the identities of their organizations will remain strictly confidential and be used only for the purposes of this research. The participants were offered an electronic summary of the research results should they wish to provide an e-mail address.

Limitations of the Study

The results of this research are totally dependent upon the sample population that chooses to return the questionnaire. An additional limitation is that the survey results are dependent upon the accuracy of the information provided by the sample population as there is no direct researcher observation to verify accuracy. A final limitation is that there was no way to ensure a high survey return rate. The only actions available to promote a high return rate were to make the survey task easy for the subject and to make the questionnaire as brief and direct as possible.

Research Findings

There are indications that CSE provides a mechanism to improve the quality, cost, and schedule of software development projects, but statistical proof of this hypothesis was not demonstrated by this research. However, the results of this research can be used as a basis for further study to determine why CSE has not achieved the demonstrable quality, cost, and schedule benefits that concurrent engineering achieves in manufacturing.

Chapter II

REVIEW OF THE LITERATURE

There are three primary literature areas that relate to concurrent software engineering use in the United States. The first is the current size and condition of the U.S. software development situation. The second area documents the theoretical advantages achievable using concurrent engineering for product development. The final relevant literature is concerned with actual CSE use in the U.S.

It should be noted that most literature in this area relates to generic concurrent engineering. There are relatively few references directed specifically to CSE. However, CSE is an almost one hundred percent subset of general concurrent engineering. Thus concurrent engineering literature directly relates to CSE in almost every case.

U.S. Software Development Situation

The current U.S. market for software products and services exceeds $200 billion per year [Baker, McWilliams, and Kripalani, 1997], and the IT professional services market in the U.S. stands at $25 billion per year with an annual growth rate of 37 percent [King, 1997a]. This coincides with the work of Dutta and Van Wassenhove (1997) showing that the amount of software now required by businesses and products is doubling every two to three years. At the same time, Gibbs (1994) has provided significant examples showing the overall quality of U.S. software is poor, projects are normally over budget, and software systems are delivered behind schedule. According to IBM's Consulting Group study of 24 leading companies, 55 percent of

the software projects cost more than expected, 68 percent overran their schedules, and 88 percent had to be substantially redesigned. Also, ninety percent of American programmers do not even count the number of mistakes they find, and less than ten percent of American companies measure programmer productivity [Gibbs, 1994]. One study has shown that for every 100 lines of code that are written, 95 must be rewritten, which is an overhead factor of 20 [Dean, 1994].

Both Van Genuchten (1991) and Blackburn, Hoedemaker, and Van Wassenhove (1996) state that a key reason for the excessive costs, schedule overruns, and poor quality of U.S. software is that software development does not follow a true engineering product development processes. Instead, software development still closely resembles craft production. Dean (1994) observes that the general acceptance of poor software quality is exemplified by the fact that the U.S. software industry is the only industry legally indemnified from producing a quality product. This is despite the contentions that quality is the most effective factor in competing for customers [Morup, 1992] and that the cost of quality is essentially zero [Crosby, 1979].

The main objective of any manufacturing company is to bring new products market sooner than the competition with lower cost and better quality [Sullivan, 1986], yet most commercial software fails to gain adequate market share [Dean, 1998]. There have been heavy investments in computer assisted software engineering (CASE) to improve software engineering, but Yeh (1992) observes that these investment have generally produced disappointing results because the old, sequential engineering techniques have been applied.

Concurrent Engineering for Product Development

One of the available techniques to improve software quality, reduce costs, and improve delivery schedules is concurrent engineering (CE), which is the simultaneous execution of product development phases at the earliest possible moment [RACE, 1998; University of Tennessee, 1998; Shina, 1991; and Blackburn, Hoedemaker, and Van Wassenhove, 1996].

The techniques supporting CE include Quality Function Deployment (QFD), Failure Mode and Effects Analysis (FMEA), Design for Manufacturing and Assembly (DFMA), and designing for cost. These are described in detail by RACE (1998), Dean (1992), Dean and Unal (1992), Akao (1990), Takeuchi and Nonaka (1986), Taguchi (1986), and Thackeray and van Treeck (1990).

Concurrent engineering is the consideration during the design phase of factors associated with the life-cycle of the product, including manufacturability, assembly, testing, maintenance, reliability, and quality [O'Grady and Young, 1991]. Since 90 percent of a product's quality and 70 percent of its life-cycle costs are determined in the design phase [Kusiak and Park, 1990], it is crucial that there is heavy customer input in the design phase [Keil and Carmel, 1995].

Another key element of CE is that there must be a team approach and that the team members should be located together [Ho, Poh, and Auzare, 1995; Calvano, 1998; The Institute of Management and Administration, 1997]. All the engineers from the development life cycle disciplines, including design, manufacturing, service, and support, must be on the concurrent engineering team in order to properly influence the product's design. The size of the team can

also be a factor in a project's success as smaller teams tend to be more effective [Blackburn, Scudder, and Van Wassenhove, 1996].

In CE, there is a concurrency of activities, but there is also consensus among Blackburn, Hoedemaker, and Van Wassenhove (1996), Curtis, Krasner, and Iscoe (1988), and Loch and Terwiesch (1997) that there must be a concurrency of effective information. In order for development activities to proceed in parallel, there must be information concurrently available to all team members in order to minimize rework.

Concurrent Software Engineering Use in the U.S.

Despite the general use of CE by U.S. manufacturers, there apparently is less acceptance of concurrent software engineering by U.S. IT organizations. The literature contains references to the theoretical advantages and limitations of CSE by Blackburn, Hoedemaker, and Van Wassenhove (1996). Blackburn and Scudder (1996), Dean (1998), Hoedemaker, Blackburn, and Van Wassenhove (1995), and Thackeray and van Treeck (1990), but there are no published reports that actually measure the results of CSE either in case studies or surveys.

Summary

The current literature clearly shows that the general state of software development in the United States is poor. Software project schedule delays, cost overruns, and poor quality are common. The literature further indicates the theoretical and practical advantages of concurrent engineering that specifically improve schedule, cost, and quality metrics. There are,

however, no published reports that demonstrate how concurrent software engineering actually improves software schedules, costs, and quality in the United States. The purpose of this research is to begin to fill that gap.

Chapter III

METHODS

Overview

Concurrent engineering has become standard practice for manufacturing in advanced industrial societies. According to Akao (1990), companies that have adopted concurrent engineering have experienced a one-third to one-half reduction in product development time. However, concurrent software engineering (CSE) for software development has not been adopted to the same extent as concurrent engineering for product manufacturing, and there have been no previous, definitive studies that measure concurrent software engineering advantages.

The purpose of this research is to determine the state of CSE deployment in the United States along with CSE's benefits and limitations. This supports the specific research hypothesis: The use of concurrent software engineering reduces software development time, decreases software development costs, and improves software quality.

Research Expectations

This research answers the following questions concerning CSE:

1. To what extent is CSE used in the United States?
2. What are the benefits of CSE in terms of quality, cost, and schedule, if any?
3. What are the drawbacks of CSE, if any?