ANALYZING THE IMPACT OF CHANGING SOFTWARE REQUIREMENTS: A TRACEABILITY-BASED METHODOLOGY

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
James Steven O'Neal
B.S., Mississippi College, 1987
M.S., Clemson University, 1989
December 2003

Acknowledgments

The completion of this dissertation was only possible with the contributions of many people. Foremost is the generous support, guidance, and patience of my advisor, Dr. Carver, who brought forth this research and allowed me to extend my education beyond the formal studies leading to this dissertation. Thanks to Dr. Kraft for his mentoring, for his encouragement, and for recommending the papers that sparked a fuzzy way of thinking. I also thank Dr. Iyengar, Dr. Kak, and Dr. Lou for serving as members of my doctoral committee.

A special acknowledgment and thank you is given the members of the software engineering class who participated in the experimental application of my ideas. I am truly indebted to them for their extra work during that semester.

Finally, behind the scenes are my friends and family who were always there for me during my doctoral studies. I am beholden to you all for your fellowship, reassurance, and support. This is especially true of my niece and nephews, who simply allowed me to share in their play.

Table of Contents

List of Tables

List of Figures

Abstract

Software undergoes change at all stages of the software development process. Changing requirements represent risks to the success and completion of a project. It is critical for project management to determine the impact of requirement changes in order to control the change process. We present a requirements traceability based impact analysis methodology to predictively evaluate requirement changes for software development projects. Trace-based Impact Analysis Methodology (TIAM) is a methodology utilizing the trace information, along with attributes of the work products and traces, to define a requirement change impact metric for determining the severity of a requirement change. We define the Work product Requirements trace Model (WoRM) to represent the information required for the methodology, where WoRM consists of the models Work product Information Model (WIM) for the software product and Requirement change Information Model (RIM) for requirement changes. TIAM produces a set of classes of requirement changes ordered from low to high impact. Requirement changes are placed into classes according their similarity. The similarity between requirement changes is based on a fuzzy compatibility relation between their respective requirement change impact metrics. TIAM also identifies potentially impacted work products by generating a set of potentially impacted work products for each requirement change. The experimental results show a favorable comparison between classes of requirement changes based on actual impact and the classes based on predicted impact.

Chapter 1
Introduction

Software undergoes change at all stages of the software life cycle. That is, changes to requirements may occur at the requirements definition phase, requirements specification phase, design phase, implementation phase, and maintenance phase. Managing changes to a software product is frequently critical to the success of the product [Glass, 1998]. Accepting too many changes will cause delays in the completion of product, whereas failure to implement critical changes can affect the success of the product. According to Brooks, "Clearly a threshold has to be established, and it must get higher and higher as development proceeds, or no product ever appears," [Brooks, 1995]. Additionally, the expense of implementing changes in a software product becomes greater with each subsequent phase in the software life cycle [Boehm and Papaccio, 1988]. To effectively manage change in software development projects, methods are required to provide information about changes, such as how will the change impact a development schedule or what changes will have the greatest impact on a product. With information about changes, appropriate planning can be performed by project management for implementing or deferring changes.

Software development frequently starts with a customer that has a need that a software product may satisfy. Initial discussions with the customer and members of a software development group will usually yield a requirements definition document. The requirements definition document records the requirements of the software system in a manner that is understandable to the customer and to the designers of the development group. Each requirement in the definition should be

identified in such a manner that the requirement can be referenced by subsequent development work or by future changes in the requirements. The requirements definition document details the needs of the customer and related details of the customer's processes that the software system is to satisfy. This document serves as the mechanism for discussing the goals of the software system with the customer and may function as part of a contract between the customer and the software development group [Andriole, 1998].

The next step of development is the requirements specification. While the requirements definition document defines the requirements usually in natural language, the requirements specification document defines the requirements using more technical language the developer can use. The requirements specification should be unambiguous, therefore formal languages and other formalized techniques are often used in the requirements specification. Formal languages and techniques provide a strict or more exact model of the requirements definition to eliminate any ambiguity. Since formal languages may not be understood by the client, each individual requirement specification should be traceable to a corresponding requirement definition to facilitate future discussions of the requirements.

After the requirements specification is completed, the design of the software system begins. For most large software projects, a high level design of the system, called the architectural design, is initially performed. The architectural design is the first step in deciding how a product will be structured into modules to implement the requirements. Detailed design further defines the internal structures of modules and the interactions between modules. The detailed design of the product defines how the product is to work.

The design is then implemented, or coded, in a programming environment. The source code is organized into software modules according to the software architec-

ture. Test cases are then written to verify the basic operations of the modules. The software modules are the source code of the software product. Individual testing of each module is called unit testing.

Software modules are integrated to form the complete software product. Verification must be done at this point to ensure the modules interact as designed. This testing is called integration testing. Testing is also performed to verify that the product modules are assembled correctly.

When the product has passed integration testing, the functionality of the product must be tested and demonstrated. Function testing uses test cases that validate that the requirements have been satisfied in the complete product. Successful function testing marks production of a validated software product, ready for customer acceptance testing.

After the requirements definition, requirements specification, architectural design, detailed design, implementation, and integration phases, the product is validated to meet the needs of the customer. The product then moves into the operational phase where the product is used by the customer.

Most products will undergo changes after they become operational. Frequently, these changes will be incorporated into a new development cycle that uses the existing product as the base. These subsequent development cycles create new versions of the product that supersede the previous version. Changes for a new release can either correct or enhance the product. Frequently, it is not possible to include all needed changes and still release the next version in a timely manner. The decision as to what changes are incorporated into a release is a trade-off between development time and changes critical to the success of the product. Some changes may be saved for a future release of the product in order to deliver a version of the product that includes more important changes in a timely manner.

The requirements definition, specification, design, implementation, and operational phases are the general software development phases. The exact manner in which these phases are executed depends on the development model.

The most straight forward development model is the waterfall model [Royce, 1970]. Each phase of development is completed before the next phase is started, as shown in Figure 1.1. In this model the entire product is delivered at the end of development to the customer. The benefit of the waterfall model is the delineation of development activity into the separate phases, each with clearly defined outputs. However, the model is fairly rigid, requiring the activity of each phase to be completed before moving to the next phase.

The incremental life cycle model delivers successive builds of the product to the customer, until the entire product is completed [Schach, 1999]. Once the requirements definition, requirements specification, and architectural design are completed, multiple iterations of detailed design and implementation occur, where each iteration's build adds a subset of the functionality of the product until the complete functionality is delivered. This model provides the customer with a view of the product before complete functionality is delivered. Developing the product by this model requires the product to be designed in a manner that each build easily includes the newly completed functionality. This model often results in a product with an open architecture that facilitates the addition and modification of functionality. Figure 1.2 shows the incremental model.

To address the risks associated with major software products, the spiral life cycle model incorporates strict verification and risk analysis tasks between each phase [Boehm, 1988]. The risk analysis is performed to determine the objectives, alternatives, and constraints at each phase. Then, risks are identified, alternatives are evaluated, and risk resolution is attempted. If the risks cannot be resolved,

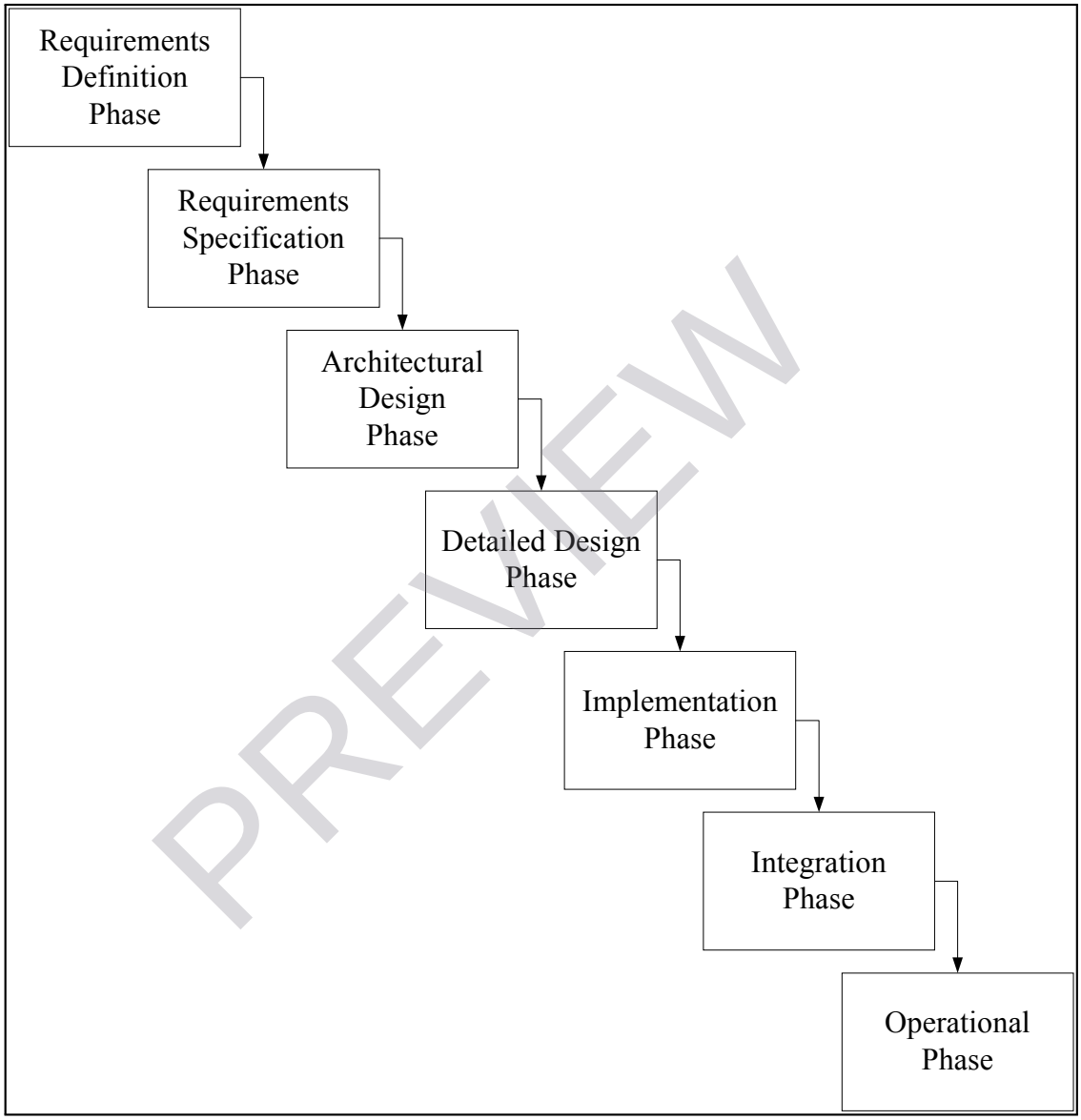

FIGURE 1.1. Waterfall life cycle model

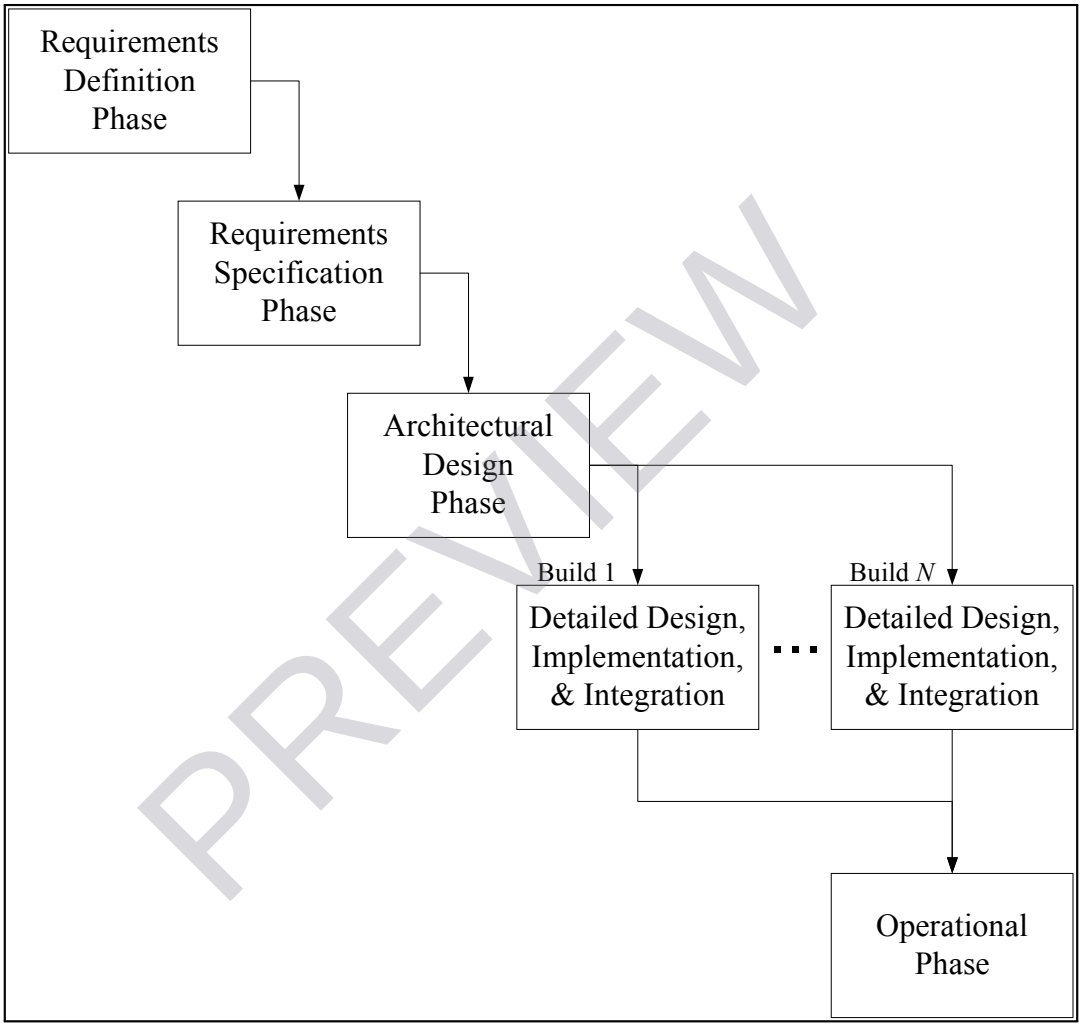

FIGURE 1.2. Incremental life cycle model

the project can be ended at that point, saving additional expense on the project. Failure to adequately manage risks associated with software development is one reason for large cost overruns in some software projects [Boehm, 1988].

The development of object oriented software is highly iterative, with the products in each phase are refined on each iteration. This practice has been described in the fountain life cycle model [Henderson-Sellers and Edwards, 1990]. The phases of the software life cycle overlap. Some activities performed in the phases are performed in parallel. Often discoveries in a latter phase require the revisiting of activities performed in an earlier phase.

In practice, variations of these models are used to fit the needs and the experience of the development group and customer. Two emerging models, extreme programming [Beck, 1999] and synchronize and stabilize [Cusumana and Selby, 1997], are examples of models created for specific types of projects and corporate cultures. In the extreme programming model, the development team selects a small set of requirements that are related to a feature, or story, to implement. The product is rapidly built to support this story and released to the customer. This process repeats until all the features required by the customer are implemented. Adaptability to vague and changing requirements is a key feature of the extreme programming model.

The synchronize and stabilize model, used by development organizations within Microsoft, is also similar to the incremental model. In this model, as soon as a feature is implemented, the code is integrated into the product during the daily build. Testing and debugging is performed after each daily build. Periodically the product is stabilized by halting the addition of features and working to remove the remaining flaws in the product. The synchronize and stabilize model allows

concurrent testing and program development along with the ability to incorporate customer feedback during development.

Regardless of the type of model used, the completed software product includes product documentation, source code, test suites, as well as the executable product. The product documentation is a collection of the documentation used in development of the product, from requirements through design to source code documentation. This information is important for future changes to the product so that appropriate analysis and caution is used when making a change to the product [Basili, 1990]. A developer making a change to a product that is being used by a customer should understand the design and implementation of the product. There are several reasons to make sure a developer understands the architecture and design of the product in sufficient detail. One is to ensure that the change is completely implemented within the product. Another is to prevent new faults from being introduced in the product as a result of changes in the product. For example, when an interface is changed between two modules, if other modules that use the called module are not changed to reflect the new interface, new faults would be introduced into the product. One other reason for the importance of attention to the product documentation is to make sure that the architecture of the product is not compromised and future maintenance made more difficult. An example could be a developer adding code to handle a unforeseen exception condition and not utilizing a sophisticated exception processing facility designed into the product that should have been invoked.

1.1 Managing Software Changes

The ability to manage change during software product development is critical for completing the product and satisfying the needs of the client requesting the prod-

uct. According to Brooks, changeability is one of the essential difficulties of software production [Brooks, 1987]. Software is frequently the most malleable component of a system, and therefore it is most likely to be changed. If changes to a software development project are not restrained, expectations for meeting estimates are not realistic. However, changes must be allowed so that the product satisfies evolving needs of the client [Kotonya and Somerville, 1998].

In the early stages of development during requirements definition, a set of requirements is created that identifies the needs of the client. During product development, changes to this set of requirements may be needed. These requirement changes are modifications to existing requirements or new requirements that may or may not affect existing requirements. It is naive to expect that no changes would be made after requirement specification. IBM's Santa Teresa Laboratory reported that an average of 25% of the requirements for an average project will change before completion of the project [Boehm, 1981]. A number of reasons exist for requirements changes, including: changes in customer needs or wants, clarification of requirements, changes in the target operating environment, and correcting errors in requirements [Bohner, 2002].

Requirements changes must be considered so that the product will satisfy the needs of the client; however, not all changes are equal [Bach, 1999]. Some changes may be critical to the success of the product, whereas some changes may be optional and thus should be included in a particular version of the product only if time allows. The ability to determine the risks that the change has on the potential to complete the product within established schedules represents a key aspect of project management [Kotonya and Somerville, 1998].

It is vitally important that project managers have information available to enable appropriate decisions to be made with respect to changes introduced during

product development. In general, management has three choices when present a change. First, the change can be incorporated in the current product development. This may require more resources to be allocated to the project and cause delays in delivering the product. Second, the change can be deferred from current project, to be included in the product at a future date. Lastly, the change can be rejected.

In [Jones, 1995], creeping requirements, which includes changes to current requirements, are identified as one of the top ten factors associated with the success or failure of a software project. Projects that fail to control changes to requirements and quantify the impact of changing requirements imbibe a risk to the successful completion of the project. Since requirement changes can be costly, frequently a formal business case must be made to justify the requirement change in a software project [Maciaszek, 2001].

Impact analysis is used in many different forms to manage changes to software. Impact analysis, as defined by Arnold and Bohner, "is the activity of identifying what to modify to accomplish a change, or of identifying the potential consequences of a change," [Arnold and Bohner, 1993]. There are several areas of research that are investigating how to measure the impact of change. Some techniques are source code based. The research using this approach focuses on the dependencies that exist in the source code in order to determine what program elements may be affected by a change. Various types of program dependency techniques have been proposed, including control and data dependency [Podgurski and Clarke, 1990; Loyall and Mathesen, 1993] and program slicing [Horwitz *et al.*, 1990; Chen *et al.*, 1996]. A second research direction uses traceability relationships between all work products or documents created during development. Bianchi, et.al., use several types of traces to predict impacted components of a software system [Bianchi *et al.*, 2000]. Other traceability approaches view work products as documents, incorporating traceabil-

ity as hypertext links or as interfaces between documents [Garg and Scacchi, 1990; Horowitz and Williamson, 1986]. These approaches attempt to identify what may be modified to make a change.

One problem resulting from incorporating changes in a software product after the requirements definition is that the change may only be reflected in work that has yet to be completed. When this occurs, requirements, design documents, and other product documentation become incorrect and outdated. Product documentation that does not reflect the actual product makes product maintenance difficult as the product source code must be used as the arbitrator of what the product does. Unfortunately in practice, developers rarely update existing documentation due to limited resources. As noted by Lindvall and Sandahl, experienced developers usually assume that the documentation is outdated and rely mainly on source code when making changes[Lindvall and Sandahl, 1998a]. The appropriate implementation of changes includes the updating of the product specification and design to reflect changes to the source code of the product [Maciaszek, 2001]. It is important to include the additional work that keeps the product documentation current as part of impact analysis.

1.2 Change Scenarios

To emphasize the need to manage changes, we present the following scenarios. The first scenario involves management's need to determine the impact of planning for a more flexible information technology infrastructure, specifically allowing a choice of database management systems. The second scenario describes a policy change that must be incorporated into the supporting software application.

The first scenario involves a company that is developing a new inventory control system. The company's current information systems are based on a single vendor's

database management system (DBMS). Since the current software development organization is experienced with using the existing DBMS, it is a requirement for the new inventory control system to use the proprietary database interface to the existing DBMS. Management has decided to investigate minimizing the risk of depending on the vendor of the existing DBMS. A standard is evolving to uncouple database applications from specific DBMS, called Open Database Connectivity (ODBC). Since the existing DBMS has an ODBC interface, management wishes to know if the new inventory control system under development can be changed to use the ODBC interface rather than the proprietary database interface. Implementing this change would allow the company to choose a new DBMS in the future without significant changes to the inventory control system.

Determining the impact of the ODBC migration scenario depends on how far the development of the inventory control system has progressed and the architecture of the inventory control system. If the development of the system has not reached the design phase, the impact of the change on the product should be minimal, since there is nothing to change except for requirements documents. However, if the system has been designed and possibly source code modules completed, then any design documents and source code modules including a database interface would be impacted and require modification. If the architecture of the inventory control system isolates the DBMS access from the business application functions, the impact would be less than if many modules implementing application functions independently access the DBMS.

Impact analysis of this scenario should be able to give information on what may be impacted and by to what degree, if the change was to be made, without regard to what point development has progressed on the product or dependent on the product architecture. Management would use this information to decide if it was

cost effective to migrate to ODBC at this time or defer the migration to a later date. Management may determine that the costs are to high to migrate at this point, or they may find that the costs of the migration would be offset by the savings incurred by using another vendor's DBMS.

The second scenario involves a university. To enroll in classes that have prerequisite courses, a student must have passed the prerequisite courses with the required grade or the student is automatically withdrawn from the course at the start of the term. A student is allowed to retake a prerequisite course only twice. The university's enrollment system implements this restriction. A new policy has been made that removes the limit to the number of times a student may retake a course, but the student must have approval from the dean of the college in which the course is offered when enrolling in a course after the second time.

In this scenario, the change must be implemented. Management of the university enrollment system requires information from impact analysis to determine the extent of the impact in order to schedule resources for implementing the policy change. Additionally, impact analysis should be performed after changes have been implemented to ensure that no other restrictions in the enrollment system are affected.

1.3 Research Objective

The research objective is to develop a predictive impact analysis technique that identifies classes of requirements changes that have similar impact levels. By predicting the impact that requirement changes may have, the effects of making a requirement change can be compared to other requirement changes with respect to the predicted effort to implement the change. This information can be used

as a criterion in a process that selects which changes can be implemented within schedule constraints.

We develop formal models and a general methodology for applying requirement traceability to impact analysis. Next, we develop a specific instance of the general methodology to use for impact analysis. Finally, the specific instance of the methodology is applied to a software development project to evaluate capability of the methodology for impact analysis.

The general methodology for using requirement traces for impact analysis includes the definition of formal models that represent a software development project and a set of requirement changes. The models include information on attributes of the work products and traces. The attributes are complexity, effort, phase, and influence. The methodology defines an impact metric that is directly related to the effort predicted to implement a requirement change. With these results the requirement changes are grouped into classes of changes that have similar impact.

The specific instance of the impact analysis general methodology defines the attribute levels that are used to describe the work products and traces. The instance is applied to a software development project in which the actual impact with respect to effort is documented. The actual impact information is used to classify changes into groups with similar impact. These results are compared with the results from our impact analysis methodology.

1.4 Motivation

This research is motivated by the continuing need to increase the efficiency of software evolution. Previous work has focused on impact analysis for maintenance changes to completed software product using traceability. The use of traces between work products is an appropriate basis for impact analysis for software prod-

ucts [Strens and Sugden, 1996]. Research providing impact analysis for software products during development is needed [Zave, 1997]. The traceability approach does not depend on the internal structure of the work product, therefore is suitable to apply to a software project that is in progress, where work products may not be developed to the degree that would allow the internal structure to be used for any analysis. Also, previous work has focused on the work products that may be impacted but not on the effect of the change on resources.

Further, to improve software evolution we need to be able to evaluate a set of proposed changes to software product in order to help assess the degree of impact a change may have on the software project with respect to the other proposed changes [Nuseibeh and Easterbrook, 2000]. This information would allow management of the project to initially determine potential impacts on resources and what changes may require significant changes in resources to implement.

1.5 Summary

Requirement changes during the development of a software product threaten the timely completion of the product and the proper documentation of the product. The decision to incorporate changes during development should allow for the modification of work products that have been completed so that all documentation is correct and so that those who perform future maintenance will not be dependent solely on the source code to determine what the product does.

This research determines the impact that a change has on existing work for a software product. It derives a comparative relationship of severity of impact between changes, and evaluates changes that are modifications to existing requirements or new requirements that affect existing requirements. The evaluation is a prediction of the amount of effort required to modify existing work to make the