THE IMPACT OF ENVIRONMENTAL EVOLUTION ON REQUIREMENTS CHANGES

by
Vivek Nanda

School of Computer Science
McGill University, Montréal, CANADA
October 1996

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

ISBN 0-612-19839-1

Abstract

An evolving environment can have such an impact on the requirements of an existing software system that it can render the system obsolete. We have experienced precisely such a loss in our research environment. The Congruence Evaluation System was an innovative proof of concept (CES POC) system, developed over two years with three person-years of effort. It was part of an environment which was constantly evolving, and this change in the environment dictated the fitness of the CES POC system in the environment. The system served excellently for the basic research goal of discovery and proof, but failed seriously in the long term goal of evolvability of the entire suite of tools being built concurrently by several researchers in the team. In this thesis, we describe a case study of requirements changes in an evolving environment involved in the development of unprecedented systems. In particular, we identify requirements-oriented factors that caused the demise of an innovative system in such an environment. The thesis also describes some lessons learnt from this experience.

Résumé

Un environnement en évolution peut avoir un tel impact sur les spécifications d'un système existant que cela peut rendre le système désuet. Nous avons subi une telle perte dans notre environnement de recherche: le système d'évaluation de la congruité était un système expérimental innovateur qui fut développé sur une période de deux ans par un effort de trois années-personnes. Il faisait partie d'un environnement qui évoluait constamment, et cette évolution a affecté la congruité du système par rapport à son environnement. Il a très bien servi les objectifs de découverte et de preuve, mais ne s'est pas prêté à une évolution compatible avec les différents outils construits par plusieurs chercheurs de l'équipe. Dans cette thèse, nous décrivons l'étude d'un cas de changements de spécifications dans un environnement en évolution où l'on développe des systèmes inédits. En particulier, nous identifions des facteurs associés aux spécifications qui ont causé la perte d'un système innovateur dans un tel environnement. La thèse décrit aussi les leçons qui ont été tirées de cette expérience.

Acknowledgements

I would like to thank Prof. Nazim H. Madhavji, my supervisor, for his support, encouragement and advice during my research. I regard my learning experience under his guidance as immensely helpful, especially what I learnt from his attention for detail in research and mastery of the nuances of the English language (when reviewing my writings!).

I have enjoyed being part of the software engineering group. I would like to thank: Imran for his help whenever I was stuck during my research (which was quite often!) and for our frequent trips to Bombay Maharajah and Basha; Dirk for his prompt attention when I had problems with my system account and Won for allowing me to reuse his user-interface library!

Special thanks to the administrative staff: Lorraine, Franca, Josie, Lise and others for dealing with all my inquiries in an efficient way.

I would also like to thank my friends: Simran for making me feel at home away from home; Pierre and Pung for translating my abstract to French; and all my other friends in Montréal.

Last, but not least, I would like to thank my parents for their constant encouragement and morale boosting advice and to them I dedicate my thesis.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Historically, the process of software development has played an important role in the field of software engineering. The software development process comprises software engineering activities, including technical and managerial ones, that are carried out in the production of software. The scope of these activities includes determination and specification of system software requirements; analysis and management of risk; software prototyping; design; implementation; verification and validation; software quality control and assurance; integration of components; documentation; management of software configurations and versions; management of data and evolution of software [17].

Many current software development methods are still closely related to the well-known classical Software Development Life Cycle (SDLC), also widely known as the *waterfall* model [31]. While much work in recent years has focused on improving these software development processes (e.g., the spiral model [4]) and thus also software quality, a critical issue, that software must be adaptable to changes in user requirements or environmental changes is still considered problematic. This problem domain is better known as the *software maintenance problem.*

Software systems used in the academia or the industry are continually modified to adapt to changing user needs. The inevitability of software evolution and the intrinsic nature of its causes has been recognised for many years [40, 41]. Even if a system is totally reliable, completely meets user requirements at a given time, and is well structured, it might still be required to update the system due to *environmental* problems, v.i.z., environment change, new hardware, difficulty in integrating new tools with existing suite of tools [23]. Software maintenance dominates the software life-cycle. In many organizations, software maintenance activities consume three-fourths of the total life-cycle expenditures and over one-half of the data processing personnel resources [19].

Taking into account the marked difference between the production (industry) environment and the research environment, it is important to discriminate requirements changes between these two different environments. By discriminating in this way, we are able to have a deeper understanding of: the role of software systems in the two environments; the processes that should (or should not) be used in each case; the methods tools and techniques that are suited to the two environments; and other related matter.

Prior to such discrimination, we define some key terms so that these can be referred to subsequently:

1. *Software maintenance* may be defined as the modification of a software product after delivery, to correct faults, improve performance or other attributes, or to adapt the product to a changed environment [2].

2. *Software evolution* consists of the activities required to keep a software system operational and

responsive after it is accepted and placed into production. It means a continuous change from a lesser, simpler, or worse state to a higher or better state [14].

3. *Software maintainability* is defined as the ease with which a software system can be corrected when errors or deficiencies occur, and can be expanded or contracted to satisfy new requirements [19].

4. *Software engineering environment* means a collection of tools, practices, and working conditions supporting software engineering efforts [6].

5. *Functional requirements* define the actual 'functions' which must be implemented in the system [21].

6. *Non-functional requirements* define all other kinds of requirements the system should satisfy, such as performance specifications, memory requirements, development platforms, user friendliness, changeability etc. [21].

1.1 Production and Research Environments

It is well known that in a *production* environment, a software system once built generally needs to be evolved for longevity. A system may evolve due to new functional and non-functional requirements. For instance, a planned new version of the current system may require new features to be added to the current version and these added functionalities (new functional requirements) would require changes be to be made in the current system. Similarly, if the performance of the existing system has degraded due to changes in hardware or system software support (i.e., changes in non-functional requirements) then appropriate changes could be required in the existing system to make it effective. Lehman and Belady cite an operating system which increased from 3,682 modules to 4,800 modules over four major release cycles! [32, 33, 34]. Other possible causes for the

software maintenance problem can be changes in the system context, i.e. the overall environment in which the system is to fit, changes in the objectbases accessed by the system, changes in programming language requirements, etc. There are various reasons why these new functional and non-functional requirements crop up: some of these are, increased competition and market pressures which goads the organizations to further improve their product, new business opportunities which can be exploited by modifying/improving the product, customer satisfaction issues, organisation mergers which put immense pressure to harmonise existing systems, etc.

In contrast to this, in certain *research* environments, evolution of a software system may not always follow a planned path seen in many production environments. Typically, such a system is a 'high-end' research system which is considerably innovative and risky to build. Such a research project has little theory and practice backing the research. The purpose of building such a system is to test the theory, and *not* to build a production version system for use in any application domain. Because of the high content of originality in such work, the researcher does not have exemplary systems which could be relied upon to build new concepts. That is, not assuming the magnitude and impact of such a revolution, the research progress is *revolutionary* rather than *evolutionary*. Consequently, the researchers do not know much about the domain in which the system would be used. For example, the idea of syntax directed programming environments as first proposed by Hansen [60] in 1971 was a highly innovative idea at that time since there was no precedent to such work. This idea, after almost a decade formed the basis of the development of the Cornell Program Synthesizer [61] which has had considerable impact on integrated tools for software development. Another example is Osterweil's proposal [62] that software processes be described by *programming* them much the same way as computer applications are programmed.

Since its proposal, this idea has taken different paths of research amongst many researchers and practitioners (such as development of enactment engines and process modelling).

Generally, a system being built in a research setting is on a path leading to a short term goal of *proof of concept* (POC) system and not to a long-lived system. Thus, designing for future evolutionary changes is not generally of prime concern. Often the development techniques used include hard-coding the decisions iteratively and subsequently testing the validity of the POC. Also, the learning curve is minimised, if not eliminated totally, on such aspects as the programming language and the software and hardware platforms used. Once the POC system is operational, frequent changes might be needed to the system for various reasons. For instance, to deal with new core features, new computational algorithms may have to be implemented; system changes may be necessitated by an improved understanding of the system domain concepts, etc. Also, in such a research environment, the original researcher or student (hereafter called 'champion') who built the POC system would in many cases not be present to help make these changes. Reasons include transfer to industrial jobs, or other opportunities once their goal of building a POC system has been achieved. As a result, the POC system in such a research environment, once built, is often scrapped and subsequently re-engineered from scratch to meet the long term goal of on-going evolvability.

1.2 Research Context and Problem Definition

In our research environment, at the Software Engineering Lab at McGill, we have experienced precisely such a loss. The Congruence Evaluation System (CES) [9, 10] was one such POC system championed by a research student. It served excellently for the basic research goal of discov-

ery and proof, but failed seriously in the long term goal of evolvability of the suite of research tools being built concurrently by the entire research team at McGill. The key decisions in the CES POC system were hard-coded and the development platform (FoxPro on MS-DOS) was incompatible with that used for the rest of the suite of tools (UNIX). There seemed no other alternative at that time but to follow this design strategy, taking into account the work skills of the champion of the CES POC system, the time available to complete the POC system, and the resources available to support the project. The CES POC system took two years to build, with three person-years of effort, from concept understanding to system validation. A considerable amount of empirical research had been carried out [3] which lead to the basic hypothesis and to the idea of building the CES POC system; it was not a pre-planned goal, from the outset, to build the POC system.

Once we scrapped the CES POC system, we decided to rebuild the system from scratch (in C++), reusing concepts and algorithms as appropriate from the CES POC system. No longer were the basic functional requirements of the CES POC system the *only* key requirements. Suddenly, the *environmental* requirements, at the time when the future of the CES POC system was being decided, had a severe grip on the fate of the system. It just had to be decommissioned. A new one had to be built with the long term evolvability of the overall suite of tools (four systems being built in the suite) as a primary goal. The new CES system took approximately one year with one person year of effort to build. It provides all the services provided by the POC system, but is designed to be highly user-programmable, user-friendly and is fully compatible with the suite of tools in the environment. Based on this new version, there are *evolutionary* plans to enhance the system in conjunction with those of the sister systems in the environment.

From this experience, however, emerges the following key question which needs answering:

- "How does the environmental evolution affect the survivability of a given software system?"

This is an important question because it deals with a difficult issue of software development that is often overlooked at the outset. An adequate answer would help in increasing our collective experience which can perhaps be used to better plan research systems. The aim of this thesis, then, is to address the question described above.

While there is a healthy body of literature on software evolution, e.g. planning for software evolution [6, 18, 38], risk in evolving software systems [8,13, 29], software maintenance activities and management [14, 19], current software maintenance practices [15, 16, 23] and other related matters reviewed in the next chapter, there is not much information contributing towards the answers to the described questions.

Because of limited previous work on this topic, these contributions attain added significance. It is important to note the types of issues that are *not* the focus of attention in this thesis. We are *not* discussing related and perhaps equally important issues of software maintenance process models [1, 15, 23], different types of software maintenance [20, 21, 37], metrics for software maintenance [35, 36], how to perform design maintenance [6, 8], how to specify software requirements [7, 22], and similar issues. There has been relatively more work carried out in this arena. We are attempting to shed light on certain equally critical maintenance issues on which significant work has not been carried out in the past and work in this sphere is still in its infancy [39]. Our concern is high risk, highly innovative research environments. We investigate why serious changes occur in systems; the type of requirement changes which affect system survival and utility; and, the type of

requirements that are imposed by environmental changes. Thus, our focus is *not* on the detailed *development* of the system (i.e., design, coding, inspections, testing, etc.) as much as it is on the *requirements* for the system. Thus, one can narrow down the scope of our work to *requirements change*.

1.3 Research Contributions

The key contributions of this thesis are:

1. an insight into how environmental evolution affects the survivability of a software system, and
2. some lessons learnt which can be considered in the development of POC systems in a class of research environments.

1.4 Organization of the thesis

The next chapter describes the background material. Chapter three describes the CES POC system and analyses its revolutionary properties. Chapter four analyses the benefits of the POC system and the problems that beset it. Chapter five describes the impact of environmental changes on the CES POC system requirements. Chapter six describes the impact of environmental changes on the requirements for the re-implemented system. Chapter seven lists the lessons learnt from this exercise. Finally, Chapter eight concludes the thesis and describes the future work.

Chapter 2

Background on Requirements Changes

Requirements changes can be regarded, in part, as *drivers* of software evolution. As discussed in chapter one (and later elaborated in chapter four), majority of the problems in the CES POC system cropped up due to changes in the *environment*, resulting in *changes* in system *requirements*. It is therefore imperative that along with analysing these requirements changes, one must also study the *environment characteristics* which *influenced* these requirements changes.

These relationships are depicted in Figure 2.1.

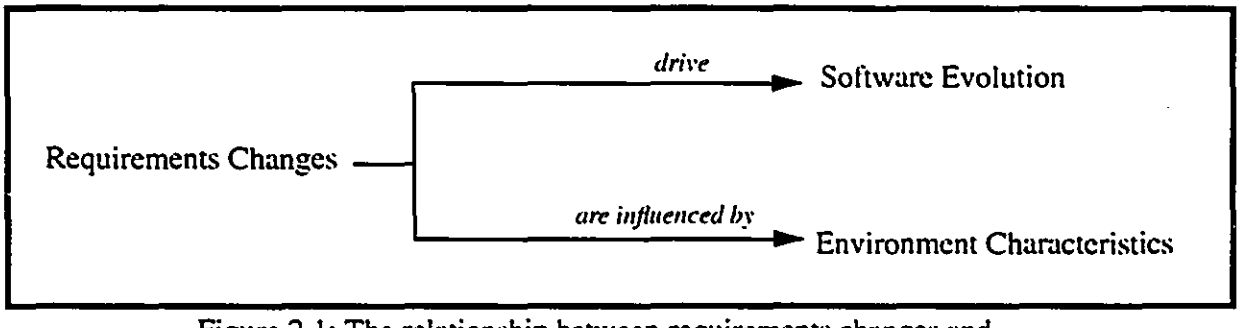


Figure 2.1: The relationship between requirements changes and software evolution and requirements changes and environment characteristics

In this chapter, we review the literature related to requirements changes and it is divided into three sections. The first section reviews the literature on software evolution and how requirements

change drive such evolution. The second section describes the characteristics of research and production environments. The final section summarises the key-points from both the preceeding sections.

2.1 Software Evolution

Arthur [14] categorizes software evolution activities into: correcting defects (maintenance), enhancing software functionality (evolution) and improving the quality of existing software (maintenance). Also, the maintenance activities have traditionally been categorized as: *corrective maintenance, adaptive maintenance and perfective maintenance* [6, 19, 20, 21]. Corrective maintenance focuses on fixing defects. Defects refer to the system not performing as originally intended, or as specified in the requirements. Perfective maintenance includes all efforts to improve the quality of the software. These activities include restructuring code, creating and updating documentation, improving reliability or efficiency or any other quality factors. Adaptive maintenance includes all work related to changing how the software functions. It includes system changes, additions, insertions, deletions, modifications, extensions, and enhancements to meet the evolving needs of the user and the environment in which the system must operate. Lamb [6] elaborates that adaptive maintenance includes changing the system to match changes in the environment, such as moving it to a new hardware or a new operating system.

The maintenance process for all the above three mentioned maintenance activities begins with a *change request.* A change request is a vehicle for recording information about a system defect, requested enhancement, or quality improvement. It should be noted that the change request for each of the three different types of maintenance activities are different. Change requests essen-

tially fall into two categories: *problem reports* and *enhancement requests*. Problem reports describe defects and system actions (in the present system) that are out of line with the system's requirements and, thus, they drive corrective maintenance activities. On the other hand, enhancement requests describe a change in system requirements (functional or non-functional) and quality requirements and thus they drive adaptive maintenance and perfective maintenance activities. Specifically, changes in system requirements drive adaptive maintenance activities while changes in quality requirements drive perfective maintenance activities. The process of uniquely identifying, describing and tracking the status of each change request is known as *change management*.

Lehman and Belady pioneered the study of the evolution of software [34,67]. They studied a variety of systems over a period of time and observed several patterns and trends from which they developed the *Laws of Program Evolution [41]*:

☛ *Law: All useful programs undergo continuing change*. They state that programs undergo a continual change in response to changing system requirements. For example, Basili and Turner [66] studied the SIMP-T compiler which evolved from 3,404 statements and 4 modules to 6,350 statements and 37 modules over five major release cycles!

☛ *Law: Over time, programs exhibit increasing entropy*. That is, as a program evolves, its structure degrades and its size increases, resulting in increased complexity. For example, a commercial software metric analyzer, PC-METRIC, increased from 995 executable statements, 22 procedures, and a cyclomatic complexity of 118 to 2291 executable statements, 43 procedures, and a cyclomatic complexity of 298, over a period of release cycles stretching from 1987 through 1989 [68].

☛ Law: *Program evolution exhibits statistically smooth growth.* The system and its metasystem,

the project organisation developing it, constitute an organism constrained by conservation laws. These may be locally overcome, but they direct, constrain and control the long-term growth and development patterns and rates. Some of the constraints identified by Lehman and Parr from the analysis of the evolution of the OS/360 system were: reduction in level of demand for enhancements and reduction in resources allocated for system development [39].

Osborne [23] divides software maintenance problems into five distinct categories: (i) software quality, (ii) environment, (iii) management, (iv) users and (v) personnel, as shown in Table 2.1. The maintenance problems highlighted in Table 2.1 are those which were encountered in the CES POC system. He lists three major causes for these software maintenance problems as inadequate attention to: (1) requirements specifications, (ii) quality assurance and (iii)configuration management.

Software Maintenance Problems	
Software Quality	Program Quality
	Programming languages used
	Lack of common data definitions
	increasing inventory
	excessive resource requirements
Environment	evolving/change
	new hardware
	difficulty integrating tools

Table 2.1 Software Maintenance Problems

Software Maintenance Problems	
Management	growth
	change control/configuration management
	maintenance techniques/procedures
	maintenance tool usage
	standards enforcement
Users	**demanding more capabilities**
Personnel	lack of experience
	image/morale problems
	view of maintenance: unchallenging, unrewarding

Table 2.1 Software Maintenance Problems

Osborne states that we can reduce the chance of incorrectly or inadequately specifying requirements or requirement changes by *prototyping*. Prototyping is usually employed to shorten the time required to code and test an application or to understand and demonstrate development capability. Pressman [20] defines prototyping as 'a process that enables the developer to create a model of the software that must be built'. The focus is on the work product rather than the development process. Prototyping enables experimentation with the construction of the desired system from which lessons can be learnt for revising requirements [23]. Protoyping, thus, helps in improving requirements change specifications.

Brooks [42] states that:

In most projects, the first system built is barely usable. It may be too slow, too big, awkward to use or all three. There is no alternative but to start again and build a redesigned version in which

these problems are solved. When a new system concept or technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time. The management question, therefore, is not whether to build a pilot system and throw it away. You will do that. The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers.

Schneidewind [16] explains that even if the code is inelegant and possibly not reusable, a study of the specifications and identification of the most frequently used components could reveal a set of generic classes of algorithms and functions which could be usable in future systems. Belady [24] believes that we cannot and should not declare *old* software obsolete or not worth studying. Harrison and Cook [68] state that managers are frequently beset by the maintenance problem: whether to make an isolated change in an existing module, or to totally redesign and rewrite the module from scratch. Arthur [14] advocates rewriting the system from scratch and treating the existing system as a prototype to build a new system in any of the three scenarios: the system is expensive to run, out-of-date technically, or expensive to maintain.

Lientz and Swanson [25] state that the approach which is in vogue of getting the requirements right before starting the design may be based on the fallacious assumption that requirements are fixed. The reality is that requirements change continually, often in response to organizational change. These changes are more likely to emanate from experience in the use of the system than from an abstract specification in the early design of the system. We can draw an analogy between 'organizational change' in production environments and 'environmental evolution' in research environments. It is interesting to note that because requirement changes are likely to result from,

(be *influenced* by) changes in the environment, it is imperative to study the environment characteristics as well. Studying the environment characteristics enables us to identify the type and magnitude of requirement changes as an impact of an evolving environment and thus helps in chalking out strategies to cope with an evolving environment.

2.2 Environment Characteristics

Examining the two distinct types of software development environments reveals different characteristics which form the underlying reasons why a system follows different evolutionary paths in the two environments. Table 2.2 characterises these two environments and analyses the impact on requirements changes of individual environment characteristics.

Factor	Highly Innovative Research Environment Characteristics and Impact on Requirements Changes	Production Environment Characteristics and Impact on Requirements Changes
1. Motivation	Characteristic: The need to learn, discover and prove concepts is the primary objective Impact: Requirements are likely to change frequently due to the 'experimental' nature of the work	Characteristic: Monetary gain by marketing a product based on validated concepts or demonstrated technology is the primary objective Impact: Requirements are less likely to change frequently due to 'stability' in the nature of work

Table 2.2 Characteristics of research and production environments and impact on requirements changes

Factor	Highly Innovative Research Environment Characteristics and Impact on Requirements Changes	Production Environment Characteristics and Impact on Requirements Changes
2. Goals	Characteristic: The champion of the system has a short term goal of obtaining his/her degree or result within a certain period; whereas the supervisor has a long term goal of developing a set of solutions and the POC only forms a part of his/her repertoire Impact: Requirement changes are less likely to be carried out by the original champions of the system	Characteristic: The project remains an 'on-going' effort and the initial champions or teams of the system are generally available for a comparatively longer time frame Impact: Requirement changes are more likely to be carried out by the original champions or teams
3. Target system	Characteristic: Due to high uncertainty in the achievement of the goals and due to the usual academic constraints, the POC is *the* target Impact: Because the POC is primarily to validate key concepts, and the goals are uncertain, requirements changes even after successful completion of the POC can be severe	Characteristic: Due to low uncertainty in achieving the project goals and organisational survivability goals, *future releases* of the system is *the* target Impact: Because the focus here is on 'future releases' of the system, requirements changes are likely to be less severe 'evolutionary' changes
4. Theory and concepts	Characteristic: Weak or non-existent Impact: Better understanding of the underlying concepts usually results in evolution of system requirements	Characteristic: Considerably stable Impact: Evolution of system requirements is less often because of relative stability
5. Risk type	Characteristic: Projects are highly innovative and thus risks involved are *discovery* or *innovation* risks; can a product be built at all? Impact: The requirements changes are controlled, often by further basic research, to reduce discovery risks.	Characteristic: Projects are relatively banal and thus risks involved are *delivery* risks; can the product be developed of the right quality, on time, and within budget? Impact: The requirements changes are controlled, often by the use of appropriate methods and tools, to reduce the delivery risks.

Table 2.2 Characteristics of research and production environments and impact on requirements changes

Factor	Highly Innovative Research Environment Characteristics and Impact on Requirements Changes	Production Environment Characteristics and Impact on Requirements Changes
6. Severity of risk	Characteristic: Due to high degree of innovation and thus uncertainty in achievement of goals, risks are very high Impact: High risks often affect the requirements in quite unpredictable ways	Characteristic: Severity of the risks involved, even-though of a different type, are often dependent on the maturity of the developing organisation Impact: Changes to the requirements are often controllable through the use of appropriate technologies (e.g., configuration management systems) and methods (e.g., user participation)
7. Evolvability	Characteristic: The goal of long term evolvability is often not even thought of, prior to the realization of a POC, and it often emerges as a primary goal once there is an evidence that the POC system is successful Impact: Since the focus is on the validity of the POC rather than its evolvability, functional requirements take precedence over non-functional requirements. These 'ignored' non-functional requirements then form a significant part of the requirements change effort	Characteristic: The goal of long term evolvability is often (ideally, should be) thought of from the very beginning of the project Impact: Since the stress here is on *both* functional and non-functional requirements, requirements changes may result from changes in functional or non-functional requirements or both
8. Development process	Characteristic: Ad-hoc development processes are followed Impact: The requirements change pattern is not regular in any way	Characteristic: Generally, some software development process model is followed, v.i.z., waterfall model, spiral model, evolutionary model, etc. Impact: The requirements changes pattern is often dictated by the process model used (e.g., large volume of changes in the waterfall model, incremental changes in the evolutionary model, etc.)

Table 2.2 Characteristics of research and production environments and impact on requirements changes

Factor	Highly Innovative Research Environment Characteristics and Impact on Requirements Changes	Production Environment Characteristics and Impact on Requirements Changes
9. Team size	Characteristic: There is usually one champion who has a deep insight into the theory, concepts, implementation, and operation of the system Impact: The requirements changes cycle is usually short when simple and extremely long when complex, based on the understandability of the problem and solution spaces	Characteristic: There is not one 'champion' per se but a 'team' of champions amongst whom the system basics and implementation details are distributed Impact: The requirements changes cycle is generally regularised by a change management process, based on business goals.
10.Pride and ownership	Characteristic: The champion has a strong sense of ownership of the system (high pride) Impact: There is usually a strong personal motivation and commitment to requirements changes	Characteristic: This sense of ownership is distributed across the developers involved Impact: There is usually a loyalty or business motivation and commitment to requirements changes
11. Commitment	Characteristic: The champion is on the 'learning track', implying he/she is likely to be out of the environment as soon as a significant milestone (e.g., a degree) is reached (short term commitment) Impact: The knowledge on requirements changes is often not documented for future needs, perhaps because of immature processes, lack of resources, time, etc.	Characteristic: The champions of the system are generally salaried employees and are not usually bound by any milestones to resign from the job (long term commitment) Impact: The knowledge on requirements changes is often documented (in mature organizations) for future needs because of system and organizational survivability
12. Budget	Characteristic: Budget for system development is generally low Impact: There is usually not budget set aside explicitly to manage requirements changes	Characteristic: Budget for system development is relatively higher Impact: A portion of the budget is usually set aside to manage requirements changes

Table 2.2 Characteristics of research and production environments and impact on requirements changes

Factor	Highly Innovative Research Environment Characteristics and Impact on Requirements Changes	Production Environment Characteristics and Impact on Requirements Changes
13. Comparables	Characteristic: No comparables for the system being built Impact: Because systems in such environments are unprecedented, changes in requirements can be sudden, severe and totally unexpected	Characteristic: Several comparables for the system being built Impact: because several comparables for systems in such environments exist, requirement changes are generally gradual, less severe and less unexpected
14. Customer type	Characteristic: Research and scientific community Impact: The requirements changes affect the pool of scientific knowledge on the subject	Characteristic: Individuals, organizations and governments Impact: The requirements changes affect the service provided to the customer
15. Documentation	Characteristic: Poor or no documentation exists Impact: Poor documentation acts as a major impediment in tracking original system requirements and clearly specifying their change or evolution	Characteristic: Substantial documentation usually exists Impact: Because original system requirements are usually explicitly identified, specifying their change or evolution is much easier

Table 2.2 Characteristics of research and production environments and impact on requirements changes

The described differences in environment characteristics have a profound effect on requirement changes in the two types of environments. In the production environment, generally the development processes used are the waterfall [31] or spiral [4] processes alone with commercially available methods, techniques and tools.The development effort is focused on customer satisfaction and the causal factors (e.g., adequacy of the system features, quality of service from the system, development costs, definite delivery dates, short delivery cycles for future enhancements, user-friendliness of the product, etc.). This primarily stems from the organizational need to survive in a competitive market. Thus, *customer satisfaction issues* play a significant part in dictating the requirements changes in such environments. On the other hand, in a research environment, often

the development processes followed are ad-hoc using primitive development methods, techniques and tools. This primarily stems from the fact that the champion's focus is geared towards demonstrating the POC within a certain time-period, which is usually bounded by the availability of project funds. Thus, in such environments, *environmental issues* play a major part in changing system requirements.

2.3 Summary

Requirements changes drive software evolution activities and are influenced by the environment characteristics. The environmental characteristics have a bearing on both the type and magnitude of requirement changes. Attempting to build a research system using a production process (in order to escape from severe, frequent and unexpected requirements changes in research environments) could lead to severe problems in achieving the primary goals. If the rigorous production environment development processes, methods, techniques and tools were to be employed in a research environment, the project is likely to run into difficulties. For example, the POC is unlikely to materialize in the 'shortest-possible' time span; it would be beset with problems of decreasing project funds; research time over-runs for the champions would increase; and resources would be wasted in a stringent development process when the project is a highly innovative, uncertain and risky exercise. Similarly, if in a production environment, the 'ad-hoc' work processes followed in the research environment were to be employed, there would be: a debilitating effect on the product quality; cost and schedule over-runs; lowered customer satisfaction and other unpleasant circumstances. The software engineering community is learning that advances in productivity and quality do not materialize only because of the use of advanced software tools, and that the quality of the software product is governed significantly by the quality of the proc-

esses used to create and evolve it [5, 11, 12]. Humphrey et al. [11] state that an effective process must consider relationships amongst the required tasks, tools and methods, and amongst the skills, training, and motivation of the people involved. In a mature software process, elements such as people, methods, techniques and technology are effectively integrated to develop quality software consistently within the constraints of cost and schedule requirements.

Chapter 3

The CES Proof of Concept System

As mentioned in Chapter One, a key issue in this research is identifying the causes for the demise of the CES POC system and the lessons that have been learnt from this experience that can be utilized in the development of such POC systems in research environments. In order to better understand the causes, one should study the design strategy adopted for developing the CES POC system and the rationale behind it. In this chapter, we first give a brief description of the concept of *process model congruence*[1] and describe the salient features of the CES POC system with the aid of examples. We then discuss the design strategy for the design of the CES POC system and its rationale.

3.1 Congruence Evaluation and Design Assistance

The CES POC system is a tool that assists in the evaluation of process model congruence and process model customization [10]. *Congruence* is a measure of how fit a process model is in the given development environment in which it is used. The development environment is referred to

1. the words 'fitness' and 'congruence' are used interchangeably in this thesis

as the *process context*, which is defined by the set of key characteristics of the total environment housing a process [43, 44]. These include the corporate culture of the organization, the size and complexity of the software system to be developed, the experience of the practitioners, the tools they are using, the team structure and size in the project, budget, delivery cycle time, etc. The CES POC system is based on the premise that an increased congruence measure for a given process model will result in an increased process effectiveness. An extensive body of literature suggests that an organization's ability to achieve its goals is a function of the congruence between various organizational components. Incongruence between the methodology and the style and culture of the organization is one of the possible causes of failure to implement a methodology [58]. If the components *fit well*, then the organization functions effectively; if they *fit poorly*, it will not [45]. The strategic management, organization theory, and organization behavior literatures are replete with statements that suggest that a particular structure should be *matched* [46, 47], that technology *dictates* structure [48, 49], that the environment and strategy should be *aligned* [50, 51, 52], that administrative systems should *fit* strategy [53] and that reward systems should be *congruent* with strategy [54, 55, 56].

The CES POC system is based on a method for the evaluation of process model congruence which was developed at McGill [10]. This method was developed after analysing the data gathered during a field study to determine the relationship between process model and process context characteristics with respect to process performance. This evaluation method resulted in a congruence measure that was validated to ensure that the congruence measure had empirical significance. A process model measures high on the congruence scale if it is fit in its environment (i.e., if its properties are suited to those of its context).

The evaluation method takes as input the values for a set of process context attributes for one project, and the values for a set of process model attributes representing the model under study. It provides as output a value (i.e., congruence index) indicating the model's congruence with the process context, a list of congruence *trouble-spots* found in the model, and the congruence value for each process model and context attribute (Figure 3.1).

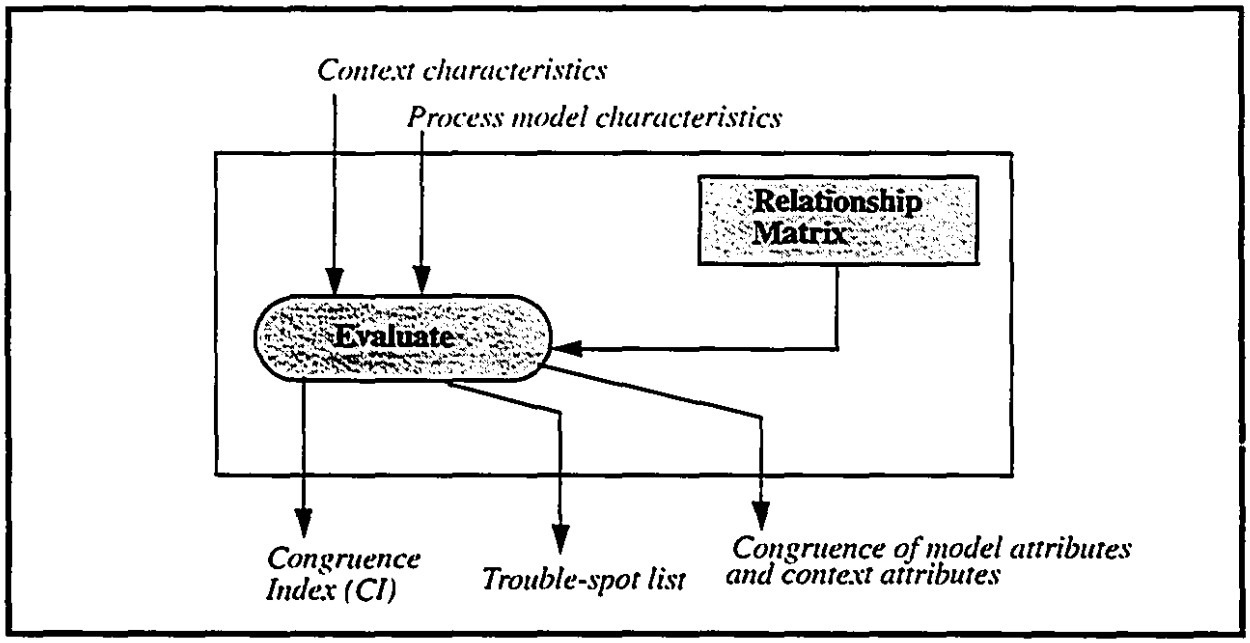


Figure 3.1: Congruence Evaluation System - Evaluation function

The congruence index is meant to be used as an overall index which may be used to make congruence comparisons among different process models and a given context, or amongst different process contexts and a given model. The congruence value of model attributes provides a good indication of which values of model attributes have low congruence with regards to the context. The *trouble-spot list* displays those model and context attributes which have low congruence with respect to each other. The relevant process model attributes may then be subject to change. Using the trouble-spot list, one can therefore analyse the cause of congruence problems and take corrective action. Thus, the trouble-spot list aids the user in performing *process improvement.*

An important purpose of evaluating the congruence of process models is to use the evaluation results to improve the fitness of the process model (i.e., in the *process adaptation process*). The congruence of context attributes provides an indication of which values of the context attributes have low congruence with regards to the process model. The relevant context attributes may then be subject to change. Although this is not always possible, at times the context can be adapted to improve its congruence with the process. Thus, the tool can could find practical application in implementing *organizational change*. Also, the method helps in *process reuse* by enabling the user to evaluate the fitness of the same process model in different contexts.

The CES POC system also helps in the *design* of congruent process models by using a method quite similar to the congruence evaluation method: the *design assistant method*. The design assistant method takes as input the values for a set of process context attributes only. It provides as output the congruence of the different possible values of each process model attribute. This output can then be used to decide the process model attribute values which are the most appropriate for the context (Figure 3.2).

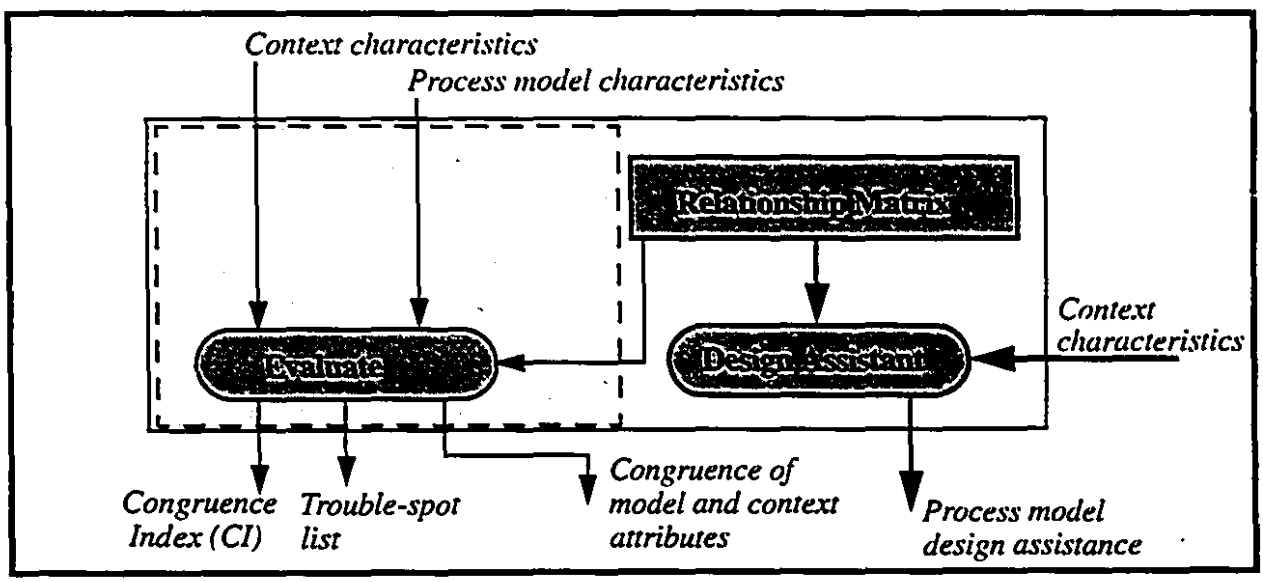


Figure 3.2: Congruence Evaluation System: Design-Assistant method

In the next section, we illustrate the use of the CES POC system with the aid of three example applications. The data that is used for the examples has been obtained from actual projects and processes in the field study.

3.2 Tool Usage and Examples

Example 1 illustrates the evaluation of congruence measures for a process model, and the interpretation of the results for improving the process model. Example 2 demonstrates how the system can be utilized for choosing the most appropriate process model in a given context. Example 3 shows an application of the tool for process model reuse.

3.2.1 Example 1: Evaluation and Improvement of a Process Model

The first two steps of the evaluation method consist of characterizing the process model and the context under study [10] (see Appendix A). Once the process model and characteristics have been input, the evaluation algorithm is performed and the output is displayed, as shown in Figure 3.3. The screen dump shows only part of the information, but it is possible to scroll through the whole list of attributes-congruence and the trouble-spot list.

In this example, the evaluation yields an overall congruence index of 0.71. In order to analyse possible improvement of congruence, the 'trouble spot list' should be consulted to determine process model and context attributes with a low congruence value. The process model and context attributes which have a low mutual fitness, are listed in 'pairs' as *process attribute* and *related attribute*. We can alter the values of the attributes with low congruence and again recompute the congruence for the process model and the context. It should be noted that, an alteration in the val-

07/02/94 McGill - CONGRUENCE EVALUATION SYSTEM 9:02:38 am

PROCESS MODEL CONGRUENCE

CONGRUENCE INDEX: 0.71

PROCESS MODEL ATTRIBUTES - CONGRUENCE

Process Attributes	Value	Congruence
Exit criteria	Lower than	0.00
Output/Deliv divisionalization	Yes	1.00
Integration mechanisms	Low	0.00
Devices for underst/valid func	Prototypes	-1.00
Coordination mechanism	Stand. of	1.00
Formaliz. of change management	Low	1.00
Resources PM control & plann.	High	1.00
Amount of Project Manag. deliv	High	1.00
Emphasis on control and plann.	High	1.00

PROCESS MODEL - TROUBLE SPOTS

Rel	Process Attribute	Value	Related Attribute	Value	Type
-	Integration mechanisms	Low	Project size	Large	C
-	Devices for underst/valid	Prototy	Application uncertainty	Low	C
-	Devices for underst/valid	Prototy	Prototyping tool support	Not Ava	C

Figure 3.3: Output generated by the evaluation algorithm when input was process model 1 and context 1

ues of the attributes may not necessarily result in an *improved* (higher) congruence index. This is due to the fact that certain model and context attribute values may be 'negatively' related. The model thus reaches an *optimum* congruence value when changing the values of attributes listed in the trouble spot list does not result in any change in the overall congruence index.

3.2.2 Example 2: Selection of a Process Model

Often a project manager has a library of process models from which an appropriate one needs to be selected for a specific project. In this section, we demonstrate that the CES POC system can aid in such a decision making process.

Assume that the project under study can be characterized by process context 1 (see Appendix A), and that the project manager has to choose between process models 1 and 2 (see Appendix A and B). Then, he/she has to evaluate both the process models with context 1 and compare the results. The one with the higher overall congruence index would generally be the more appropriate for the context. The results of performing congruence evaluations of the two different process models 1 and 2, with the same context 1 yield the results shown below:

Process Model	Overall Congruence Index
Model 1	0.86
Model 2	0.10

Table 3.1: Overall congruence indices for different models in same context

The comparison shows that the likelihood of process success is greater using process model 1 than using process model 2.

3.2.3 Example 3: Process Reuse

Modelling a software process can be a costly exercise in terms of effort and money invested. Moreover, there is a risk of developing a new model of low quality which does not measure up to expectations. Thus, it is beneficial to reuse a familiar and sound process model in a new project. Prior to such a reuse, however, it is advisable to assess the congruence of the old model in the new environment.

For example, if we desire to reuse process model 1 in process context 2 (see Appendix A & B),

after having used process model 1 in process context 1, then we need to evaluate the congruence between process model 1 and context 1, and between process model 1 and context 2. The result of these evaluations are given below:

Process Model	Context 1	Context 2
Model 1	0.71	0.60

Table 3.2: Overall congruence indices of same process model in different contexts

The difference in the congruence indices indicates the appropriateness of using process model 1 in context 2, perhaps even as a starting point for minor modifications. The process model may then be 'tailored' specifically to fit context 2.

3.3 Design Strategy for the CES POC System and its Rationale

In order to understand the causes for the demise of the CES POC system one should analyse the design strategy used in the development of the system. This will also provide us with an insight into the question (Chapter 1) which we seek to address in this thesis.

Also, along with analysing the design strategy adopted for the design of the CES POC system, it is imperative that we also study the *rationale* behind this strategy. This will enable us to understand the factors behind the adoption of the design approach.

3.3.1 Design Strategy for the CES POC system

When the CES POC system was implemented, the prime focus of the design strategy was on dem-

onstrating, within the shortest time possible and within the limited budget, the concept of congruence and the validity of the congruence method. This is in direct contrast to the approach in the re-implementation, where the prime concern was one of evolvability of the system while retaining its validity.

The design strategy for the CES POC system was thus to employ the programming language and the development platform most familiar to the developer. It was essentially a *prototyping* approach to *understand* (empirically) the underlying theory of congruence and process fitness. It may also be viewed as *exploratory* programming which is used in circumstances where fixed system specifications are generally not available, and where a determination of feasibility is more important than abstract correctness [37]. It should be noted that because it was the first implementation of such a system, the requirements were not clearly specified and thus the developer was expected to follow an exploratory approach. This implied that the work process followed was 'ad-hoc'.

Also, the usual software development concerns for user customizability, user friendliness, future evolvability, and integration into existing suite of tools did not form part of the design strategy. With hind-sight, one could agree that these are critical issues. However, at the time of the CES POC system development, all these issues were superseded by the prime focus of demonstrating the *proof of concept* as fast as possible and within the limited research budget. Also, the *state* of the environment of the system (i.e., other related tools being developed at McGill) was not clear. It too was evolving without a strict overall focus. Thus, the described concerns for the CES POC system only emerged as prime concerns once the CES POC system was found to be significant

from the research point of view.

3.3.2 Rationale for the Design Strategy

The urgency in time for the completion of the CES POC system stemmed from two major factors: The first factor, competitive research, is invariably found in the field where competing research teams are engaged in high-end research. The drive to produce novel results is directly related to the evaluation by thesis committees and conference and journal refrees. The second factor, time limitation, was due to the fact that the developer of the system had already invested a significant amount of time in experimental work on gathering data from a field study and in developing a method for evaluating congruence, and was thus left with extremely limited time to actually develop a software tool which demonstrated and validated the concept of congruence.

Another important factor was the limited budget available to support the project until a specific deadline set by the research supervisor. This factor, coupled with the fact that research results were uncertain, meant that the CES POC system had to be completed within the shortest time span in order to minimize possible loss. Unlike in the production environments, wherein much stress is laid on *exciting* requirements in addition to *normal* and *expected* requirements, the stress in the case of the CES POC system was basically on the normal requirements [59]. That is, the normal requirements *were* the exciting requirements.Thus, the issues of user customizability, user friendliness, evolvability and system integration did not make the development agenda. This was not so much a conscious decision as much as it was the pressing constraints that rendered a tunnel vision. Moreover, the software development exercise was similar to a 'technology demonstration' and was not intended to be marketed. Understanding the underlying congruence theory with the

aid of the system was *the* reason why a 'prototyping' approach was adopted for system implementation.

Yet another factor that drove the design and implementation strategy was the development *skill* of the particular researcher. The decision was to use the programming language FoxPro in order to ensure minimum learning time and rapid development. The development platform, MS-DOS, which allowed the software to be loaded on a lap-top and thus was handy for presenting at meetings and presentations.

3.4 Summary

There were some critical factors that drove the design strategy: competitive research, severe time limitation, severe budget constraints and development skills. In an unconstrained environment, these factors would not have been significant and thus an alternate design and implementation strategy would have been more appropriate (e.g., evolutionary approach tied in with learning of appropriate development skills). Arguably, the use of the alternate strategy in the described constrained environment could have meant total failure of the development project.

Chapter 4

Analysis of the CES POC system

In the first section of this chapter, we analyse the CES POC system in order to determine the requirements satisfied by the system and to identify the deficiencies in the system which contributed to its demise. We first describe the analysis method and then the analysis results. In the second section, we summarise the key-points of this chapter.

4.1 Analysis of the CES POC System

In this section, after describing the analysis method and results, we list the requirements satisfied by the system, (*old* requirements) and the requirements which were *not* satisfied by the system and thus led to *new* requirements. It should be noted that because the CES POC system was implemented using the typical 'ad-hoc' development method adopted in research environments, the system requirements were not explicitly documented in a software requirements specification document, although there were some functional and non-functional requirements listed in [10] which the system was supposed to satisfy. We, therefore, assessed the system more from the 'quality' perspective and as per the quality criteria listed by Boehm et al. [64] which we further augmented by including certain criteria which we deemed necessary.

4.1.1 Analysis Method

The quality criteria which were used to assess the CES POC system can be categorized into two groups: *system-specific criteria* and *environment-specific criteria*. All these criteria were included in an 'Instrument to assess system deficiencies and change in requirements' (see Section A of Appendix C). The instrument developed consisted of 15 criteria to assess a system from a quality standpoint. Of these, 12 criteria were the quality criteria for system assessment [64] which were predominantly 'system-specific'. The other 3 criteria were predominantly 'environment-specific' and these were determined during the course of interviews conducted by the author with other members of the research team. The several rounds of interviews focused on answering the question: "What criteria can be used to assess the deficiencies of the CES POC system?". Determining the deficiencies of the system at the time of completion would help us list the 'new' system requirements which could be considered during the re-implementation of the system, so as to eliminate the identified deficiencies. Considerable time was spent in follow-up interviews on modifying, refining and rewording all the criteria listed so as to make them lucid and unambiguous.

Based on these criteria to assess the CES POC system deficiencies, we conducted a survey and distributed the instrument to the research team members (number of respondents: 8, 100% response rate). For this instrument, a semantic differential scale was utilized [65,.63]. This scale consists of a concept and a bi-polar (opposite-in-meaning) adjective pair at the extremes of a 7 point scale. The instrument was also *construct validated* [57] by the research team members so as to ensure that the scales chosen did describe the true construct, the construct here was *system assessment*. The instrument also required the respondent to include his or her 'confidence level' in

giving the response and also the rationale for the response. Subsequently, during the data analysis phase, responses with a confidence level of less than 6 ('Quite High' confidence) were eliminated from data analysis, while the mean of all responses with a confidence level of 6 or higher was considered for system assessment purposes.

4.1.2 Analysis Results

The analysis results of the data gathered during this survey are shown in Figure 4.1. It is suggested that Figure 4.1 be studied in tandem with the questionnaire (Section A of Appendix C) so as to better understand the data analysis results. Basically, the survey fell under two types of questions:

(a) Type 1: *What was good about the CES POC system? that is, what were the purposes served by the CES POC system?*

This helped us to identify the system requirements (hereafter, the old requirement set: 'R1') that were satisfied by the system. It is important to note that, the set R1 (R1.1 to R1.13) also includes functional and non-functional requirements for the system which are listed in Table 4.1 and are not being discussed here because they did not form part of the questionnaire which was an assessment from a quality perspective.

(b) Type 2: *What were the deficiencies of the CES POC system as at the time of completion of the system in Sept.'94?*

This enabled us to identify the system requirements that were not satisfied by the system, and which matured into new requirements for the system re-implementation (hereafter, the new requirement set: 'R2'). The requirement set 'R2' (R2.1 to R2.9) is shown in Table 4.2.

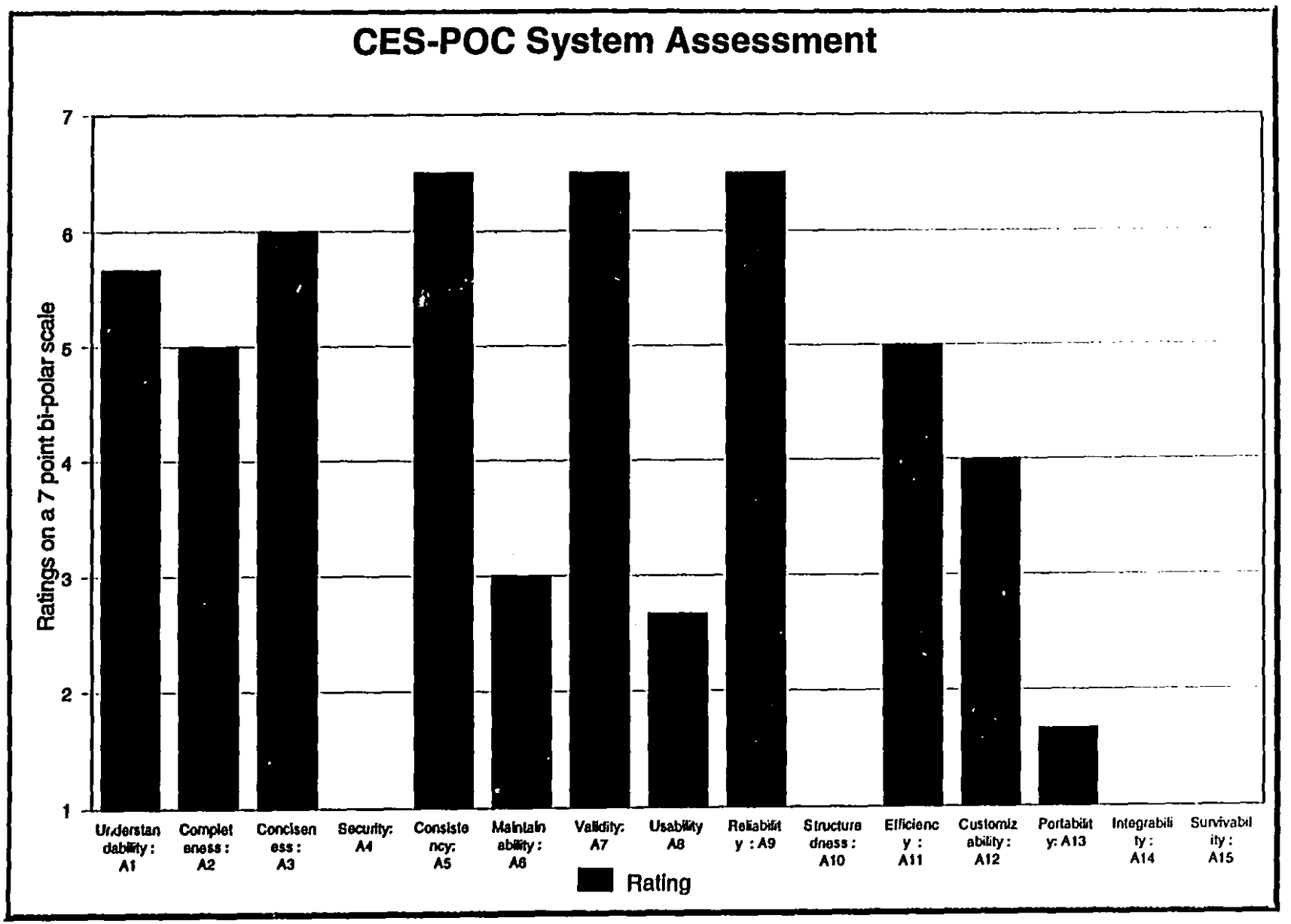


Figure 4.1: CES-POC system assessment

4.1.2.1 Type 1: Purposes served by the CES POC system

Figure 4.1 shows that the CES POC system was assessed as 'Quite High' for the criteria[1] of system 'conciseness', 'consistency', 'validity' and 'reliability'. As stated in chapter 3, when the CES POC system was built, the prime concerns were understanding (empirically) the underlying theory of congruence (with the aid of the system) and validating the system with respect to the concept of congruence. Thus, the key requirements for the CES POC system were to have an easily understandable system which was reliable, valid and concise. Figure 4.1 shows that the system satisfied these key requirements of 'conciseness', 'validity', 'reliability' and 'understandability'. The system was also regarded by the respondents as highly 'consistent' in its operation. The high consistency of the system was due to the fact that the requirement that the system exhibit consistent terminology, symbols, notations and concepts in its operation were closely tied to the requirement of 'understandablity'. Poor consistency would have inevitably contributed to a poor understanding of the working of the system by the users. Clearly, the system was driven by certain key requirements which *had* to be met. The system performance based on these key issues was thus highly satisfactory.

When the CES POC system was implemented, there were certain other key requirements which though equally important were not as critical as the ones listed above. These may thus be regarded as *expected requirements* as opposed to the *normal requirements* described above [59]. These expected requirements were that the system be reasonably 'complete' in providing all the key features in the domain of process fitness and the system also be 'efficient'. Figure 4.1 shows that the

1. for a detailed explanation of the various criteria please refer to Section A of Appendix C

system ranked as only 'Slightly High' for both these requirements. This is primarily because, the respondents believed that due to the vastness of the domain of process fitness, it could not be confidently claimed that the system provided all the key features in the domain. However, the respondents were of the opinion that to the best of their knowledge, key features pertaining to process model congruence were present in the system though certain other features, e.g., *impact of change* on process fitness due to a change in the value of the process model/context attribute (see Chapter 7), needed to be added to the system. Also, the system efficiency was 'Slightly High' as was apparent from the use of the system but there was no formal documentation of the system to judge if the design implementation was also efficient, i.e., database accesses did not require excess CPU time or cause undue strain on memory requirements. Design documentation would be imperative to perform such an analysis so as to determine if there were more efficient design implementations possible (than the ones adopted) so as to optimize system performance.

As described earlier, all these requirements described above (which were to be satisfied by the CES POC system when it was built) can be regarded as belonging to the requirement set: 'R1'. Thus the requirements included in the set 'R1' formed the old requirements for the system (as at the start of system implementation).

The requirement set 'R1' is as follows:

Requirement Type	CES POC Requirements in Sept.'92 [R1]
Functional requirements	R1.1: The system should facilitate the evaluation process model fitness based upon the model/context attributes and their relationships.
	R1.2: The system must display the 'Trouble Spot List' for the process model and context attributes which exhibit poor fitness.
	R1.3: The system should assist in designing process models, i.e., given process context characteristics, it should identify the fitness of the different values of each process model attribute.
Non-functional requirements	R1.4: The system must employ the congruence evaluation and design assistance algorithms as developed during the congruence evaluation method study.
	R1.5: The system must be programmed on the MS-DOS operating system.
	R1.6: The system must be programmed in FoxPro programming language.
Quality Requirements	R1.7: The system must be easily understandable in its operation.
	R1.8: The system must be reliable in its operation (must repeatedly produce correct results).
	R1.9: The system must be concise in information displayed in screens without sacrificing understandability.
	R1.10: The system must exhibit consistent terminology, symbols, concepts and notations in its operation.
	R1.11: The system must provide all the key features in the domain of process fitness (to the best of knowledge of the researchers).
	R1.12: The system must be efficient in its operation without a waste of resources (e.g., CPU time, memory requirements, etc.).
	R1.13: The system must be validated to ensure that the congruence measures produced by the tool indeed characterize congruence.

Table 4.1: The requirement set 'R1' for the CES POC system

4.1.2.2 Type 2: Deficiencies of the CES POC system upon completion

We now discuss the requirements in R2 which specifically arose due to deficiencies identified in the CES POC system. These requirements can be discussed in two distinct categories: *system-specific requirements* and *environment-specific requirements*.

Figure 4.1 shows that the system-specific requirement which was 'Moderately' met by the system was that of system 'customizability' (Question A12), i.e., the requirement that the process model/context data stored in the database be changeable by the user. The system was very limited in customization because it only allowed for the change of 'selection' of a model/context attribute value instead of allowing the user to actually change the data stored. The system was regarded as 'Slightly Low' in 'maintainability' (Question A6), i.e., documentation support for future maintenance activities (corrective, adaptive or perfective). This was because there was no formal documentation of the system development barring the thesis of the researcher. Also, the system was regarded as 'Quite Low' in 'user-friendliness' (Question A8) due to the lack of on-help in both understanding and using the system.

The system was evaluated as 'Extremely Low' in satisfying the system-specific requirements related to 'security' (Question A4) and 'structuredness' (Question A10), and the environment-specific requirements of system 'portability', 'integrability' and survivability' (Questions A13, A14 and A15 respectively). The requirement of system security required that the system be safeguarded against possible damage to the internal system consistency due to change of the data by the user. For instance, if a process model attribute were to be deleted by the user, then, if some relations were dependent on the presence of this process model attribute, the system should warn

and prevent the user from making the deletion by displaying the relations dependent on the 'to be' deleted attribute. In such a scenario, the user would thus be required to ensure that no dependencies exist in the system on the attribute to be deleted, so as to ensure internal system consistency is maintained after deletion. The CES POC system *did not* have any system security features when it was implemented. As stated in chapter 1, the system lacked 'structuredness' since there was no modularization of the code.

Since the system was built using FoxPro on the MS-DOS platform, it was not 'portable' (Question A13) to an operating system like UNIX. Also, the system was completely stand-alone and did not form part of the process cycle tool-kit. Therefore, the 'integrability' (Question A14) of the system was low. The system was also not 'robust' enough to survive changes in the environment which was evident from the fact that the system had to be decommissioned soon after its completion because it failed to work in the new environment which had resulted from contiuous environmental evolution during the course of the development of the system (Question A15).

Thus, the new requirements (present in the set R2) for the re-implementation that emerged after this 'post-mortem' analysis of the CES POC system were: the system's database should be easily 'customizable', the system should be well documented so as to assist in system 'maintenance', the system should be 'user-friendly' and there should be 'help menus' throughout the system screens, the system should be programmed in a 'structured' language so as to aid in 'intrinsic' understandability of the system and system maintenance, the system should be 'integrable' with the rest of the suite of tools; this required that the system be implemented on the same OS (here, Sun OS) as the other tools. Finally, the system should be able to 'survivable' (atleast for a reasonable dura-

tion) beyond system completion. It was decided not to impose the requirement of system 'security' for the re-implementation because it would require research into the dependencies within the system and could be dealt with in the future work dealing with the 'impact of change'.

The requirement set 'R2' is as follows:

Requirement Type	New requirements due to CES POC system deficiencies [R2]
Non-functional requirements	R2.1: The system must be programmed on the Sun OS operating system.
	R2.2: The system must be programmed in C++/MOTIF programming language.
Quality Requirements	R2.3: The system must be user-programmable. i.e., it must allow the user to change the data stored in the database.
	R2.4: The system must be well documented to assist in future maintenance.
	R2.5: The system should be user-friendly and there should be 'help menus' throughout the system screens.
	R2.6: The system should be programmed in a structured language so as to aid in 'intrinsic' understandability of the system and system maintenance.
	R2.7: The system should be portable to different or compatible families of UNIX with minor changes. e.g., LINUX
	R2.8: The system must be integrable with the process cycle tool-kit.
	R2.9: The system must be survivable (for a reasonable duration) in environment changes.

Table 4.2: The New Requirement set 'R2'

4.2 Summary

The Type 1 question helped in identifying the requirement set R1, which basically reflected the functional strength of the CES POC system. The Type 2 question helped in identifying the requirement set R2, which basically reflected the weakness of the CES POC system. Here, it is interesting to note that the new environment-specific requirements ('portability', 'integrability' and 'survivability' in Figure 4.1) were more devastating to the CES POC system than the new system-specific requirements (all 'other' requirements in Figure 4.1). That is, while some of the new *system-specific* requirements could have been satisfied to some extent with some effort, *all the new environment-specific requirements required that the system be re-implemented.*

For example, the system design and code could have been documented after implementation so as to improve the maintainability of the system. Similarly, the system code could have been improved to provide for 'help menus' throughout the system and this would have led to an improved usability. As opposed to these system-specific requirements which could have been satisfied after system completion, *the same approach would not have worked for the environment-specific requirements*. For instance, the fact that the system had been programmed in FoxPro rendered it unportable to the Sun OS environment in which the other prototype systems were being developed. Portability required that the system be implemented in a UNIX compatible programming language, e.g., C, C++, Pascal, etc. The issue of 'integrability' was also closely tied to the issue of 'portability'. Since the CES POC system was not portable to the environment of the other prototype systems, it was essentially isolated and could not be integrated into the suite of tools being built by the entire research team. Clearly, the severe impact of the environment-specific requirements on the system led to its demise.

Chapter 5

The Impact of Environmental Evolution

Having analysed the CES POC system as at the time of system completion in Chapter 4, in this chapter we first study the evolution of the development environment during the course of system implementation. This is essential because, as will be demonstrated in Section 5.2, the changes in the development environment have a profound effect on the requirements of the systems housed in that environment. Therefore, the requirements for a system, change *not only* due to the new requirement set R2, but *also* due to new requirements which are introduced as a result of the environmental evolution (the requirement set: 'R3').

5.1 Environmental Evolution

Before we study the environment during the course of the development of the CES POC system (Section 5.1.2), we examine the environment (at a macroscopic[1] level) from Jan.'91 onwards when the process cycle [17] first started influencing the software development activities of our entire research team (see Figure 5.1). We then discuss the CES POC development effort and the related empirical research within this environment.

1. For a microscopic discussion on the environment, please refer to Section 5.1.2 and Section 6.2.

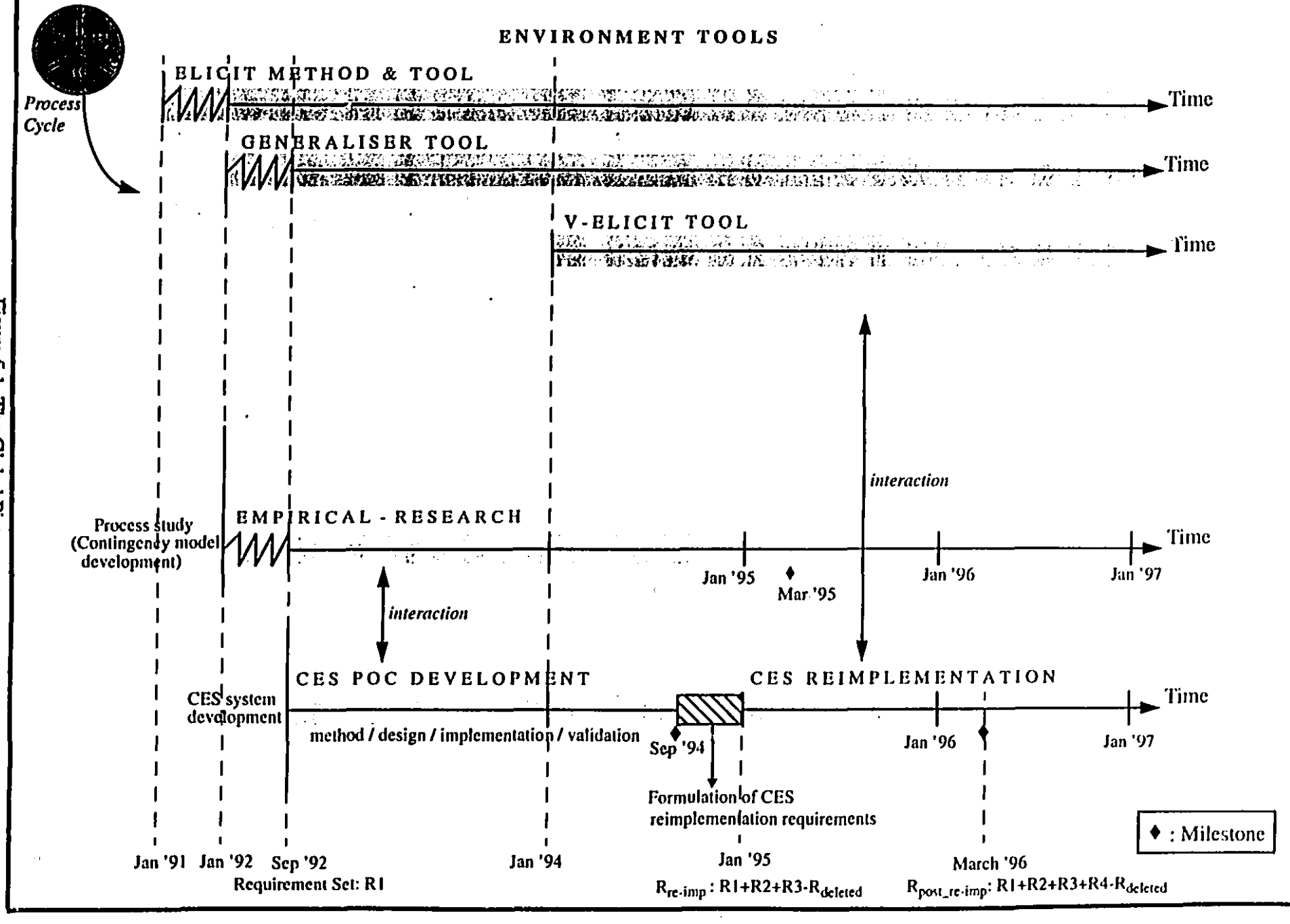


Figure 5.1: The Global Picture

5.1.1 The Global Picture

Figure 5.1 depicts pictorially the 'time traces' of the development of the various prototype systems in our software engineering lab from Jan'91 onwards. All the prototype systems, v.i.z., the Elicit tool, Generaliser, V-Elicit tool and the CES POC system have their roots in the concept of the process cycle, developed at McGill, which is a logical model for process definition, adaptation, enactment, measurement and improvement, based on formal process models [17]. The process cycle had little impact at the start, if any, on the *software development activities* in the lab. The key impact of the process cycle was initially in the process studies that were being carried out in collaboration with several organizations such as IBM Canada and DMR Group Inc. But from early 1994 (Question B14, Figure 5.2), there has been a growing realization of the actual impact of the process cycle on the software tools being built in the lab. Initially, all the prototype systems were being developed in isolation with a weak common vision (guided by the process cycle) for the entire team (Question B13, Figure 5.2).

The empirical research for developing the contingency model (on which the congruence method for the CES POC system was based) started in Jan'92 and the development of the congruence method commenced in Sep'92. Much of the time (approx. 1.5 years) in the development of the system was spent in developing the congruence method in close association with the on-going research in contingency models, with the rest of the time being spent on designing, implementing and validating the system. Since the congruency method was fundamentally based on the continually evolving (and refining!) contingency model, consequently there was a close 'interaction' between the empirical research and the CES POC development (see Figure 5.1 left hand side). These two lines of research were therefore closely tied together and were isolated from the envi-

ronment which was increasingly binding the other prototype systems (Elicit, Generaliser and V-Elicit).

Once the system was completed (in Sept.'94) and demonstrated to be valid with respect to the underlying concept of congruence, the focus in the re-implementation effort shifted to the *evolvability* of the system in conjunction with the suite of tools in the environment. Therefore, the *environmental* requirements were closely tied to the system in the re-implementation effort (see Figure 5.1 right hand side). Now, the process cycle also started exerting a strong influence on the re-implementation of the CES. The system was required to fit into the overall framework of the process cycle by enabling the adaptation of the improved generic process models. Therefore, the system was required to be invocable from the process cycle tool-kit. Thus, there was an increased interaction between the system re-implementation effort and the environment.

5.1.2 Environmental Evolution from Sept.'92 to Sept.'94

Having studied the overall picture of the development environment for all the prototype systems, in this section we focus specifically on the environmental changes during the course of the CES POC system development (Sept.'92 to Sept.'94). This will help us determine the consequent change in system requirements which was *the primary cause for the re-implementation.*

5.1.2.1 Analysis Method

The analysis method for determining the environmental changes was similar to the one adopted for determining the system deficiencies (Section 4.1.1). The criteria which were used to assess the

environmental evolution can be categorized into three clusters[1]: questions pertaining to *environment goals*, questions pertaining to the *predictors of environmental change* and other *miscellaneous* questions. The questions were derived from the interviews conducted by the author with other research team members and from the literature [28, 30, 41]. All these criteria were included in an 'Instrument to assess Environmental Evolution' (see Section B of Appendix C). The instrument developed consisted of 17 criteria to assess environmental evolution. Of these, 9 criteria were pertaining to environmental change, 3 criteria were related to predictors of environmental change while the remaining four were miscellaneous criteria.

5.1.2.2 Analysis Results

The analysis results of the data gathered during this survey are shown in Figure 5.2. It is suggested that Figure 5.2 be studied along with the questionnaire (Section B of Appendix C) so as to better understand the data analysis results.

In this section, we first analyse the environment at the start of the CES POC system development in Sept.'92. We then analyse the environment at the time of system completion in Sept.'94. Figure 5.2 graphically illustrates the environment as it existed in Sept.'92. Clearly, while there was a low realization (almost negligible!) of the environment goals (Questions B1-9, Figure 5.2), realization of the predictors of environment change (Questions B10-12, Figure 5.2) and other environment related issues (Questions B13-17, Figure 5.2) was basically non-existent. Interestingly, this low realization of the predictors of environmental change explains why the environmental changes had such a devastating effect on the CES POC system, since the environmental

1. The idea of clustering criteria in an instrument is borrowed from [65, 27]

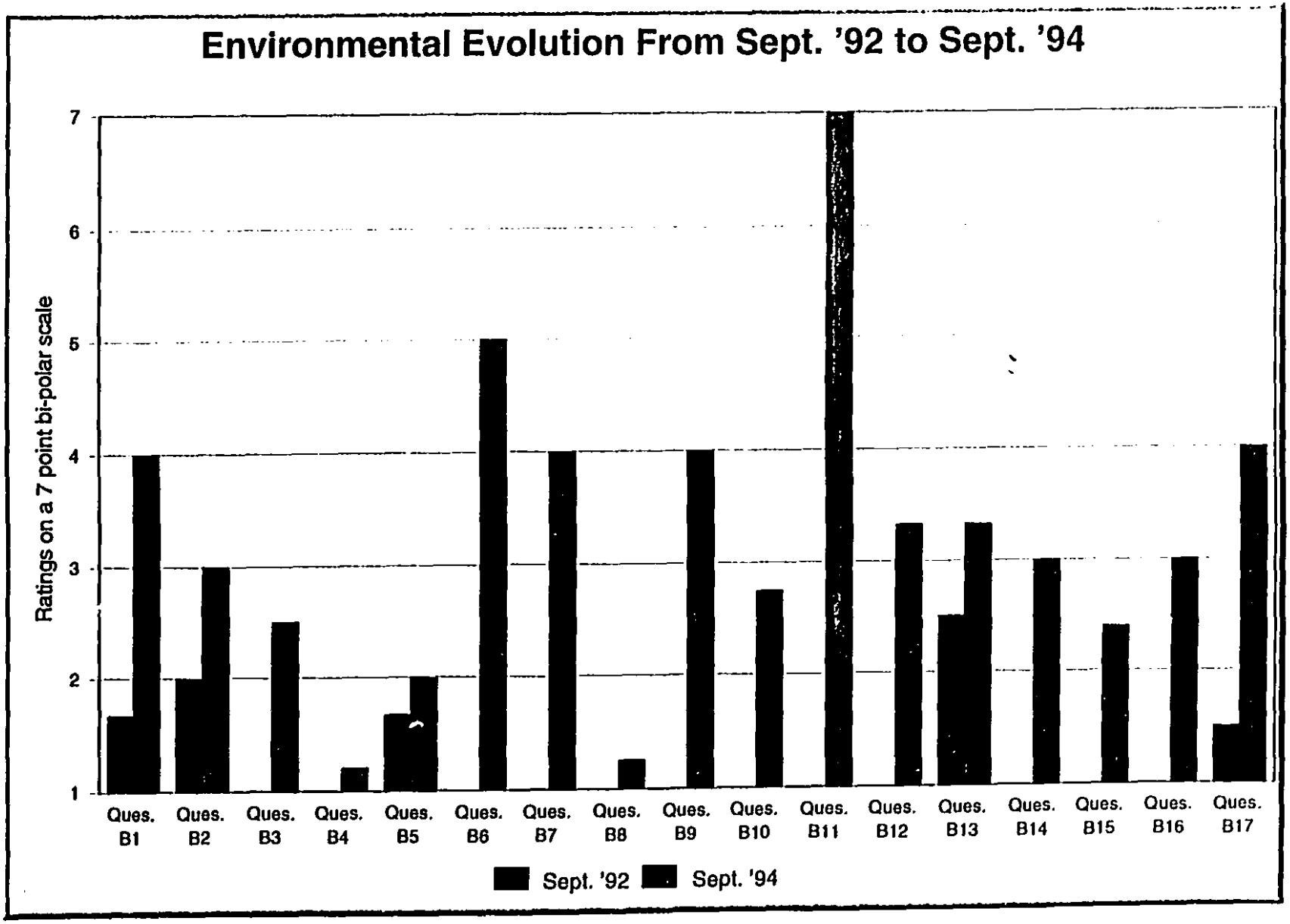


Figure 5.2: Environmental Evolution Snap-shots: Sept. '92 and Sept. '94

changes were totally *unexpected*. When the CES POC system was implemented, there was a certain degree of ignorance of the evolution of the environment and thus the survivability of the system. The ignorance of the researchers (in Sept.'92) about the environmental evolution and its likely impact on the existing prototype systems was also revealed by the extremely poor response to question B17 of the questionnaire (see Figure 5.2). The original developer of the CES POC system also believed that it would be possible to enhance the system in some way and blend it with the other tools.

That the process cycle had a weak impact on the development activities (in Sept.'92) is evident from the fact that there was a low realization of the existence of a common vision (Question B13, Figure 5.2) for the entire research team (guided by the process cycle). It may also be noted that while most of the respondents to the survey did not know (in Sept.'92) of the existence of this 'common vision', only the research team leader (hereafter, RTL), who was also the proposer of the concept of process cycle [17], had begun to realize how the process cycle could drive the research activities of the entire group.

At the time of system completion in Sept.'94, the environment had changed dramatically with an increased realization (generally, 'Slightly Low') about all the criteria mentioned in the questionnaire (see Figure 5.2). This growing realization was observed in the responses both from all the researchers and the RTL as well. Infact, the trend of increased awareness of the RTL of environment related issues was even more pronounced at this time (the RTL's responses on average recorded an increase in realization by four scale points from Sept.'92 to Sept.'94!). Now, the highest awareness was of the fact that there were flaws in the existing environment that could

cause it to evolve in the future (Question B11, Figure 5.2). However, this data should be interpreted with caution because all the other respondents barring the RTL responded with a 'Don't Know' response to the question. Evidently, answering the question required a greater insight into the prevailing environment. There was more unanimity in the response to the specific question of the realization of the goal of integrating the CES POC system in the process cycle tool-kit (Question B6, Figure 5.2). The respondents believed there was a 'Slightly High' realization about this goal. However, it should be noted that the system implementation had just been completed and a detailed 'post-development' analysis had just begun (Sept.'94 to Jan.'95, Figure 5.1) to analyse the system, its future and system enhancements. Only now was it being debated within the research team: *Where and how should the CES POC system fit in the process cycle?*

In Sept.'94, there was a 'moderate' realization of: the goal of having a process-cycle tool-kit comprising of *all* the prototype systems being developed; the goal of evolving the CES POC system in the environment; the change of focus from 'software process concepts and methods' to 'software process concepts methods *and tools*'; and, a likely serious impact of environmental changes on the prototype systems housed in the environment (See Questions B1,B7, B9 and B17 respectively in Figure 5.2). The realization of the goal of having a process-cycle tool-kit was much higher in the RTL when compared to that in the other individual researchers, who were aware more of their specific research and less of the global strategy for the entire research team.

The realization of the goal of evolving the CES POC system in the environment first evolved during this period (Sept.'94). The realization matured more in the post-development analysis of the system, when the analysis of the deficiencies of the system favoured a design approach for re-

implementation which would allow for a continual evolution of the system in the future instead of repeated decommissionings of future implementations and thus a loss of development effort. The shift of focus to tools was also higher for the RTL for the reasons explained earlier while the individual researchers, who were in somewhat mutual isolation (due to the independence of work and no framework to bind all work together), were moderately aware of the shift to 'tool focus' for the entire team. The research team was now beginning to realise for the first time the impact the environmental changes were having on the prototype systems, for instance, deficiencies (due to environment-specific causes) were being realized in the CES POC system even before the formal completion! Therefore, there was a moderate realization in response to question B17.

The responses to the rest of the questions broadly fell in two distinct categories: 'Slightly Low' or 'Quite Low' realization of environmental issues, and, 'Extremely Low' realization of environmental issues. Again, the RTL exhibited a higher realization in each of these responses when compared to the other research team members. The realization of the goal of data, control and platform integration varied from 'Slightly Low' to 'Quite Low'. These were specifically the different type of tool integration [30], and since the realization of the more general goal of integrating all the prototype systems in a process cycle tool-kit was itself low, the awareness of the details of tool integration was bound to be lower.

Now, due to the frequent interactions with software organizations, there was an increasing awareness of the benefits of tool-integration (Question B10). The organizations evinced more interest in a comprehensive tool-kit than in isolated tools. Also, now there was a moderate realization of a common vision for the team (Question B13). Since this question is closely tied to the goal of hav-

ing a process cycle tool-kit (Question B1), the reasonings for the response to that question apply here as well. The other questions (Questions B12, B14, B15 and B16) dealt with more subtle issues relating to the 'analysis' of the software systems housed in the environment. The 'Slightly Low' realization of these environmental issues (In Sept.'94) clearly showed a growing[1] awareness (since Sept.'92) of the impact of the environment on the software systems housed in it.

The lowest realization at this time (Sept.'94) was regarding the issues of control integration for all the prototype systems and also a distributed, client-server architecture (Questions B4 and B8). These were questions which required a good understanding and realization of data, user-interface and platform integration as a prerequisite. However, the realization for these prerequisites was itself poor, thus contributing to an 'extremely low' realization in responses to Questions B4 and B8.

Having studied the environment at two distinct time-shots, in Sept.'92 and Sept.'94, one can observe the steady increase of awareness in the entire research team of the environmental changes and related issues. This understanding of these environmental issues is important because it helps us understand how 'new' requirements (the requirement set: 'R3') emerge due to an evolving environment.

5.2 Requirements Changes due to the Evolving Environment

In this section, we study the *impact* of the described environmental changes on the change in system requirements. Specifically, we identify the new requirements (R3.1 to R3.3, see Table 5.1)

1. All these questions recorded an 'extremely low' realization in Sept.'92

that emerged for the re-implementation effort due to these stated environment changes.

It should be noted that the growing realization of all the criteria listed in the questionnaire (Section B of Appendix C) did not necessarily contribute to requirements in R3. This is because, although there was an increased awareness of certain key environmental issues, it was still not concrete enough to be moulded into formal requirements. This only happened in March'96 when there was an even more improved understanding of the environment.

The realization of the goal of having a process cycle tool-kit (Question B1) implied that no system should be stand-alone, i.e., which could not be executed as part of the process cycle tool-kit. This question was thus closely tied to Question B6 and, in part, to Question B10, B13 and B14. Although, there was a low realization regarding platform-integration (Question B5) at the time of system completion, however the new requirement that the CES POC system be integrated into the process cycle (Question B6) required that it be re-implemented on the same operating system (Sun OS). Both of these new requirements were thus closely associated.

The increased awareness of the goal of evolving the CES POC system in the environment (Question B7) led to the new requirement that the re-implementation should support evolvability. As stated in chapter 1, evolvability was now a prime concern in system re-implementation. Infact, the growing awareness in response to Questions B11, B12, B16 and B17 (See Figure 5.2) also contributed to the decision that the system be re-implemented in a 'flexible' way. This new requirement was also closely tied to Question B15, which addressed the specific issue of how the system could be designed in such a way that the system be easily evolvable. Although the realization of

separating 'software functionality' from 'integration mechanisms' was quite low in the entire research team as a whole, there were independent design decisions of some researchers based more on foresight, that they decided to make this separation in their design[1].

It is easy to see why the CES POC system 'died' in these circumstances. These new environmental requirements questioned the very design and implementation of the CES POC system. The CES POC system was out-of-date technically because it was developed on a different OS which was incompatible with that used for the rest of the suite of tools, and because integrating it into the existing suite of tools was impossible. Arthur's views [14] reported in chapter 2 are thus relevant here: *that, the existing system should be treated as a prototype and a new one built from scratch.*

The new requirement set 'R3' is as follows:

Requirement Type	New Requirements due to Environmental evolution from Sept.'92 to Sept.'94 [R3]
Quality Requirements	R3.1: The system must be so programmed that the 'software functionality' is separated from 'integration mechanisms' so as to enable easier 'tool integration'.
	R3.2: The system must be integrable with the process cycle tool-kit.
	R3.3: The system must be survivable (for a reasonable duration) in environment changes.

Table 5.1: The new requirement set 'R3'

1. Only the Generaliser and the CES POC system were designed in this way. However, there was no formal group policy to follow this approach.

5.3 Summary

The preceeding sections explained how the system requirements change continually for a system due to an evolving environment. Both the graphs (Figures 4.1 and 5.2) form a strong rationale for the requirements that are generated at a given point of time. Looking at the environment at the two time-shots of Sept.'92 and Sept.'94, we can observe how the requirements have changed from the original set R1.

Thus, the requirement set, $R_{re\text{-}imp}$, for the re-implemented system (in Sep.'94 to March'96) is defined as:

$$R_{re\text{-}imp} = R1 + R2 + R3 - R_{deleted}$$

where,

R1: the old requirement set for the CES POC system (R1.1 to R1.13)

R2: the new requirement set for the system due to CES POC deficiencies (R2.1 to R2.9)

R3: the new requirement set for the system due to environmental changes from Sept.'92 to Sept.'94 (R3.1 to R3.3)

$R_{deleted}$: the requirements for the CES POC system which were not to be satisfied in the system re-implementation

We now formally tabulate the old requirements for the CES POC system and the new requirements that emerged as at the time of system re-implementation.

Table 5.2 shows the change in system requirements from Sept.'92 to Sept.'94. The requirements are categorized into functional, non-functional and quality requirements. A requirement

belonging, say, to the set R1 is identified as R1.x where x is an integer >= 1.

Requirement Type	CES POC system Requirements in Sept.'92 [R1]	CES POC system requirements in Sept.'94 [R1 + R2 + R3 - $R_{deleted}$]
Functional requirements	R1.1: The system should facilitate the evaluation process model fitness based upon the model/context attributes and their relationships.	R1.1: The system should facilitate the evaluation process model fitness based upon the model/context attributes and their relationships.
	R1.2: The system must display the 'Trouble Spot List' for the process model and context attributes which exhibit poor fitness.	R1.2: The system must display the 'Trouble Spot List' for the process model and context attributes which exhibit poor fitness.
	R1.3: The system should assist in designing process models, i.e., given process context characteristics, it should identify the fitness of the different values of each process model attribute.	R1.3: The system should assist in designing process models, i.e., given process context characteristics, it should identify the fitness of the different values of each process model attribute.
Non-functional requirements	R1.4: The system must employ the congruence evaluation and design assistance algorithms as developed during the congruence evaluation method study.	R1.4: The system must employ the congruence evaluation and design assistance algorithms as developed during the congruence evaluation method study.
	R1.5: The system must be programmed on the MS-DOS operating system.	R1.5 DELETED
	R1.6: The system must be programmed in FoxPro programming language.	R1.6 DELETED
		R2.1: The system must be programmed on the Sun OS operating system.
		R2.2: The system must be programmed in C++/MOTIF programming language.

Table 5.2: Requirements changes for the CES POC system

Requirement Type	CES POC system Requirements in Sept.'92 [R1]	CES POC system requirements in Sept.'94 [R1 + R2 + R3 - $R_{deleted}$]
Non-functional requirements		R3.1: The system must be so programmed that the 'software functionality' is separated from 'integration mechanisms' so as to enable easier 'tool integration'.
Quality requirements	R1.7: The system must be easily understandable in its operation.	R1.7: The system must be easily understandable in its operation.
	R1.8: The system must be reliable in its operation (must repeatedly produce correct results).	R1.8: The system must be reliable in its operation (must repeatedly produce correct results).
	R1.9: The system must be concise in information displayed in screens without sacrificing understandability.	R1.9: The system must be concise in information displayed in screens without sacrificing understandability.
	R1.10: The system must exhibit consistent terminology, symbols, concepts and notations in its operation.	R1.10: The system must exhibit consistent terminology, symbols, concepts and notations in its operation.
	R1.11: The system must provide all the key features in the domain of process fitness (to the best of knowledge of the researchers).	R1.11: The system must provide all the key features in the domain of process fitness (to the best of knowledge of the researchers).
	R1.12: The system must be efficient in its operation without a waste of resources (e.g., CPU time, memory requirements, etc.).	R1.12: The system must be efficient in its operation without a waste of resources (e.g., CPU time, memory requirements, etc.).
	R1.13: The system must be validated to ensure that the congruence measures produced by the tool indeed characterize congruence.	R1.13 DELETED
		R2.3: The system must be user-programmable, i.e., it must allow the user to change the data stored in the database.

Table 5.2: Requirements changes for the CES POC system

Requirement Type	CES POC system Requirements in Sept.'92 [R1]	CES POC system requirements in Sept.'94 [R1 + R2 + R3 - $R_{deleted}$]
Quality requirements		R2.4: The system must be well documented to assist in future maintenance.
		R2.5: The system should be user-friendly and there should be 'help menus' throughout the system screens.
		R2.6: The system should be programmed in a structured language so as to aid in 'intrinsic' understandability of the system and system maintenance.
		R2.7: The system should be portable to different or compatible families of UNIX with minor changes. e.g., LINUX
		R2.8, R3.2: The system must be integrable with the process cycle tool-kit.
		R2.9, R3.3: The system must be survivable (for a reasonable duration) in environment changes.
All requirements		$R_{deleted}$: R1.5, R1.6, R1.13

Table 5.2: Requirements changes for the CES POC system

Table 5.2 shows how the requirements for the CES POC system changed from Sept.'92 to Sept.'94. This was primarily due to the fact that when the system was first implemented, the concept of process model congruence was highly original. It was *not* possible to decide all the tool features and the application domain was also unclear. Lientz and Swanson's views [25] reported in chapter 2 are thus relevant in this situation: that, requirements change continually, often from

experience gained from use of the system and in response to organizational change (which can here be likened to environmental change!). The requirement set in Sept.'94 then formed the starting point for the re-implementation of the system.

Chapter 6

System Re-Implementation

Based on the requirements identified in Chapters 4 and 5 (both due to the environment changes and due to the deficiencies of the CES POC system), in this chapter, we describe the design strategy adopted for the re-implementation of the system (during Sept.'94 to March'96). We also discuss how the environment evolved during the course of system re-implementation and what new requirements arose due to these changes. We *do not* make a formal assessment of the re-implemented CES system[1] because our focus in this thesis is on the new requirements which emerge due to environmental changes. However, the importance of making a formal assessment of the system before any future enhancements still remains.

6.1 Design Strategy and its Rationale for the Customizer

In this section, we discuss the design strategy for the Customizer and present the rationale behind it. We observed that all the explicitly documented requirements for the Customizer were the *drivers* of this design strategy and that the rationale was closely tied to the satisfaction of these requirements. Also, unlike in Chapter 3, where we discussed separately the design strategy and its

1. The re-implemented CES POC system is hereafter referred to as 'Customizer'.

rationale, here we discuss them in tandem because the requirements for the Customizer were explicitly documented and thus each design strategy component and its rationale were closely coupled with a particular requirement. It therefore seems suitable to discuss both together.

As stated in Chapter 3, when the Customizer was implemented, the focus of the design strategy was on 'evolvability' of the system while retaining its validity, i.e., the system was expected to survive (for a reasonable duration) future environment changes ('new' requirements R2.9, R3.3). The design strategy for the Customizer was thus to employ a structured programming language, which would simplify understandability of the system and ease evolvability ('new' requirements R2.2, R2.6). We decided to use the C++ programming language because other prototype systems in the lab were already being programmed in this language and the developer (the author) was also familiar with it. The operating system for the implementation was required to be compatible with other UNIX families and was to also allow for easy integrability of the system with the process cycle tool-kit ('new' requirements R2.1, R2.7, R2.8 and R3.2). We used the Sun OS because all the other prototype systems, which formed part of the process cycle tool-kit, were being developed on it.

When the Customizer was implemented, the requirements were clearly specified (see Table 5.2, Chapter 5), therefore a formal software development process (here, the iterative model) was followed. This was unlike the 'ad-hoc' approach followed in the implementation of the CES-POC system due to poorly specified requirements.

In order to assist in future maintenance activities, the system was to be well documented ('new'

requirement R2.4) with a formal software requirements specification (SRS) document, design and code documentation. The database for the Customizer was to be such so as to allow for the customizability of the data stored in the system ('new' requirement R2.3). The system was to support 'help' menus throughout its screens ('new' requirement R2.5) to simplify understandability of the system operation. In order to assist in future evolvability, we decided to separate the code pertaining to 'functionality' of the system (e.g., implementation algorithm) from that dealing with integration mechanisms ('new' requirement R3.1).

In the next section, we examine the environment changes during the course of development of the Customizer. We had noted in Chapter 5, that about certain issues, e.g., 'control integration', there was a growing realization but that this realization did not necessarily translate into new requirements for the Customizer because the awareness was not mature or detailed enough lead to new requirements. However, it was observed that the situation from Sept.'94 to March'96 changed dramatically with the continued evolution of the environment. Realization of the environmental factors was much more pronounced in March'96 than in Sept.'92 and this thus resulted in a new set of requirements as described in Section 6.3.

6.2 Environmental Evolution from Sept.'94 to March'96

The analysis method for determining the environmental changes between Sept.'94 and March'96 was exactly the same as the one adopted for determining the change in the environment between Sept.'92 and Sept.'94 (Section 5.1.2.1). The analysis results of the data gathered during this survey are shown in Figure 6.1. It is suggested that Figure 6.1 be studied along-with the questionnaire (Section B of Appendix C) so as to better understand the data analysis results.

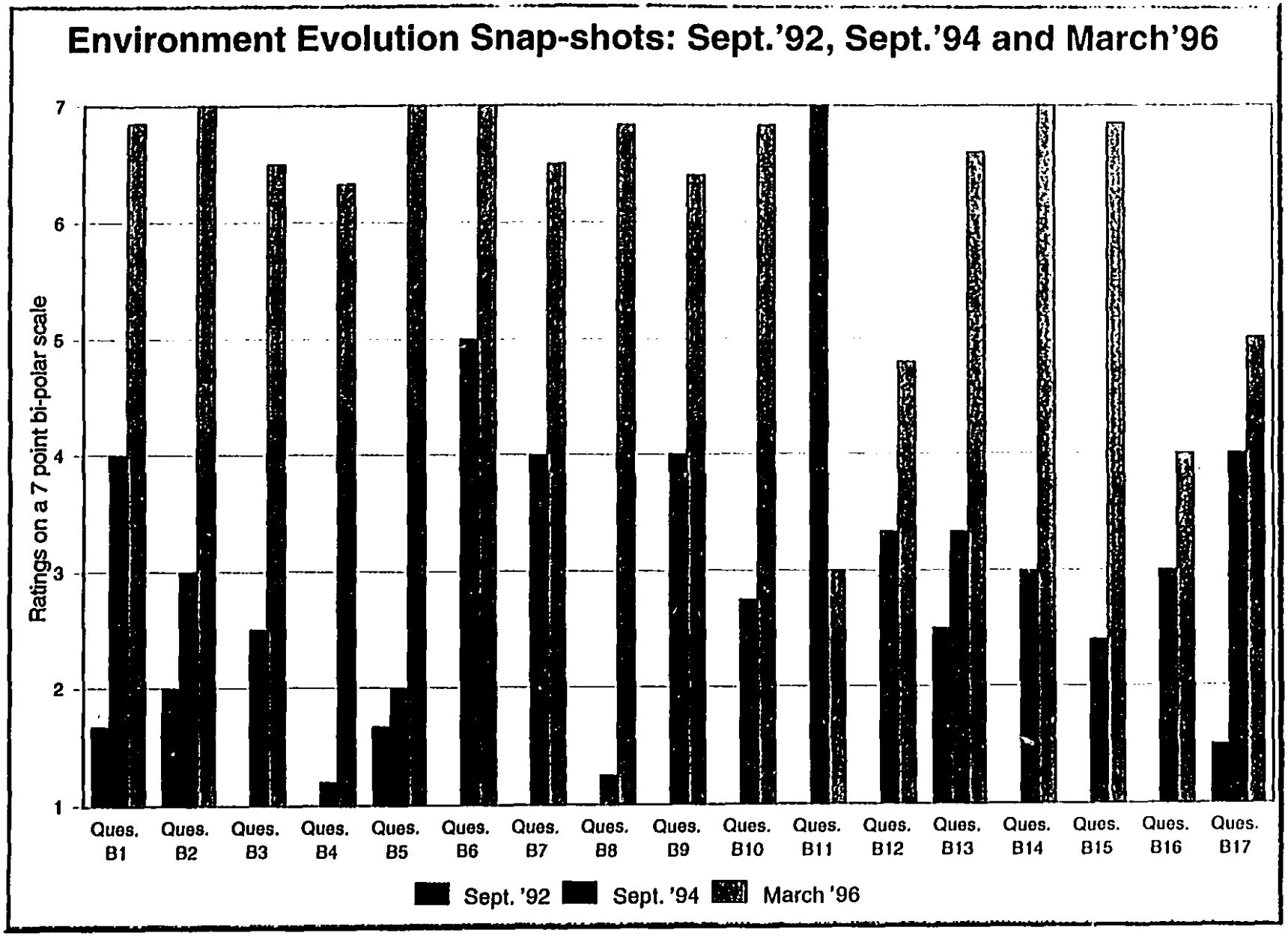


Figure 6.1: Environmental Evolution Snap-shots: Sept. '92, Sept. '94 and March '96

Figure 6.1 shows that there was a marked change in the environment from Sept.'94 to March'96. It was observed that on average, realization of the various environmental issues increased by approximately 3 scale points, which on a 7 point scale is significant! Having already analysed the environment in Sept.'94 in Section 5.1.2.2, in this section, we discuss the environment at the time of completion of the Customizer in March'96.

Figure 6.1 shows that 13 out of the 17 criteria recorded a response of 'Quite High' to 'Extremely High' realization in March'96. However, even at the time of completion of the Customizer, awareness of certain environmental issues was only 'Moderate'. These were the issues pertaining to whether the researchers realized that the existing prototypes were 'throw-away' or 'evolutionary' prototypes (Question B12), awareness of the 'robustness' of the prototype systems to survive changes in the environment (Question B16), realization that environmental changes could seriously affect the prototype systems housed in it (Question B17) and the realization of flaws in the existing environment which could cause it to evolve in the future (Question B11). Interestingly, all these questions had the same underlying reasoning to explain the response: the researchers *believe* that they are developing reasonably 'robust' systems which can survive changes in the environment in the near future, however, they *could not claim* with conviction that they had thoroughly understood the environmental changes in the present or in the future. Therefore, the responses depicted a 'cautious optimism' about the survivability of the prototype systems in the future. Also, the realization in response to Question B11 fell from a high of 'Extremely High' (in Sept.'94) to 'Slightly Low' (in March'96). This was the only environmental issue which recorded a *fall* in realization from the response in Sept.'94. Again, it only reconfirmed the fact that all the researchers, to the best of their knowledge, could not foresee any major changes in the environ-

ment in the future and thus all their present development activities were driven by this observation.

Questions B1, B6, B7, B9 and B13 recorded an average increase (compared to that in Sept.'94) in awareness of 3 scale points. All these questions were bound by one common factor and each of them merely addressed specific issues related to the same basis: the realization of the goal to integrate all the prototype systems in a process cycle tool-kit. As stated earlier, this was the driving force behind all the major environmental changes in our lab, and the awareness of the researchers on this issue has steadily increased from Sept.'92 onwards (there was already a 'Moderate' awareness in Sept.'94).

Questions B2, B3, B10, B14 and B15 recorded an average increase (compared to that in Sept.'94) in awareness of 4 scale points. Interestingly, these were the issues which addressed specific details related to the concept of process cycle tool-kit, e.g., data integration (Question B2), user-interface integration (Question B3), and other such issues. Since the awareness of such issues varied from 'Slightly Low' to 'Quite Low' in Sept.'94, the increase in awareness was more pronounced.

Finally, Questions B4, B5 and B8 recorded the highest increase in realization, up by on an average 5 scale points when compared to the data for Sept.'94! They shared a similarity to the questions which recorded a 4 scale points increase in realization from Sept.'94: these questions also addressed specific issues related to the actual accomplishment of the process cycle tool-kit. Also, all these issues were relatively 'new' and their realization in Sept.'94 was 'Extremely Low'.

To summarise, it was observed that there was a dramatic increase from Sept.'94 to March'96 about specific issues related to the formation of a process cycle tool-kit. Other questions which related to an 'on-going' understanding of more subtle issues related to environmental change exhibited a more linear growth.

In the next section, we study the impact of these environmental changes on system requirements. Specifically, we identify the new requirements (set 'R4') due to these stated environment changes. It should be noted that, some of these issues had already yielded new system requirements in Sept.'94 which remained true even in March'96. Other environmental issues, of which realization was much less in Sept.'94 had matured now to yield concrete requirements.

6.3 New Requirements due to the Environmental Evolution

Four new requirements arose in March'96 due to the environmental evolution from Sept.'94 and all these were specific to the details of establishing a process cycle tool-kit. The realization of the goal pertaining to 'data integration' (Question B2) led to the new requirement that all the prototype systems access a common data repository. The realization of the goal of 'user-interface' integration (Question B3) led to the new requirement that there be a common user-interface to load process models and to start applications, i.e. clients modules. Realization of the goal of 'control integration' (Question B4) led to the new requirement that a prototype system (in addition to 'direct' invocation) may be invocable from another prototype system too. Finally, realization of the goal of having a 'client-server architecture' (Question B8) led to the new requirement that each client will be able to connect to the same server through InterNet sockets. Thus, the components of the process cycle environment may run on separate machines.

The new requirement set 'R4' at the time of completion of the Customizer is:

Requirement Type	Customizer Requirements in March'96 [R4]
Non-functional requirements	R4.1: The system should be 'data integrated' with the other prototype systems.
	R4.2: The system should be (if deemed necessary) 'control integrated' with the other prototype systems.
	R4.3: The system should run as a client in a distributed client-server architecture.
	R4.4: The system should be 'user-interface integrated' with the other prototype systems.

Table 6.1: The new requirement set 'R4' for the Customizer

This set of requirements has been validated and would thus form a driver for system evolution in the future.

Thus, the requirement set, $R_{post\_re\text{-}imp}$, after completion of re-implemented system (March'96) is defined as:

$$R_{post\_re\text{-}imp} = R_{re\text{-}imp} + R4$$

where,

$R_{re\text{-}imp}$: $R1 + R2 + R3 - R_{deleted}$

R4: the new requirement set for the system due to environmental changes from Sept.'94 to March'96 (R4.1 to R4.4)

Note: The re-implemented system has not as yet been assessed and thus we expect some additional requirements due to system deficiencies for future release of the system.

6.4 Summary

As was observed in the development of the CES POC system, the requirements of the system change continually in response to environmental changes. No system can be 'immune' to such changes in requirements. The challenge is thus to implement the system in such a way that it affords future evolutionary changes.

All the new requirements discussed in this chapter form part of the new requirements to be considered in the enhancement of the Customizer. The existing Customizer would require minor to moderate changes to satisfy these new requirements. For example, user-interface integration is straightforward to achieve because the system functionality has been separated from integration mechanisms, though dependencies would need to be taken care of. The new requirement of a distributed client-server architecture would require server calls to be incorporated in the existing code. Similarly, to accomplish control integration, a 'call' to the other client system, at the appropriate point, would be required to be included in the Customizer code. All these changes, however, are *anticipated* and *evolutionary*, unlike the *unanticipated dramatic* changes that rendered the CES POC system defunct.

Chapter 7

Lessons Learnt

We now reflect on the case study and draw out three lessons that were learnt. These are:

❶ *Requirements for a system stem not only from system-specific deficiencies and functional enhancements but also from changes in the environment, which are not always predictable.*

(Sections 5.2 and 6.3)

The importance of the new requirement sets due to environmental evolution, the sets R3 and R4, cannot be under-estimated. We observed that as many as 9 of the 16 new requirements (specifically, requirements R2.8, R2.9, R3.1 to R3.3, R4.1 to R4.4) were introduced due to environmental changes or environment related issues! Clearly, environmental changes have a major impact in determining system requirements.

❷ *Requirements emerging due to environmental changes can have such a devastating impact on a system that they can render the system obsolete unless the system was so developed to survive these changes.*

(Section 4.2)

As stated earlier, the CES POC system suffered a *fatal* attack due to the requirement set 'R3' while the impact of the requirement set 'R4' was *evolutionary*. This was because the Customizer was so developed, so as to survive environmental changes in the near future.

❸ *Although the environment may keep on evolving all the time, these changes may not necessarily translate into new system requirements unless the changes are 'mature' enough.*

(Section 5.2)

We observed that environmental changes are sometimes subtle and that the environment evolves gradually. Therefore, such environmental changes only transform into actual requirements after a certain degree of 'maturity' in the understanding of the changes has been attained.

From our experience, the described lessons should be considered in the development of unprecedented POC systems by research teams so as to possibly avoid the fatal impact of environmental changes.

Chapter 8

Conclusions and Future Work

In this thesis, we have described a case study of requirements changes in an evolving environment involved in the development of unprecedented systems. Specifically, we have analysed how this environmental evolution had affected the requirements for, and thus the survivability of, the software system. Also, we have listed some lessons learnt from this experience which can be considered in the development of the proof of concept systems in research environments.

All these lessons suggest a circumspective analysis of the kind of criteria identified in Section B of Appendix C, at various points in time and as a system grows. Only through in-depth analysis, both qualitative and quantitative, would we be able to reason about individual and collective requirements, the source of the rest of the software development activities.

Based on the re-implemented system, we identify one requirement for future work in the short term - analysis and visualisation of the impact of change to a given process or context variable. For example, what is the impact of changing one tool for another on software productivity? One desired feature of the future system would be to identify the process or context variables (such as budget, skill level required, etc.) that would be affected by the tool change and to display contin-

gency variables (such as process and project size) and relationships that act as constraints on the tool-productivity relationship. A related feature would enable one to experiment with different variable values and visualise the impact of change. With such system features, key decisions can be made on specific issues in a given project.

Another research thrust for the future would be to monitor further evolution of the process cycle environment and its impact on requirements changes for the re-implemented system. By obtaining several cycles of evolutionary data, a pattern could emerge on how requirements emanate from environmental changes over a period of time. Such a pattern would be an important contribution to our knowledge on requirements evolution.

Appendix A

Project Characterization - I

This appendix presents the process model and context attributes (along with possible values) of one of the projects studied as part of the empirical study described in Chapter 3.

Process Model attributes	Value	Process Model attributes	Value
Exit criteria	Lower than Stand.	Existence of templates	Few
Output/Deliv. divisionalization	Yes	Existence of examples	Many
Integration mechanisms	Low	Effort examining current system	High
Devices for underst./valid. function.	Prototypes	Age of method	Old
Coordination mechanism	Stand. of deliv.	Type of modeling in PA	Functional+concep
Formaliz. of change management	Low	COTS orientation	COTS is part
Resources for PM. control & plann.	High	Workflow depiction in method	Not Obvious
Amount of Project Manag. docum.	Large	Role specialization in model	High
Emphasis on control and planning	High	Relative difference of model	Different
User's view formalism	Simple	Incremental implementation	Yes
Size of user's view of model	Large		

Characterization of Process model 1

Context attribute	Value	Context attribute	Value
Accuracy of estimates required	High	Package modification needed	No
User experience with method	Low	Use of consultants (process)	Extensive
Delegation of author. to users	SC validates	Availability of users	Available
Manag. method for personnel selec.	By capability	User wants a package	Yes
Ability to change	High	Package already purch./selected	No
IS spending	High	Technology strategy	Proc. improv.
Management expectation	Patient	Highly interactive system	No
Project size	Large	Time of package decision	Early
Application uncertainty	Low	Existence of a legacy system	
Application complexity	Low	User wants something new	No
Number of user departments involved	Few	Style of project manager	Strict control
Budgetary constraints	No	Centraliz. of decision making	Centralized
User interface	Not critical	Schedule constraints	No
Prototyping tool support	Not Available	Management Method	Deliv. oriented
Main user emphasis	Functionality	Use of consultants (any cons.)	Extensive
User's IS experience	High	Information system size	Large
User's communication skills		Reward system	Long term
Customiz. of business process	Low	PM technical involvement	Low
Scope of system	Narrow	Extent of tech. driven culture	Low
Process experience of practitioners	Large	Stand-alone package	Yes
Organization has own standards	No	User involvement	Low

Characterization of Context 1

Appendix B

Project Characterization - II

This appendix presents the process model and context attributes (along with possible values) of one of the projects studied as part of the empirical study described in Chapter 3.

Process Model attribute	Value	Process Model attribute	Value
Exit criteria	Lower than Stand.	Existence of templates	Many
Output/Deliv.divisionalization	No	Existence of examples	Few
Integration mechanisms	High	Effort examining current syste.	Low
Devices for underst./valid. function.	Prototypes	Age of method	Old
Coordination mechanism	Stand. of activ.	Type of modeling in PA	Just conceptual
Formaliz. of change management	Low	COTS orientation	COTS is part
Resources for PM. control & plann.	Low	Workflow depiction in method	Not Obvious
Amount of Project Manag. docum.	Small	Role specialization in model	High
Emphasis on control and planning	High	Relative difference of mode	Similar
User's view formalism	Simple	Incremental implementation	Yes
Size of user's view of model	Small		

Characterization of Process model 2

Context attribute	Value	Context attribute	Value
Accuracy of estimates required	Low	Package modification needed	No
User experience with method	Low	Use of consultants (process)	Not Extensive
Delegation of author. to users	Delegation	Availability of users	Available
Manag. method for personnel selec.	By capability	User wants a package	No
Ability to change	High	Package already purch./selected	No
IS spending	High	Technology strategy	Second to market
Management expectation	Patient	Highly interactive system	Yes
Project size	Large	Time of package decision	
Application uncertainty	High	Existence of a legacy system	No
Application complexity	High	User wants something new	Yes
Number of user departments involved	Few	Style of project manager	Strict control
Budgetary constraints	No	Centraliz. of decision making	Decentralized
User interface		Schedule constraints	No
Prototyping tool support	Available	Management Method	Deliv. oriented
Main user emphasis	Functionality	Use of consultants (any cons.)	Not Extensive
User's IS experience	Low	Information system size	
User's communication skills	Expressive	Reward system	
Customiz. of business process	Low	PM technical involvement	High
Scope of system	Narrow	Extent of tech. driven culture	Low
Process experience of practitioners	Large	Stand-alone package	No
Organization has own standards	No	User involvement	High

Characterization of Context 2

Appendix C

An Instrument to Assess System Deficiencies and Change in Requirements

This section contains the instrument which was used to assess the deficiencies of the CES POC system at the time of completion of the system (in Sept.'92) and measure the environmental evolution during system implementation (Sept.'92-Sept.'94) and re-implementation (Sept.'94-March-96).

AN INSTRUMENT TO ASSESS SYSTEM DEFICIENCIES & CHANGE IN REQUIREMENTS

The purpose of this instrument is to assess any deficiencies of the Congruence Evaluation System, a *proof of concept* system (CES POC system), at the time of completion of the system (Sept. 1994), and also to assess the change in requirements for this system due to the evolution of the process cycle environment (i.e., all the tools that are being built in the Software Engineering Lab. at McGill) during the course of development of the CES POC system (hereafter called "the system") and thereafter.

Determining the deficiencies of the system at the time of completion (Section A) will help us list the 'new' system requirements which can be considered during the re-implementation of the system, so as to eliminate the identified deficiencies. Also, determining the environmental evolution (Section B) during the course of development of the system will help us determine the consequent change in system requirements which was the primary cause for the re-implementation. Similarly, we would like to assess the evolution of the environment during the re-implementation and thereafter.

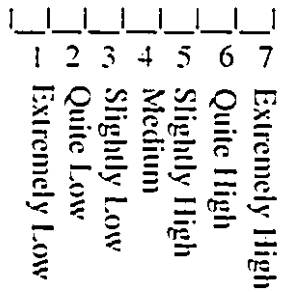
The gathered data would be used, without disclosing the identity of any individual for research purposes.

Participant Name: _____

Date: _____

Note: The need for the name is only for purposes of tracking issues and for following up on issues should it become necessary.

You will find in this questionnaire a list of criteria to measure the purposes served by the system at the time of completion of the system, and the deficiencies of the system (Section A). You will also find questions related to the environmental evolution and the consequent change in system requirements (Section B). Included with each question is a 7-point polar scale to record your response and another 7-point scale to record your 'confidence level' in answering the question:



☛ If you do not know the answer, please leave the response blank and check the Don't Know box (☒).

Example: Let us assume that we are concerned about the criteria (or construct) "Defect Quality", then:

- If you feel that the concept is 'extremely closely' characterized by one end of the scale, you should check-mark as follows:

  Extremely Low |__|__|__|__|__|__|X| Extremely High

  Extremely Low |X|__|__|__|__|__|__| Extremely High

- If you feel that the concept is 'quite closely' characterized by one end of the scale, you should check-mark as follows:

  Extremely Low |__|__|__|__|__|X|__| Extremely High

  Extremely Low |__|X|__|__|__|__|__| Extremely High

- If you feel that the concept is 'slightly closely' characterized by one end of the scale, you should check-mark as follows:

  Extremely Low |__|__|__|__|X|__|__| Extremely High

  Extremely Low |__|__|X|__|__|__|__| Extremely High

- If you feel that the concept is 'equally closely' characterized by one or the other end of the scale, or is characterized as 'neutral' on the scale, you should check-mark as follows:

  Extremely Low |__|__|__|X|__|__|__| Extremely High

SECTION A [INSTRUMENT TO ASSESS SYSTEM DEFICIENCIES]

Note: For each of the questions below, assess the criteria as at the time of system completion (Sept. 1994). It is important to give the rationale for the measure chosen; otherwise the measure is not very helpful.
Please feel free to add any other factors which you may deem fit to mention here.

A1. **Understandability:** Is the system easily understandable? (i.e., Is the purpose of the system clear? Is the system operation easy to comprehend? Exclude system design and implementation issues here.)

Not understandable |__|__|__|__|__|__|__| Understandable
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A2. **Completeness:** Does the system provide all the key features necessary in the domain of process fitness?

Incomplete |__|__|__|__|__|__|__| Complete
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A3. **Conciseness:** Is the system concise (i.e., there is no 'excess' information in user screens or in the system as a whole) without sacrificing understandability?

Poor conciseness |__|__|__|__|__|__|__| Extremely concise
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A4. **Security:** Can you damage the process model/context values and relationships and does the system warn you of such inadvertent actions?

Not secure |_|_|_|_|_|_|_| Highly secure

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A5. **Consistency:** Does the system exhibit consistent terminology, symbols, notations and concepts in its operation?

Not Consistent |_|_|_|_|_|_|_| Consistent
1 2 3 4 5 6 7

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A6. **Maintainability:** Has adequate documentation of the system been provided to assist in future maintenance (corrective, adaptive or perfective)?

Poor documentation |_|_|_|_|_|_|_| Very good documentation
1 2 3 4 5 6 7

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

A7. **Validity:** Has the system been validated with respect to the underlying concept of congruence (i.e., the congruence measures produced by the tool indeed characterize congruence)?

No validation |__|__|__|__|__|__|__| Thorough validation
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A8. **Usability:** Is the system user-friendly? (e.g., are the displays simple to understand; does the system have 'help' menus?)

Not user-friendly |__|__|__|__|__|__|__| Extremely user-friendly
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A9. **Reliability:** Is the system reliable? (i.e., does it repeatedly produce correct results?)

Not reliable |__|__|__|__|__|__|__| Extremely reliable
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A10. **Structuredness:** Has the system been developed with a high degree of structuredness (for

instance, using a highly structured language like C++ and/or structured design methods)?

Extremely Unstructured |__|__|__|__|__|__|__| Structured
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A11. **Efficiency:** Does the system fulfil its purpose without a waste of resources (for instance, CPU time, memory requirements)?

Inefficient |__|__|__|__|__|__|__| Extremely efficient
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A12. **Customizability:** Is the system user-programmable (i.e., is it possible to customize data stored in the database, for instance, can the user add or delete existing process model attributes: can the user change the process model-context relationship values?)?

Poor user-programmability |__|__|__|__|__|__|__| Highly user-programmable
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A13. **Portability:** Is the system portable to other platforms (say, UNIX)?

Not portable |__|__|__|__|__|__|__| Portable (no modifications needed)
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A14. **Integrability:** Is the system stand-alone or must it be executed as part of the process-cycle tool-kit?

Completely stand-alone |__|__|__|__|__|__|__| Extremely Integrable
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

A15. **Survivability:** Is the system 'robust' enough to survive any changes in the environment?

Not survivable |__|__|__|__|__|__|__| High survivability
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

SECTION B [INSTRUMENT TO ASSESS ENVIRONMENT EVOLUTION]

Note: For each of the questions below, assess the criteria as at the start of system implementation (Sept. 1992), at the time of system completion (Sept. 1994) and at the time of completion of re-implementation (March 1996). It is important to give the rationale for the measure chosen; otherwise the measure is not very helpful.

Please note that this section has been divided into three sub-sections: B.1, B.2 and B.3. Section B.1 comprises of questions specific to the environmental goals, section B.2 comprises of questions specific to the predictors of environmental change and section B.3 comprises of other related questions.

Please feel free to add any other factors which you may deem fit to mention here.

Section B.1 [Questions pertaining to environmental goals]

B1. Realization of the goal of having a process cycle tool-kit (i.e., 'process integration', all tools to be used to support a software development process):

Sept. 1992: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

March 1996: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B2. Realization of the goal of having 'data integration' for all the tools[1] in the process cycle environment (i.e., data is shared among different tools, e.g., by using a shared repository):

Sept. 1992: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

1. CES, X-Elicit, V-Elicit, Generaliser

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B3. Realization of the goal of having 'user-interface integration' for all the tools in the process cycle environment (i.e., all the tools can be invoked from a common user interface):

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B4. Realization of the goal of having 'control integration' for some or all the tools in the process cycle environment (i.e., a tool can be invoked through another tool):

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B5. Realization of the goal of having 'platform integration' for all tools in the process cycle environment (i.e., all tools run on the same or compatible operating system so as to allow 'inter-operability'):

Sept. 1992: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

March 1996: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B6. Realization of the goal of integrating the CES system, in particular, in the process cycle environment:

Sept. 1992: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

March 1996: No realization |_|_|_|_|_|_|_| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |_|_|_|_|_|_|_| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B7. Realization of the goal of evolving the CES system in the process cycle environment:

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B8. Realization of the goal of having a distributed, client-server architecture in the process cycle environment:

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B9. Realization of change of goals (for the entire team) from focus on 'software process concepts and methods' to 'software process concepts, methods **and tools**'?

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

Section B.2 [Questions pertaining to predictors of environmental change]

B10. Realization that software organisations will evince more interest in a 'fully integrated' (all types of integration) tool-kit than in isolated tools?

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B11. Realization of the existence of 'flaws' in the existing environment which could cause the environment to evolve in the future (i.e., could it have been predicted at any given time that there would be an imminent change in the environment in the future)?

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B12. Realization that existing prototypes were 'throw-away' prototypes rather than 'evolutionary' prototypes?

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

Section B.3 [Other related questions]

B13. Existence of a 'common vision' for the entire team (guided by the process cycle)?

Sept. 1992: Extremely Low |__|__|__|__|__|__|__| Extremely High
1 2 3 4 5 6 7

Sept. 1994: Extremely Low |__|__|__|__|__|__|__| Extremely High
1 2 3 4 5 6 7

March 1996: Extremely Low |__|__|__|__|__|__|__| Extremely High
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B14. Realization of the impact of the concept of process cycle on the actual software development activities:

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B15. Realization about the prioritization on separating code dealing with 'software functionality' from code dealing with 'integration mechanisms' (i.e., standards to facilitate different types of integration, e.g., MOTIF as a standard to support user-interface integration):

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

B16. Were the existing software systems 'robust' enough to survive any changes in the environment (this is analogous to Darwinian concept of 'Survival of the fittest'!)?

Sept. 1992: Not robust |__|__|__|__|__|__|__| Extremely Robust
1 2 3 4 5 6 7

Sept. 1994: Not robust |__|__|__|__|__|__|__| Extremely Robust
1 2 3 4 5 6 7

March 1996: Not robust |__|__|__|__|__|__|__| Extremely Robust
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

B17. Realization that any changes in the laboratory environment could seriously affect the prototype systems housed in the environment (i.e., realization that the systems were *not* immune to changes in the environment):

Sept. 1992: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Sept. 1994: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

March 1996: No realization |__|__|__|__|__|__|__| High realization
1 2 3 4 5 6 7

Confidence Level: Not Confident |__|__|__|__|__|__|__| Confident Don't Know |_|
1 2 3 4 5 6 7

Rationale: _____

_____

_____

References

[1] Aziz bin Deraman, "Requirement for a Software Maintenance Process Model: A Review", Malaysian Journal of Computer Science, vol. 8, no. 2, pp. 174-202, Dec. 1995

[2] An American National Standard IEEE Standard Glossary of Software Engineering Terminol ogy, ANSI/IEEE Standard 729, 1983

[3] Khaled El Emam and N. H. Madhavji, "A model of the factors affecting the success of a requirements engineering process", Technical report, Macroscope Project Phase III Report, Montréal, Feb.'94

[4] B. W. Boehm, "A Spiral Model of Software Development and Enhancement", ACM SIG SOFT Software Engineering Notes, 11(4): 14-24, Aug. 1986

[5] B. Curtis, "Maintaining the Software Process", In Proc. of the IEEE Conference on Software Maintenance, pp. 2-6, 1992

[6] David Alex Lamb, "Software Engineering: Planning for Change", Prentice Hall, 1988

[7] Merlin Dorfman, Richard H. Thayer, "Standards, Guidelines, and Examples on System and Software requirements Engineering", IEEE Computer Society Press Tutorial, 1990

[8] Tom Gilb, Susannah Finzi, "Principles of Software Engineering Management", Addison-Wesley Publishing Company, 1988

[9] G. Perez, K. El Emam, N. H. Madhavji, "Customising Software Process Models", In Proc. of the 4th European Workshop on Software Process Technology, Noordwijkerhout, The Netherlands, Springer Verlag, April 1995

[10] Graciela Perez, "A System for Evaluating the Fitness of Software Process Models", Masters Thesis, McGill University, Montreal, Canada, June 1994

[11] W. S. Humphrey, D. H. Kitson and T. C. Kasse, "The State of Software Engineering Practice: A preliminary report", In Proc. of the 11th International Conference on Software Engineering, pp. 277-288, Pittsburg, Pennsylvania, May 1989

[12] M. Dowson (chair), W. S. Humphrey, M. M. Lehman, and L. Osterweil, "Why is Process Important?", In Proc. of 1st International Conference on the Software Process, pp. 2-5, Redondo Beach, CA, Oct. 1991

[13] Kenneth D. Shere, "Software Engineering and Management", prentice Hall, 1988

[14] Lowell Jay Arthur, "Software Evolution: The Software maintenance Challenge", John Wiley & Sons, 1988

[15] P. J. Layzell, L. Macaulay, "An Investigation into Software Maintenance - Perception and Practices", In Proc. of the IEEE Conf. on Software Maintenance, pp. 130-140, San Diego, CA, Nov. 1990

[16] Norman F. Schneidewind, "The State of Software Maintenance", IEEE Transactions on Software Engineering, vol. SE-13, no. 3, March 1987

[17] Nazim H. Madhavji, "The Process Cycle", IEE/BCS Software Engineering Journal, 6(5): 234-242, Sep. 1991

[18] Martyn A. Ould, "Strategies for Software Engineering: The Management of Risk and Qual-

ity", Wiley Series, 1990

[19] James Martin, Carma Mcclure, "Software Maintenance: The Problem and its Solutions", Prentice Hall, 1983

[20] Roger S. Pressman, "Software Engineering: A Practitioner's Approach", McGraw Hill International, 1992

[21] Ian Sommerville, "Software Engineering", Addison Wesley, 1992

[22] Richard H. Thayer, Merlin Dorfman, "System and Software Requirements Engineering", IEEE Computer Society Press Tutorial, 1990

[23] Wilma M. Osborne, "Building and Sustaining Software Maintainability", In Proc. of IEEE Conf. on Software Maintenance, Austin, TX, Sept. 1987

[24] L. A. Belady, "Evolved Software for the 80's", Computer, vol. 12, no. 2, pp. 79-82, Feb. 1979

[25] B. P. Lientz and E. B. Swanson, "Software Maintenance Management", Reading, MA: Addison-Wesley, 1980

[26] William D. Rowe, "An Anatomy of Risk", Robert E. Krieger Publishing Co., Malabar, FL, 1988

[27] Khaled El Emam, Soizic Quintin and Nazim H. Madhavji, "User participation in the requirements engineering process: An empirical study", Requirements Engineering Journal , 1(1), Springer, 1996.

[28] D. P. Chattopadhyaya, "Environment Evolution and Values", South Asian Publishers Pvt. Ltd., New Delhi, 1982

[29] Robert N. Charette, "Software Engineering Risk Analysis and Management", McGraw Hill, 1989

[30] Anthony I. Wasserman, "Tool Integration in Software Engineering Environments", International Workshop on Environments, Chinon, France, Sept.'89

[31] W. W. Royce, "Managing the development of large software systems", Proc. IEEE Wescon, August 1970, pp. 1-9 (see also Proc. 9th Intl. Conf. on Software Engineering, 1987 (IEEE Computer Society Press) pp. 299-310)

[32] L.A. Belady and M. M. Lehman, "Programming System Dynamics or the Metadynamics of Systems in Maintenance and Growth", IBM Research Report RC 3546 T. J. Watson Research Center, Yorktown Heights, N. Y., Sep 1971

[33] L. A. Belady and M. M. Lehman, "An introduction to Growth Dynamics, Statistical Computer Performance Evaluation", Academic Press, pp. 503-511, N. Y., 1952

[34] L. A. Belady amd M. M. Lehman, "Evolution Dynamics of Large Programs", IBM Systems Journal, vol. 15, No. 3, 1976

[35] Robert B. Grady, "Measuring and Managing Software Maintenance", IEEE Software, pp. 35-45, Sep 1987

[36] R. Grady and D. Caswell, "Software Metrics: Establishing a Company-Wide Program", Prentice-Hall, Engelwood Cliffs, N.J., 1987, pp.88, 110, 123, 124 and 159.

[37] Gregory W. Jones, Software Engineering, John Wiley & Sons, 1990

[38] David Lorge Parnas, "Software Aging", Invited Plenary Talk, International Conference on Software Engineering, 1994

[39] M. M. Lehman, F. N. Parr, "Program Evolution and Its Impact on Software Engineering", Proc. of 2nd Intl. Conf. on Software Engineering, pp. 350-355, Oct 1976

[40] M. M. Lehman, "The Programming Process", IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969

[41] M. M. Lehman, L. A. Belady, "Program Evolution: Processes of Software Change", Academic Press, London, 1985

[42] F. Brooks, "The Mythical Man Month", Addison-Wesley, 1975

[43] N. H. Madhavji, K. El Emam, T. Bruckhaus, "An on-going study of factors causing process evolution", In Proc. of the Intl. Workshop on the Evolution of Software Processes, Gault Estate, Mt. St. Hilare, PQ, Canada, Jan 1993

[44] K. El Emam, N. H. Madhavji, K. Toubache, "Empirically driven improvement of generic process models", In Proc. of the 8th International Software Process Workshop, pp. 61-65, Warden, Germany, March 1993

[45] W. Fry, D. A. Smith, "Congruence, contingency and theory building", Academy of Management Review, 12(1): 117-132, 1987

[46] W. G. Egelhoff, "Strategy and structure in multinational corporations: An information processing approach", Administrative Science Quarterly, 27, pp. 435-458, 1982

[47] R. Rumelt, "Strategy, structure and economic performance", Boston, Harvard University Press, 1974

[48] C. Perrow, "Organizational analysis: A sociological view", Belmont, CA: Wadsworth publising, 1970

[49] J. Woodward, "Industrial Organization: Theory and practice", London: Oxford University Press, 1965

[50] K. R. Andrews, "The concept of corporate strategy", Homewood, IL, Irwin Publishing, 1971

[51] R. E. Miles, C. C. Snow, "Organizational strategy, structure and process", New York, McGraw Hill, 1978

[52] R. E. White, R. G. Hamermesh, "Toward a model of business unit performance: An integrative approach", Academy of Management Review, vol. 6, pp. 213-224, 1981

[53] J. Gaibraith, D. Nathonson, "Strategy implementation: The role of structure and process", New York, West publishing, 1978

[54] J. L. Kerr, C. C. Snow, "Corporate strategiew and rewards: A conceptual framework", Paper presented at the meeting of the Academy of Management, Detroit, Aug. 1980

[55] M. S. Salter, "Taylor incentive compensation to strategy", Harvard Business Review, 51(2): pp. 94-102, 1973

[56] D. A. Scheilenberg, "Strategy implementation: The effects of congruence between strategy, structure and reward systems on performance", Doctoral dissertation, Indiana University, 1983

[57] J. Nunnally, "Psychometric theory", McGraw Hill, 1967

[58] R. Veryard, "Implementing a methodology", In Information and Software Technology, 29(9): pp. 469-474, Nov. 1987

[59] Richard E. Zultner, "TQM for technical teams", Communications of the ACM, vol. 36, no. 10, Oct. 1993

[60] W. Hansen, "Creation of hierarchic text with a computer display", Ph. D. thesis, Comp. Sci. Dept., Stanford University, Stanford, CA, June 1971

[61] Tim Teitelbaum, Thomas Reps, "The Cornell Program Synthesizer: A Syntax Directed Programming Environment", Coomunications of the ACM, vol. 24, no. 9, Sept. 1981

[62] Leon Osterweil, "Software processes are software too", Communications of the ACM, 1987

[63] C. Osgood, G. Suci, P. Tannenbaum, "The measurement of meaning", Univ. of Illinois Press, 1967

[64] Barry W. Boehm, John R. Brown, Hans Kaspar, Myron Lipow, Gordon J. MacLeod and Michael J. Merritt, "Characteristics of Software Quality", TRW Series of Software Technology, Noth Holland Publishing, 1978

[65] Khaled El Emam, Nazim H. Madhavji, "Measuring the success of requirements engineering processes", Proc. of the Second IEEE International Symposium on Requirements Engg, pp. 204-211, York, England, March 1995

[66] V. R. Basili, A. J. Turner, "Iterative enhancement: A practical technique for software development", IEEE Transactions on Software Engineering, vol. SE-1, no. 4, pp. 390-396, Dec 1975

[67] L. A. Belady, M. M. Lehman, "A model of large program development", IBM Systems Journal, 15, 3 (1976), pp. 225-252

[68] Warren Harrison, Curtis Cook, "Insights on improving the maintenance process through software measurement", IEEE Conf. on Software Maintenance, pp. 37-45, San Diego, CA, Nov. 1990