Capabilities Engineering
Promoting Change-Reduction and Constructing Change-Tolerant Systems

by

Ramya Ravichandar

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Dr. James D. Arthur, Chair

Dr. Shawn A. Bohner
Dr. Robert P. Broadwater
Dr. Manuel A. Pérez-Quiñones
Dr. David P. Tegarden

May 7, 2008
Blacksburg, Virginia

Keywords: Capabilities, Capabilities Engineering, Complex Emergent Systems, Requirements Engineering, Change Tolerance, Software Evolution

Copyright © 2008, Ramya Ravichandar
Capabilities Engineering

Promoting Change-Reduction and Constructing Change-Tolerant Systems

Ramya Ravichandar

(ABSTRACT)

We propose a Capabilities-based approach for constructing complex emergent systems such that they are change-tolerant, and the development effort promotes change-reduction. The inherent complexity of software systems increases their susceptibility to change when subjected to the vagaries of user needs, technology advances, market demands and other change inducing factors. Despite the inevitability of change, traditional Requirements Engineering strives to develop systems based on a fixed solution. This is a mostly unsuccessful approach as evidenced by the history of system failures. In contrast, we utilize Capabilities — functional abstractions that are neither as amorphous as user needs nor as rigid as system requirements — to architect systems to accommodate change with minimum impact. These entities are designed to exhibit desirable characteristics of high cohesion, low coupling and balanced abstraction levels.

Capabilities are generated by a two-phased process called Capabilities Engineering. Phase I mathematically exploits the structural semantics of the Function Decomposition graph — a representation of user needs — to formulate change-tolerant Capabilities. Phase II optimizes these Capabilities to conform to schedule and technology constraints. Results from an empirical evaluation of a real-world Course Evaluation System indicate, with statistical significance, that a Capabilities-based design is more change-tolerant than a requirements-based design. In addition, we observe that the use of the CE process inherently reduces change, otherwise generated, during the regular development effort. Empirical analysis on the change-requests of Sakai, a complex emergent system, supports this claim. Finally, we observe that the process of Capabilities Engineering assists in pre-requirement specification traceability by bridging the complexity gap between the problem and solution spaces.
Now I myself, Phaedrus, am a lover of these processes of division and bringing together, as aids to speech and thought; and if I think any other man is able to see things that can naturally be collected into one and divided into many, him I follow after and “walk in his footsteps as if he were a god” . . .

- Socrates
Dedication

Amma and Appa, my loving parents

&

Ramakant, my dear husband
Acknowledgements

Dr. Arthur, my advisor, is inspirational as a teacher, as a mentor, and as a human being. Today, I see his influence on my writing, my research process, and my perception of life. I will always be grateful for his guidance, and will remember that it is a privilege to work for such a wonderful advisor.

I am grateful to my committee members: for Dr. Bohner’s advice that spanned from work-life balance to impact analysis, for Dr. Broadwater’s expertise and his insightful courses, for Dr. Tegarden’s business perspectives and valuable comments, and for Dr. Perez’s encouragement and research guidance.

My colleagues, and friends have sustained me through my temperamental research storms - be it the spring of exciting discoveries, or the forlorn winter of the writer's block. Amine Chigani, has been a source of conversation, cookies, and camaraderie, all of which I believe, are essential to enliven the monotony of human life. I thank Boby George for being the willing wall to bounce off my research ideas, Elena Rusticelli for proving that a doctoral degree can help change the world (she works for the United Nations), and Pardha Pyla for the deliberately instigated but thoroughly enjoyable Usability-Software Engineering wars.

My research progress has been lovingly monitored by my periamma, and periappa from Pittsburgh. As my surrogate parents in this second home, I am ever grateful for their support. I thank my in-laws for their constant support, and encouragement throughout this journey.

For a man who believes his daughter is going to create history, this work is a small step in that direction. My father’s passion, ambition, resilience, creativity, and charisma, are qualities I hope to have inherited. My mother is my lifeline, my strength, and my joy. Without her guidance I would be lost in this maze of life. My brother Raghav’s humor, antics, and blog posts have more than amused me through times of stress.
My journey of doctoral research began with the encouragement of a single person - my husband, Ramakant Komali. I am thankful for his unfailing support, judicious counsel, and calming influence that have anchored me through this process. I cherish these years of going to school with my better half.

I have received much support throughout my years here at Virginia Tech from so many different people; I thank you all. Lastly, thatha and avva, where ever you are, I hope you are proud of me.
Contents

1 Introduction 1
  1.1 Motivation ....................................................... 2
  1.2 Problem Statement ............................................. 3
    1.2.1 Issues .................................................. 5
  1.3 Solution Approach ............................................. 7
  1.4 Document Organization ....................................... 9

2 Background 11
  2.1 Process-Centric Approaches .................................. 12
    2.1.1 RE Process Improvement ................................. 13
    2.1.2 Design .................................................. 13
  2.2 Specification-Centric Approach ................................ 15
    2.2.1 Performance Based Specifications ....................... 15
    2.2.2 Capabilities Based Acquisition ......................... 15
  2.3 Capabilities Engineering .................................... 17

3 Capabilities Engineering Process 18
  3.1 Function Decomposition Graph ................................ 21
  3.2 Rationale for Capability Definition: The Three Criteria .. 24
    3.2.1 Cohesion Measure ...................................... 28
    3.2.2 Coupling Measure ...................................... 32
    3.2.3 Abstraction Level ...................................... 37
3.3 Capability Identification Algorithm

3.3.1 Slices

3.3.2 Algorithm

3.4 Conclusion

4 Empirical Validation

4.1 Improving Change Tolerance: Course Evaluation System

4.1.1 Hypothesis

4.1.2 Experimental Setup

4.1.3 Change Scenarios

4.1.4 Analysis of Results

4.2 Change Reduction: Sakai Experiment

4.2.1 Hypothesis

4.2.2 Experimental Setup

4.2.3 Analysis of Results

5 Pre-Requirement Specification Traceability: Bridging the Complexity Gap through Capabilities

5.1 Introduction

5.2 Background

5.3 Role of CE in Pre-RS Tracing

5.3.1 The Problem Space

5.3.2 The Transition Space

5.3.3 The Solution Space

5.3.4 Connecting Spaces

5.4 Traceability Challenges

5.4.1 Supporting Evolution

5.4.2 Link Semantics

5.4.3 Scalability
5.4.4 Human Factors ................................................. 93
5.4.5 Cost Benefit Analysis ........................................ 94
5.5 Conclusion ...................................................... 94

6 Conclusion .......................................................... 96

6.1 Main Contributions .............................................. 97
6.1.1 Capabilities Engineering ..................................... 97
6.1.2 Capability Identification Algorithm ....................... 97
6.1.3 Function Decomposition Graph ............................ 98
6.1.4 Measures ...................................................... 98
6.1.5 Pre-Requirement Specification Traceability .............. 98
6.1.6 Requirement classification .................................. 99
6.1.7 Optimal Scheduling Sequence .............................. 99
6.1.8 Empirical Analysis .......................................... 99
6.1.9 Model Driven Architecture ................................ 99
6.2 Future Work ..................................................... 100
6.2.1 Alternative / Additional Mathematical Characterization(s) of Capabilities ........................................... 100
6.2.2 Establishing the Paradigm Independence of Capabilities ......................................................... 107
6.3 Summary ......................................................... 110

Bibliography ........................................................ 119

A Sakai Experiment ................................................ 120
A.1 Capability Identification ....................................... 120
A.2 Change-Request Analysis ..................................... 125
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Capabilities Engineering Process</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>FD Graph of Course Evaluation System</td>
<td>25</td>
</tr>
<tr>
<td>3.3</td>
<td>Undirected FD Graph of Course Evaluation System</td>
<td>35</td>
</tr>
<tr>
<td>3.4</td>
<td>Example of Common Functionality Scenario</td>
<td>39</td>
</tr>
<tr>
<td>3.5</td>
<td>Example of the No Common Functionality Scenario</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>High Level Design based on CE</td>
<td>56</td>
</tr>
<tr>
<td>4.2</td>
<td>High Level Design based on RE</td>
<td>56</td>
</tr>
<tr>
<td>4.3</td>
<td>Change-Request Repository</td>
<td>65</td>
</tr>
<tr>
<td>5.1</td>
<td>Transition from Needs to Requirements</td>
<td>75</td>
</tr>
<tr>
<td>5.2</td>
<td>Problem Space</td>
<td>79</td>
</tr>
<tr>
<td>5.3</td>
<td>Transition Space</td>
<td>83</td>
</tr>
<tr>
<td>5.4</td>
<td>Solution Space</td>
<td>86</td>
</tr>
<tr>
<td>5.5</td>
<td>Capabilities Engineering in terms of Spaces</td>
<td>90</td>
</tr>
<tr>
<td>A.1</td>
<td>FD Graph of Sakai</td>
<td>121</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Relevance Values .................................................. 30
3.2 Slices Ranked in top 10 (either cohesion or coupling) of Course Evaluation System ............................................. 52
4.1 # Impacted Classes in RE/CE-based design .................. 58
4.2 Example: Missing Information Category ....................... 68
4.3 Example: Modularization Category ............................... 68
4.4 Example: Unstructured Process Category ....................... 69
4.5 Potential Causes- Summary ........................................ 69
4.6 Percentage of Change Requests Caused by RE/CE Process ................................................................. 70
5.1 Examples in the context of a Course Management System ................................................................. 80
5.2 Connecting the Spaces ................................................ 89
A.1 Slices Ranked in top 10 (either cohesion or coupling) of Sakai system .................................................. 123
A.2 Candidate Slices of Sakai system .................................. 124
A.3 Sample Sizes .......................................................... 126
A.4 Sample 1 .............................................................. 126
A.5 Sample 2 .............................................................. 127
A.6 Sample 3 .............................................................. 127
A.7 Sample 4 .............................................................. 128
A.8 Sample 5 .............................................................. 129
A.9 Sample 6 .............................................................. 130
A.10 Sample 7 ........................................................................................................... 131
A.11 Sample 8 ........................................................................................................... 132
A.12 Sample 9 ........................................................................................................... 133
Chapter 1

Introduction

The widespread advancements in technology have encouraged the demand for large-scale problem solving. This has resulted in substantial investments of time, money, and other resources in complex engineering projects such as hybrid communication systems, state-of-art defense systems, and technologically advanced aeronautics systems. Unfortunately, the expenditures are belied by the failures of such systems. Plagued by evolving needs, volatile requirements, market vagaries, technology obsolescence, and other factors of change, a large number of projects are prematurely abandoned or are catastrophic failures [1] [2] [3]. The inherent complexity of these systems, compounded by their lengthy development cycles, is further exacerbated by utilizing development methods that are hostile to change. For example, traditional Requirements Engineering (RE) attempts to minimize change by baselining requirements [4]. However, needs and requirements constantly evolve, given the extended development periods of complex projects. Consequently, the impact of requirements volatility has far-reaching effects like increasing the defect density during the coding and testing phase [5] and affecting the overall project performance [6]. Thus, traditional RE is ill-equipped to scale the monumental complexity of large-scale systems and accommodate the dynamics of extended development periods. Hence, there is a need for an alternative approach that transcends the complexity and scalability limits of current RE methods.
1.1 Motivation

Software development is a dynamic process of developing a system in the implementation domain in order to solve a problem in some application domain. Brooks [7] observes that determining system functionality is a formidable phase in this process. The problem of requirements volatility [8] further emphasizes the need for precise, complete and unambiguous specifications. In the past, the approach of baselining requirements has helped combat volatility in small-scale systems. The relative simplicity of system functionality and brief development cycles disallowed changing user perceptions. Furthermore, the inability to foster new technology in a short time-period assured the realization of systems using initial technology specifications. However, large-scale computer-based systems are inherently more complex and necessitate an alternate approach of development. Specifically, our research focuses on complex emergent systems, whose characteristics are described below.

- **Complex**: System is in a state of constant flux under the influence of dynamic factors. The intricate relationships between the constituent components heighten the system complexity.

- **Emergent**: The system is greater than the sum of its parts. The interactions between the components generate new system properties.

- **Extended Development Cycle**: The average development period is at least 5-10 years.

- **Incremental Development Process**: System is developed and deployed in increments.

- **Prolonged System Lifetime**: The final system is used for several decades.

Although, it is convenient to baseline requirements given the sheer magnitude of a complex emergent system, the extended development cycle makes this approach untenable. Users
mature with time and consequently, are able to express more clearly their expectations of the system. Furthermore, original requirements become obsolete with the accelerated pace of technology evolution. Hence, it is ideal to maximally delay requirement specifications and yet detail the system in a manner that is: comprehensible by the user, accommodates incremental evolution and amenable to change.

To alleviate the detrimental effect of dynamic agents on large-scale systems, it is imperative that they are designed to be change-tolerant; change-tolerance is the ability of the system to accommodate change with minimum impact. Apart from the impact of external change factors, we are also concerned with decreasing the number of potential change requests generated because of the development process, or constraints of system design. For example, factors such as loss of information when translating user needs to system requirements, or the incomplete specifications can results in change requests later in the development cycle. We seek to achieve change-reduction by developing a structured process of development. Thus, there is a need for an alternate development approach for large-scale complex systems based on the objectives of change-tolerance and reduction. We introduce the notion of Capabilities and the process of constructing them \textit{i.e.} Capabilities Engineering (CE) to fulfill these objectives.

### 1.2 Problem Statement

We propose a Capabilities-based approach for building complex emergent systems. The CE process is designed to promote change-reduction, and formulate Capabilities as change-tolerant functional abstractions that are foundational to the composition of system functionality. Specifically, they are embodied with the following characteristics:

- **Maximally cohesive:** Lehman \cite{9} observes that building systems using functionally independent units, helps manage system evolution. We construct Capabilities as functionally cohesive entities, whose elements work towards the same objective.
• **Minimally coupled**: Dependency links between Capabilities are minimized to decrease coupling, consequently, reducing the ripple effect generated by change.

• **Balanced Abstraction**: Requirements represent a reductionistic approach to system definition. On the other end user needs are often holistic. Capabilities subscribe to the concept of a balanced level as a reconciliation between the extremities of abstraction [10].

The CE process also employs a multi-disciplinary optimization (MDO) approach to select optimal sets of Capabilities that accommodate an incremental development approach, and reflect external constraints as discussed below:

• **Changing Needs**
  User needs inevitably change throughout the system development process, especially given the lengthy development period. Needs expressed at the beginning of a project have a high probability of varying after several years into development. Ignoring these changes only results in a system with undesirable functionality. Hence, the dynamic nature of needs has to be accommodated to build a successful system.

• **Schedule Constraints**
  Scheduling has been empirically identified as a key risk component in software development [11]. It is often discussed with respect to global project management aspects such as the distribution of personnel effort, allocation of time, determination of milestones and others [12]. We manipulate the sequence in which Capabilities are delivered to accommodate scheduling constraints.

• **Technology Advancement**
  We examine two possible scenarios when incorporating technology in a system—obsolescence and infusion. The former involves replacing obsolete technology with new technology and the latter introduces new technology in the system, as a result of building new Capabilities. We classify both these aspects under technology advancement.
1. **Technology Obsolescence**

The mission of a complex emergent system is usually critical and therefore, has to be developed using the state of art technology. Requirements are specified based on the current technology available at that point in time. However, given the rapid rate of hardware advancements, a lengthy development period can render the initial technology requirements invalid. Unfortunately, decisions made on the basis of obsolete technology are executed, as incorporating new technology stipulates changing the baselined requirements.

2. **Technology Infusion**

The functionality expected of a system may undergo substantial modification over a long period of time. Hence, it is highly probable that new Capabilities are expected of the system. For best results, these will be implemented using the latest technology. Thus, infusion of technology into the existing system must be executed with minimum impact of change.

Hence, we propose a **Capabilities based development framework** that accommodates changing needs and incorporates technology advancement with minimum volatility.

We claim that Capabilities abet the progression from needs to requirements and so, occupy an intermediary region between the problem space (needs) and solution space (requirements). Therefore, CE promotes a more disciplined, deliberate and scientific transition from user needs to system requirements, reducing and accommodating change.

### 1.2.1 Issues

We describe the issues of our problem in terms of the two major objectives that the Capabilities-based approach seeks to accomplish: *change-reduction* and *change-tolerance*. 
1.2.1.1 Change Reduction

The vastly different perspectives of an analyst and a developer result in a disconnect during system development. Where the former is concerned with higher order descriptions of functionality, the latter focuses on details. Although, the output from the analyst feeds as input for the developer, the disparity in their views generates documents that are often uncoordinated in nature. This results in the need for additional configuration, and traceability mechanisms to track changes in one or more entities. We propose that the CE process provides a visualization strategy that records user needs, and system requirements in a manner that bridges this disconnect. In addition, by structuring the informality of the problem space the CE process facilitates a reduction in change, that otherwise is a by-product of a less organized process. The following are the related issues:

**Structuring informality:** User needs lack the quality characteristics of a system requirement such as precision, unambiguity, completeness, verifiability, and others. The issue here is how to record informal needs in structured manner, and enable the generation of formal system requirements. In addition, we need to depict the level of abstraction of each need, and avoid representing repetitive information.

**Bridging the disconnect:** We know that user needs are used to generate requirements that are input to the solution space. One issue is the resolution of the complexity gap between the problem space of the user needs, and the solution space of the system requirements. The other problem is the ability to encompass disparate views of the development team within a common framework.

1.2.1.2 Change Tolerance

Capabilities, embodied with change-tolerant characteristics, define the composition of the system. We discuss below several issues related to the formulation of such Capabilities:

**Capability Definition:** We theorize that Capabilities result from adapting an approach
that is neither *holistic* (as in user needs) nor *reductionist* (as in system requirements). It is essential to define the process of identifying Capabilities from needs, and deriving associated requirements. The objective of an RE process is to establish the system specifications given an initial set of user needs. Introduction of Capabilities modifies this process and entails a systematic delineation of the transition from needs to Capabilities, and from Capabilities to requirements.

**Capability Position:** We need to establish the position of Capabilities relative to the problem and solution spaces. The problem and solution space; these spaces consist of elements related to the problem being solved *viz.* needs and solution being implemented *viz.* requirements, respectively. However, a Capability is neither a need nor a requirement. Hence, we must establish its spatial topography and definitive characteristics relative to the problem and solution space.

**Capability Identification:** We envision Capabilities as units that accommodate change with minimum impact. To achieve these objectives we identify units that are *maximally cohesive, minimally coupled*, and described at a *balanced level* as Capabilities. We choose to maximize functional cohesion as it is more desirable when compared to other types of cohesion [13] [14]. In addition, it is logical to estimate functional cohesion of a functional abstraction, *i.e.* a Capability. We minimize coupling to reduce the effects of change propagation. Balanced level supports the notion of avoiding the extremes of an abstraction level.

### 1.3 Solution Approach

In this section we provide a brief description of our potential solution approach to address the issues listed in Section 1.2.1.

1. *What is the visualization strategy to bridge the disparity among different views of system functionality*
We use a Function Decomposition (FD) graph to represent needs, and thereby, understand desired system functionalities. The graph is constructed using a top-down decomposition approach, where each node represents a need (associated functionality). The distance of the node from the root is used to represent the associated level of abstraction. The representation of all the expected functionality in a single graph provides a common framework of reference for the development team.

2. How are Capabilities identified

The CE process mathematically exploits the structural semantics of the FD graph to formulate Capabilities as functional abstractions with high cohesion, low coupling, and balanced abstraction. These Capabilities can be further optimized, using an MDO approach, to accommodate external constraints.

3. How are the change-tolerant characteristics of cohesion, coupling and balanced abstraction level of a Capability measured

We measure cohesion and coupling by constructing metrics that utilize structural information provided by the FD graph. The cohesion measure is based on a nodes’ sizes and their functional affinity. Coupling between nodes is a function of the distance between them, and the probability that a particular element will change. Trade-off analysis between the values of coupling and size help decide if the abstraction level of a node is balanced.

4. What is the position of Capabilities relative to the problem and solution space

Firstly, ascertain what entities constitute the problem space and solution space. Then, identify and analyze the characteristics of these entities to determine the existence of possible links between the spaces. Utilize these links to establish the relative position of Capabilities.

5. How to revise Capabilities such that they accommodate external constraints such as schedule and technology advancement
An MDO approach can be adapted to examine alternate configurations of Capabilities to accommodate external constraints. For example, development and delivery of Capabilities can be determined by the schedule deadlines. However, this sequence of development may be sub-optimal with respect to the change-tolerant characteristics. Thus, the MDO approach provides the facility to examine several different orderings.

6. *How does the CE process structure informality*

The CE process begins with user needs in the problem space, formulates Capabilities in the transition space, and generates requirements in the solution space. This deliberate progression between the spaces preserves domain information, and provides an inherent traceability mechanism to map needs to requirements. Pre-requirement specification traceability is a consequence of using the CE process.

7. *Prove the validity of this approach*

We present two empirical experiments to validate the change-tolerant, and change-reduction features of Capabilities and the CE process. Validation is performed on large-scale real-world systems: *Course Evaluation System* and *Sakai*. In the former, we validate change-tolerance properties of Capabilities by comparing the impact of introducing random need-changes on a CE and RE- based design. To validate the change-reduction property of the CE process, we use the *Sakai* system to analyze if there is a decrease in the number of change-requests when using the CE process, in comparison to the conventional RE process.

### 1.4 Document Organization

The thesis is organized as follows: In Chapter 2 we discuss role of CE in the context of process-centric and specification-centric approaches. In addition, we present different change-management strategies. Chapter 3 describes the CE process in detail. We present the FD graph, the change-tolerant measures, and the algorithm for Capability identification. In
Chapter 4 we describe the empirical experiments that prove the validity of change-tolerance and reduction properties of Capabilities and the CE process, respectively. Chapter 5 characterizes the spaces, and discusses how the CE process facilitates pre-requirement specification traceability. Finally we conclude in Chapter 6 with a summary of the main contributions, and suggestions for future work.
Chapter 2

Background

We classify systems that are large-scale, complex, have lengthy development cycles and have a lifetime of several decades as complex emergent systems. Simon [15] characterizes a system to be complex when it consists of a large number of parts that interact in a non-trivial manner. The enormous magnitude of a large-scale complex system impedes a priori knowledge about the effects of these interactions. As a result, the behavioral characteristics of the overall system is greater than a mere aggregation of its constituent elements. This behavior includes properties that emerge from the elemental interactions and are characteristic only of the global system. Specifically, it is fallacious to attribute these emergent properties to individual elements of the system [10].

There is a growing recognition for the need to rethink traditional development methods to accommodate the emergent nature of such software-intensive systems [10]. Hitherto, these methods have been suitable for conventional systems, which when compared to complex emergent systems, are of a smaller-scale, less complex, have brief development cycles and have a shorter lifetime. Consequently, requirements can be baselined after a certain point in the development period. However, requirements and technology often evolve during the extended development periods of complex emergent systems, and thereby, inhibit a comprehensive up-front solution specification. Thus, a primary difference between developing conventional
software systems and complex emergent systems is the lack of a final solution specification in the latter case caused by continuous system evolution. Therefore, it is imperative that a complex emergent system constantly adapts to various influences of change in order to function satisfactorily [17].

We broadly view change-management techniques as process-oriented or specification-oriented. The former includes improving specific phases or activities (elicitation methods, design techniques, verification procedures, and so on) within the software development process. In contrast, the latter, specification-centric methods, concentrate on developing suitable mechanisms for detailing change-tolerant systems. The objective in either case is to minimize change, and/or accommodate change. In this section, we examine certain representative techniques in both of these categories and analyze the nature of CE to determine the approach it follows.

### 2.1 Process-Centric Approaches

A strategy to minimize change is to control the source of the change, a principle primarily demonstrated by traditional RE as it attempts to minimize change by fixing requirements prior to design and development. This mandates that needs, the originating source of requirements, be accurate and complete, which is unrealistic because users refine their needs as the system evolves. On the other hand, there are numerous models that recognize the inevitability of change and so, strive to address changing needs as long as feasible, through iterative processes and incremental development [18] [19] [20]. Some development paradigms, like Extreme Programming [21], adopt an unconventional approach of eliminating formal RE from their process. Agile Modeling proposes lightweight RE [22]. Because requirements volatility is a key dynamic, we examine some requirement maturity models that facilitate process improvement. In addition, we discuss how certain aspects of design are manipulated to minimize the effects of such volatility.
2.1.1 RE Process Improvement

The Software Engineering Institute (SEI) defines various Capability Maturity Models (CMM) that provide a process improvement framework, for software organizations to assess and improve their processes. However, their lack of detailed guidelines regarding the requirements process has spawned a number of maturity models specific to the RE phase. Sommerville et al’s work in RE process improvement [23] [24] [25] [26] resulted in the definition of an RE maturity model, which complements the SEI’s CMM. The RE maturity model has three maturity levels - initial, defined and repeated, which are determined by an algorithm that assigns scores based on 66 good practice guidelines. An empirical study of an industrial RE process suggested that the maturity model contributed in improving the requirements process [27]. A similar model called Requirements Capability Maturity Model (R-CMM), based on SEI’s framework, was developed by Beecham et al [28]. However, the R-CMM is defined for all the five maturity levels of the CMM and uses a Goal Question Metric (GQM) paradigm. This includes a ‘process’ element to continuously assess and improve the requirements process achieve a specific maturity level. Doerr et al [29] argue that requirement maturity models are inadequate in the case of a single organization handling different inter-related projects and instead, propose requirements process improvement based on an information model.

2.1.2 Design

Modularity, reduced coupling, high cohesion and other design principles are recognized as characteristics of a “good” design as they assist in managing the complexity of system development [30] [31]. More recently, the boundaries between requirements analysis and design have become fuzzy, especially in the object-oriented (OO) paradigm. This close association can be exploited to:

1. restructure the design to accommodate requirement changes, and also,

2. incorporate desirable design properties in requirements
**Restructuring design to accommodate requirement changes:** An example of this approach is the use of the intent specifications and semantic coupling to reduce the effects of requirements changes through design \[32\]. Also, Wen and Dromey \[33\] automate design modifications caused by functional requirement changes using Genetic Software Engineering (GSE). Unlike conventional software engineering, where the design is constructed to satisfy requirements, in GSE the design is evolved from requirements formally represented as behavior trees \[34\]. Therefore, changes in functional requirements automatically generate changes in corresponding design entities. Hence, aspects of design can be manipulated to reconcile with requirements change.

**Incorporating design properties in requirements:** This approach of structuring requirements with design properties is uncommon. The role of requirements is constrained to being a specification driver for system design and development. This necessitates that they display certain quality characteristics like clarity, precision, unambiguity, testability and others. As a consequence, there is a lack of emphasis on incorporating design characteristics in RE. This is unfortunate because the rate of requirements volatility has been reported to be high during the specification phase \[8\]. As mentioned earlier, modularity, reduced coupling, high cohesion and other design principles assist in managing change. Thus, structuring requirements with design attributes can help contain some amount of volatility. Although, not completely process-centric CE utilizes this approach to combat complexity.

When building large systems, empirical research evidence indicates the failure of traditional RE to cope with the attendant requirements evolution \[35\] \[36\]. Consequently, in the case of complex emergent systems, which are often mission-critical, such failures are extremely expensive in terms of cost, time and human life. Thus, there is a need for a fundamental shift in the conventional approach of minimizing change to accommodating change.
2.2 Specification-Centric Approach

In the post World War II era, the Department of Defense (DoD) provided third party contractors with detailed technical specifications, military standards, manufacturing drawings and processes, inspection procedure details, and so on regarding the system to be developed, to ensure high product quality. However, this constrained the contractor and inhibited any possible product improvement. Furthermore, developers focused on fulfilling the system’s engineering specifications rather than understanding the users’ real needs. Consequently, the resulting unsatisfactory products forced DoD to specify systems using different methods, viz. performance based specifications and later, Capability Based Acquisition.

2.2.1 Performance Based Specifications

Performance based specifications [37] were introduced with the objective of accommodating instead of minimizing change. They are statements of requirements described as outcomes desired of a system. A criteria of compliance is associated with each specification [38]. These specifications indicate what is required of the system from a high level perspective. As a result, the solution is constrained to a much lesser degree and provides greater latitude in incorporating suitable design techniques and technology.

2.2.2 Capabilities Based Acquisition

More recently, Capability Based Acquisition (CBA) [39] [40] is being used to resolve problems posed by lengthy development periods and increased system complexity. It is expected to accommodate change and produce systems with relevant capability and current technology by delaying requirement specifications in the software development cycle, and maturing a promising technology before it becomes a part of the program.

CBA uses an evolutionary approach to deliver Capability in increments. A Capability is
described as a functionality expected from a system. In the initial phases of development, capabilities desired from the system are identified, but the requirements are unspecified until much later in the development cycle. Furthermore, intuition rather than scientific guidelines is used to determine the level of abstraction at which a capability is specified. However, CBA offers certain key concepts which are useful to incorporate in the development of complex emergent systems:

- Delaying the specification of performance and functional requirements in the software development cycle, and

- Maturing a promising technology before it becomes a part of the program.

CBA and performance based specifications, both strive to avoid constraining the system in terms of its design or technology. The former awaits the maturation of a technology in order to implement a Capability. This ensures that, however futuristic, the most suitable technology is incorporated in the system. However, in the latter case, the performance criteria of the system is specified based on the currently available technologies. Consequently, developers utilize any technology or design that fulfills the performance objectives, resulting in systems with obsolete technology.

However, neither Performance based specification nor the CBA approach defines the level of abstraction at which a specification or a capability is to be described. Furthermore, they neglect to outline any scientific procedure for deriving these types of specifications from the initial set of user needs. Therefore, these approaches propose solutions that are neither definitive, comprehensive nor mature enough to accommodate change and benefit the development process of complex emergent systems. In addition, their attempts to accommodate change is restrained to the specification process only. Nevertheless, they do provide certain key concepts — reduced emphasis on detailed requirements specification, and nurturing a promising technology before it becomes a part of the program — that are incorporated in CE as a part of its strategy to accommodate change.
2.3 Capabilities Engineering

The CE process is similar to Performance based specifications and CBA in that it utilizes a specification-centric approach to accommodate change. However, it transcends the convention of mere specification to introduce design aspects into what was hitherto a purely requirements dominated realm. Specifically, it formulates Capabilities to embody the software engineering characteristics of high cohesion, minimal coupling and balanced abstraction levels. By incorporating such desirable attributes in a phase that even precedes the formalization of requirements, the CE process influences the structure of the system to be developed. This influence is guided by theoretical principles and specific measures constructed to ensure that the basic building blocks (Capabilities) of the system exhibit the ability to accommodate change with minimum impact. Embedding desirable design traits in a specification, introduces aspects of a process-centric approach. Hence, CE is a hybrid approach of both the process and specification-centric approaches to managing volatility.
Chapter 3

Capabilities Engineering Process

Capabilities are generated in a two-phased process termed the CE process. The first phase determines the change-tolerant Capability set that exhibits high cohesion, low coupling and balanced abstraction levels. The second phase optimizes these Capabilities to accommodate the constraints of technology feasibility and implementation schedule. Figure 3.1 illustrates the two major phases of the CE process.

Phase I implicitly derives expected system functionality from needs and decomposes them to directives; directives are similar to requirements but have domain information associated

Figure 3.1: Capabilities Engineering Process
with them. The decomposition activity results in the construction of the FD graph. Then, the algorithm for identifying Capabilities — based on the criteria of cohesion, coupling and abstraction level — is executed on this graph, as a part of the formulation activity. The resulting set of Capabilities are the required change-tolerant entities.

Phase II, employs an multi-disciplinary optimization approach on the Capabilities obtained from Phase I to accommodate the constraints of technology and schedule. The resulting set of Capabilities is then transformed into requirements as dictated by an incremental development process. The final set of Capabilities and their associated requirements constitute the output of the CE process.

We briefly describe these activities below:

**Decomposition:** This is the process of recursively partitioning a problem until an atomic level is reached. We begin with user needs because they help determine what problem is to be solved; in the context of software engineering this means what functionality is expected of the system to be developed. Different techniques such as interviews, questionnaires, focus groups, introspection and others [41] are employed to gather information from users. Often, because of the informality of the problem domain language, needs are expressed at varying levels of abstraction. A function derived from a need at the highest level of abstraction is the mission of the system. An abstraction presents information essential to a particular purpose, ignoring irrelevant details. In particular, a functional abstraction indicates the functionality expected of the system from a high-level perspective while ignoring minute details. We use the vertices (or nodes) of an FD graph to represent functional abstractions of the system, and its edges to depict the relationship between the various functionalities. The construction of this graph is a core component of the decomposition activity and is explained in detail in Section 3.1.

**Formulation:** The formulation activity uses the Capability Identification Algorithm to determine Capabilities from the set of all possible functionalities represented by the FD graph. Each step of the algorithm is guided by the compelling necessity for complex emergent sys-
tems to accommodate change with minimum impact. The algorithm utilizes three principle criteria — cohesion, coupling, abstraction level — to evaluate the functional abstractions and determine if they are Capabilities. These criteria are a result of analyzing the FD graph from a top-down and bottom-up perspective to identifying Capabilities [42]. To understand how the algorithm identifies change-tolerant functional abstractions, we first need to discuss the rationale behind these criteria and their corresponding mathematical measures. Section 3.2 describes why we choose high cohesion, low coupling and balanced abstraction levels as the defining criteria for a Capability.

**Optimization:** The inputs for the optimization activity are sets of highly cohesive, minimally coupled Capabilities. This activity aims to identify that set which best accommodates the constraints of schedule and technology.

**Transformation:** Requirements are still the basis for developing systems. They fulfill several purposes that include providing the rationale for design, criteria for validation and others as enumerated in [43]. Thus, there is a need to transform directives to requirements. In systems that are incrementally developed, only the directives associated with the Capability chosen for development are to be transformed to requirements. This is in agreement with the principle of delaying the specification of requirements in Capability Based Acquisition [40], and thereby, avoiding the pitfalls of fixed requirements. We hypothesize that there is a one-many mapping between directives and requirements.

Thus, Phase I generates change-tolerant Capabilities based on the principle characteristics of cohesion, coupling and abstraction level. Phase II further optimizes these Capabilities to suit external constraints. In this chapter, we discuss in detail the aspects of Phase I: Section 3.1 describes the construction of the FD graph, a core aspect for formalizing the problem space. Section 3.2 discusses the rationale of cohesion, coupling and abstraction level for defining Capabilities. In addition, we present measures specifically constructed for quantifying these attributes of a Capability. Section 3.3 enumerates the algorithm for Capability identification. Each step of the algorithm is explained in the context of the
experiment with the Course Evaluation System. This experiment is part of our empirical validation of the change-tolerant properties of Capabilities, detailed in Section 4.1.

3.1 Function Decomposition Graph

The vertices and edges of an FD graph represent expected system functionalities and the relation between them, respectively. The FD graph for the Course Evaluation System is presented in Figure 3.2, where the nodes represent different functional abstractions and the edges indicate either a decomposition, intersection, or a refinement relationship. In addition, the variance in the abstraction levels is depicted by the depth of a node in the FD graph, where depth is the distance of a node from the root. Thus, the greater the distance, the lower the abstraction level.

Formally, we define an FD graph $G = (V, E)$ as an acyclic directed graph where:

1. $V$ is the vertex set that represents system functionality at various levels of abstraction in accordance to the following rules:

   (a) **Mission:** The root represents the highest level mission of the system. There is exactly one overall system mission and hence, only one root node in an FD graph. In Figure 3.2, $n_1$ is the root node as $\text{indegree}(n_1) = 0$ (indegree is the number of edges coming into a vertex in a directed graph), denoting that the mission is to develop a Course Evaluation System.

   (b) **Functional Abstraction:** An internal node represents a functionality expected of the system. We say $n$ is an internal node if $\text{indegree}(n) \neq 0$ and $\text{outdegree}(n) \neq 0$ (outdegree is the number of edges going out of a vertex in a directed graph). For example, in Figure 3.2 internal node $n_2$, Evaluation Authoring, is a functional abstraction. The level of abstraction of a node decreases with an increase in its distance from the root; distance is the number of edges in the shortest di-
rected path. Thus, Scale \((n_8)\), is of a lower abstraction level when compared to Evaluation Authoring \((n_2)\).

(c) **Directive:** We define a directive as a low-level characteristic of the system formulated in the language of the problem domain. The leaf node of the FD graph represents a directive of the system. A system has a finite number of directives and hence, its FD graph also has the same number of leaves. For example, in Figure 3.2, nodes \(d_i, i = 16, \ldots, 19\) are directives as \(\text{outdegree}(d_i) = 0\). A directive has three main purposes:

i. **Captures domain information:** A directive can be incomplete, unverifiable, and untestable. However, it serves the purpose of describing system functionality in the language of the problem domain, which aids in capturing domain information. In contrast, a requirement is a statement formulated in the technical language of the solution domain and often neglects to preserve and convey valuable domain information. For example, in Figure 3.2, directive \(d_{49}\), \textit{Generate Report Across Semesters}, is vague and ambiguous, however, using terminology of the problem domain it indicates that there is a need to collate information from different semesters. A requirement could read: \textit{A report shall be generated for each course that is evaluated}. If this requirement is the only statement used to capture the expected functionality of report generation, then it neglects to convey that the information in the required report may need to span several semesters. In fact, Zave and Jackson [44] have identified the loss of appropriate domain knowledge in the process of requirements refinement as a key problem area in RE. Therefore, the introduction of directives provides momentum in bridging the gap between needs and requirements.

ii. **Facilitates formulation of Capabilities:** To identify Capabilities we need to examine all possible functional abstractions of a system represented in the FD graph. Although, directives are low-level characteristics in the problem
domain, they are implicitly associated with some larger functionality desired of the actual system. Hence, each functional abstraction is linked with a group of directives. Therefore, directives can be used to determine the cohesion and coupling values of various functional abstractions, and thereby play a pivotal role in the formulation of Capabilities.

iii. **Transformation to requirements:** A directive is affiliated with the problem space and a requirement with the solution space; yet both share the same objective of describing what is expected of the desired system. In addition, they are described at a similar level of abstraction. Hence, we conjecture that the transformation of a directive to one or more requirements is straightforward. Recall Capabilities are associated with a set of directives. Thus, this transformation process produces requirements associated with specific Capabilities and form the output of the CE process. The relationship between directives and requirements serves to facilitate pre-requirement specification traceability [15].

2. \( E = \{(u, v) | u, v \in V, u \neq v\} \) is the edge set, where each edge indicates a decomposition, intersection or refinement relationship between nodes. The edge construction rules are described below:

(a) **Decomposition:** The partitioning of a functionality into its constituent components is depicted by the construction of a decomposition edge. The direct edge between a parent and its child node represents functional decomposition and implies that the functionality of the child is a proper subset of the parent’s functionality. For example in Figure 3.2 the edges \((n_1, n_6), (n_1, n_2), (n_1, n_{57})\) indicate that the functionality of \(n_1\) is decomposed into smaller functionalities \(n_6, n_{57},\) and \(n_2\) such that \(n_1 \equiv n_6 \cup_{fn} n_{57} \cup_{fn} n_2\) where \(\cup\) is the union operation performed on functionality. Hence, only non-leaf nodes *i.e.* internal nodes with an outdegree of at least two can have valid decomposition edges with their children.
(b) **Refinement:** The refinement relationship is used when there is a need to express a node’s functionality with more clarity, say, by furnishing additional details. Let \((u, v) \in E\) and \(\text{outdegree}(u) = 1\), then the edge \((u, v)\) represents a refinement relationship. \(v\) is a refined version of its parent \(u\).

(c) **Intersection:** To indicate the commonalities between functions the intersection edge is used. Hence, a child node with an indegree greater than one represents a functionality common to all its parent nodes. For example, in Figure 3.2, \(n_{55}\) is a child node of parent nodes \(n_7\) and \(n_{29}\). Consequently, \(n_{55} \equiv n_7 \cap_{\text{func}} n_{29}\) where \(\cap_{\text{func}}\) is the intersection operation performed on functionality. The edges \((n_7, n_{55}), (n_{29}, n_{55})\) represent the intersection relationship.

The FD graph for the **Course Evaluation System** has been constructed using existing information from needs and requirements, documented by the project team. In addition, it has been validated by the members involved in the development of the system. In some sense, because of the construction rules the structure of the graph is reflective of the dependencies between the functionalities. The system, however, is constantly evolving and so, the graph in Figure 3.2 illustrating the system’s expected functionality, is a snapshot at a particular point in time.

### 3.2 Rationale for Capability Definition: *The Three Criteria*

Why are the characteristics of high cohesion, low coupling and balanced abstraction levels envisioned to define Capabilities as change-tolerant entities? In this section, we attempt to explain and justify the rationale underlying this definition of a Capability, and subsequently, present measures that are specifically constructed to compute each criterion.

*Why Cohesion & Coupling:* We desire that Capabilities be independent enough to serve
Figure 3.2: FD Graph of Course Evaluation System
individual functional interests, yet are allied in their overall objective to accomplish the system mission — a perspective that concurs with the principles of constructing complex systems [46]. Intuitively, units with high cohesion and low coupling satisfy this objective. Furthermore, techniques of modularization suggest high cohesion and low coupling are typical characteristics of stable units [30] [31]. Stability implies resistance to change; in the context of CE, stability refers to the property of change-tolerance.

Cohesion is a characteristic of a stable structure and depicts the “togetherness” of elements within a unit. Every element of a highly cohesive unit is directed toward achieving a single objective. We focus on maximizing functional cohesion, which indicates the strongest level of cohesion [14] among all the other types (coincidental, logical, temporal, procedural, communicational, and sequential) [31] [13] and therefore, is most desirable. In particular, a Capability has high functional cohesion if all its constituent elements, viz. directives (later transformed to requirements), are devoted to realizing the principle function represented by the Capability. In some sense, the elements of a Capability are connascent [47].

Why should Capabilities exhibit low coupling? We restate the reasons advanced by Page-Jones [48] for minimizing coupling, in the context of Capabilities.

Fewer interconnections between Capabilities reduces:

- the chance that changes in one Capability affects other Capabilities, thus promoting reusability,
- the chance that a fault in one Capability will cause a failure in other Capabilities, and
- the labor of understanding the details of other Capabilities

Thus, coupling is a measure of interdependence between units [49] and thereby, is the other indicator of stability of a Capability. We desire that units accommodates change with minimum ripple effect. Ripple effect is the phenomenon of propagation of change from the affected source to its dependent constituents [50]. Specifically, the dependency links between units
behave as change propagation paths. The higher the number of links, the greater is the likelihood of ripple effect. Therefore, we strive to design minimally coupled Capabilities.

**Why Abstraction Level:** The third criteria requires that Capabilities be defined at balanced abstraction levels. Given that holism and reductionism lie on the extremes of the abstraction scale, we seek a balance which is most desirable from a software engineering perspective. Specifically, we identify a balanced abstraction level as that point where the node is of an optimum size (size is the number of associated directives), and at the same time, whose implementation as an independent entity does not result in increased dependencies. For example, in Figure 3.2, \( n_2 \) representing the functionality Evaluation Authoring is of size 30 (number of associated directives); its children, Customized Evaluation (\( n_{29} \)) and Expert Template (\( n_7 \)), are smaller-sized nodes. Based on size, let us say we consider the children instead of the parent as Capabilities. This implies that nodes \( n_{29} \) and \( n_7 \) are independent entities. However, we see from the Figure 3.2 that the functionality Items (\( n_{55} \)) is common to both these Capabilities. This, in some sense, is a manifestation of content coupling, the least desirable among all types of coupling [31]. Consequently, the dependency between \( n_{29} \) and \( n_7 \) is increased by deploying them as separate Capabilities, because they share a common functionality in \( n_{55} \). This trade off between the convenience of developing smaller sized units and the long-term advantages of reduced dependencies, characterizes a balanced abstraction level. Thus, based on certain heuristics we use the level of abstraction to determine which nodes in an FD graph are Capabilities.

**Why Measures:** A measure is a number used to characterize some feature or property of an entity [51]. We are interested in measuring the cohesion, coupling and abstraction level of various functional abstractions of a system to identify Capabilities. It is generally observed that as the cohesion of a unit increases the coupling between the units decreases. However, this correlation is not exact [31]. Therefore, we develop specific metrics to measure the coupling and cohesion values of the internal nodes in an FD graph. Most existing coupling and cohesion measures focus on evaluating the quality of design or code [52], [53]. These measures have access to information regarding function calls, data parameters and other design
or implementation details, which are abundantly available to construct their metric computations. In contrast, measures for Capabilities are based on the fundamental definitions of cohesion and coupling, and rely on the limited information provided by the FD graph. To determine balanced abstraction levels, we compute the sizes of nodes and examine the levels in terms of their distances from the root. In addition, the FD graph helps us visually understand the commonalities between potential Capabilities. We use this information to construct heuristics to evaluate the abstraction levels.

Thus, specific metrics to compute the cohesion, coupling and abstraction levels of the nodes in the FD graph are developed in order to formulate Capabilities. Sections 3.2.1 and 3.2.2 describe in detail the cohesion and coupling measures. Section 3.2.3 presents the computations for size and depth, and discusses the heuristics for determining balanced abstraction levels.

### 3.2.1 Cohesion Measure

A unit has functional cohesion if it focuses on executing exactly one basic function. Yourdon and Constantine [31] state that every element in a module exhibiting functional cohesion “is an integral part of, and is essential to, the performance of a single function”. By virtue of its construction, in the FD graph the function of each child node is essential in achieving the function of its immediate parent node. For example, in Figure 3.2 implementing the functionality of Email \(n_{30}\) is necessary, to accomplish Evaluation Distribution \(n_{59}\).

Note that, neither the root nor the leaves of an FD graph can be considered as a Capability. This is because the root indicates the mission of the system, which is too holistic, and the leaves symbolize directives, which are too reductionistic in nature. Both of these entities lie on either end of the abstraction scale, and therefore, choosing them conflicts with the objective of avoiding such polarity when developing complex emergent systems [10]. Thus, only the internal nodes of an FD graph are considered as potential Capabilities. These internal nodes depict functionalities at different levels of abstraction, and thereby, provide a repre-
sentative sample from which to identify Capabilities. We develop the cohesion measure for internal nodes by first considering nodes whose children are only leaves. We then generalize this measure for any internal node in an FD graph.

3.2.1.1 Measure for internal nodes with only leaves as children

Internal nodes with only leaves as children represent potential Capabilities that are directly linked to a set of directives. In Figure 3.2, examples of such nodes are \( n_3, n_4, n_5, n_6, n_8, n_9, n_{31}, n_{32}, n_{41}, n_{50} \). Directives are necessary to convey and develop an in-depth understanding of the expected functionality and yet, by themselves, lack sufficient detail to dictate system development. Failure to implement a directive can affect the functionality of the associated Capability with varying degrees of impact. We hypothesize that the degree of impact is directly proportional to the relevance of the directive to the functionality. Consequently, the greater the impact, the more essential the directive. This signifies the strength of relevance of a directive and is symptomatic of the associated Capability’s cohesion. Hence, the relevance of a directive to the functionality of a unit is an indicator of the unit’s cohesion.

The failure to implement a directive can be interpreted as a risk. Therefore, we use existing risk impact categories: Catastrophic, Critical, Marginal and Negligible [54] to guide the assignment of relevance values. Each impact category is well-defined and has an associated description. This is used to estimate the relevance of a directive on the basis of its potential impact. For example, in Table 3.1 negligible impact is described to be only an inconvenience, whereas a catastrophic impact implies complete failure. This signifies that the relevance of a directive with negligible impact is much lower when compared to a directive with catastrophic impact.

Intuitively, the impact categories are ordinal in nature. However, we conjecture that the associated relevance values are more than merely ordinal. The issue of determining the natural measurement scales [55] of cohesion and other software metrics is an open problem [56]. Therefore, we refrain from subscribing both, the attribute in question i.e. cohesion
and its metric \( i.e. \) function of relevance values, to a particular measurement scale. Rather than limiting ourselves to permitted analysis methods as defined by Stevens \[55\] we let the objective of our measurement — computing the cohesion of a node to reflect the relevance of its directives — determine the appropriate statistic to be used \[57\].

We assign values to indicate the relevance of a directive based on the perceived significance of each impact category; these values are normalized to the \([0,1]\) scale. The categories and their associated relevance values are listed in Table 3.1. We estimate the \textit{cohesion} of an internal node as the average of the relevance values of all its directives. The \textit{arithmetic mean} is used to compute this average as it can be influenced by extreme values. This thereby captures the importance of directives with catastrophic impact or the triviality of directives with negligible impact, and affects the resulting average appropriately, to reflect the same.

For an FD graph \( G = (V, E) \) we denote relevance of a directive \( d \) to its parent node \( n \) as \( \text{Rel}(d, n) \) where \( d, v \in V, (n, d) \in E, \text{outdegree}(d) = 0 \) and \( \text{outdegree}(n) > 0 \). \( \text{Rel}(d, n) \) indicates the contribution of directive \( d \) to the cohesion of parent node \( n \). For example in Figure 3.2, \( \text{Rel}(d_{12}, n_9) = 0.7 \) \( i.e. \) if the system fails to provide a facility to Create new items \( (d_{12}) \), the success of \textbf{Core Item Operations} \( (n_9) \) is questionable. Note that, we measure the relevance of a directive only to the immediate functionality represented by its parent node.

Formally, the cohesion measure of a Capability that is directly associated with a set of directives \( i.e. \) the cohesion measure of an internal node \( n \in V \) with \( t \) leaves as its children \((t > 0)\), is given by computing the arithmetic mean of relevance values:

### Table 3.1: Relevance Values

<table>
<thead>
<tr>
<th>IMPACT</th>
<th>DESCRIPTION</th>
<th>RELEVANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catastrophic</td>
<td>Task failure</td>
<td>1.00</td>
</tr>
<tr>
<td>Critical</td>
<td>Task success questionable</td>
<td>0.70</td>
</tr>
<tr>
<td>Marginal</td>
<td>Reduction in performance</td>
<td>0.30</td>
</tr>
<tr>
<td>Negligible</td>
<td>Non-operational impact</td>
<td>0.10</td>
</tr>
</tbody>
</table>
31

\[ Ch(n) = \frac{\sum_{i=1}^{t} Rel(d_i, n)}{t} \]

For example in Figure 3.2, \( Ch(n_9) = 0.85 \). The cohesion value ranges between 0 and 1. A Capability with a maximum cohesion of 1 indicates that every constituent directive is of the highest relevance.

### 3.2.1.2 Measure for internal nodes with only non-leaf children

The cohesion of internal nodes with only non-leaf children is computed differently. This is because the relevance value of a directive is valid only for its immediate parent and not for its ancestors. For example, in Figure 3.2 the functionality of Reports \((n_{63})\) is decomposed into Core Report Operations \((n_{50})\) and Report Generation \((n_5)\). This implies that the functionality of \(n_{63}\) is directly dependent on the attainment of the functionality of both \(n_{50}\) and \(n_5\). Note that \(n_{63}\) has only an indirect relationship to the directives of the system. In addition, the degree of influence that \(n_{50}\) and \(n_5\) each have on parent \(n_{63}\) is influenced by their size (number of constituent directives). Therefore, the cohesion of nodes that are parents with non-leaf children is a weighted average of the cohesion of their children. Here, the weight is the size of a child node in terms of its constituent directives. This indicates the child’s contribution towards the parent’s overall cohesion. The rationale behind this is explained by the definition of cohesion, which states that a node is highly cohesive if every constituent element is focused on the same objective, i.e. the node’s functionality.

Formally, the cohesion measure of an internal node \(n\) with \(t > 1\) non-leaf children is:

\[ Ch(n) = \frac{\sum_{i=1}^{t} (size(v_i) \cdot Ch(v_i))}{\sum_{i=1}^{t} size(v_i)} \]

such that \((n, v_i) \in E\) and,
size(n) = \begin{cases} 
\sum_{i=1}^{t} size(v_i) & (n, v_i) \in E; \text{outdegree}(v_i) > 0; \\
1 & \text{outdegree}(n) = 0 
\end{cases}

In the case where outdegree(n) = 1, i.e. the node has only one child v (say), then the Ch(n) is the Ch(v); if outdegree(n) = 0, i.e. n is a leaf (directive), Ch(n) is not applicable.

### 3.2.2 Coupling Measure

As with cohesion, the concept of coupling was introduced by Stevens et al. [49] as the “measure of the strength of association established by a connection from one module to another”. Coupling is also characterized as the degree of interdependence between modules. Our objective is to identify minimally coupled internal nodes as Capabilities.

A Capability is related to another Capability through its constituent directives. These can be common directives (for example, in Figure 3.2 Customized Evaluation \(n_{29}\) and Expert Template \(n_7\) share directives \(d_{10}, \ldots, d_{19}\)) or directives that have an increased affinity because their parent Capabilities share the same parent or immediate common ancestors. For example, Email Template \(n_{32}\) and Email Notification \(n_{31}\) share Email \(n_{30}\) as their parent; thus, their directives have a greater affinity than say, with the directives of Report Generation \(n_5\). Thus, the coupling between Capabilities is the measure of the dependencies between their respective directives. In the following sections, we first discuss the coupling between directives, and then develop the coupling measure for Capabilities.

#### 3.2.2.1 Coupling between directives

To measure the coupling between directives we utilize the structural information provided by the FD graph because we have neither design information nor implementation details at our disposal. We consider an undirected version \(G'\) of the FD graph \(G\), as presented
in Figure 3.3 where $G' = (V, E')$ and $E'$ is the set of undirected edges. We denote the coupling between directives $d_x$ and $d_y$ as $Cp(d_x, d_y)$. This indicates the dependency of $d_x$ on $d_y$, which can be quantified by measuring the effect on $d_x$ when $d_y$ changes. Similarly, $Cp(d_y, d_x)$ indicates the dependency of $d_y$ on $d_x$. Thus, $Cp(d_x, d_y) \neq Cp(d_y, d_x)$ implying that the coupling measure for directives is asymmetric.

We hypothesize that the coupling between directives as a function of two components: distance and probability of change.

1. **Distance:** Directives are associated with their parent Capabilities through decomposition edges; recall, a decomposition edge signifies that the functionality of the parent is a union of its children. Thus, the directives of a Capability are highly functionally related. In $G'$, this is represented by leaves that share the same parent node. However, relatedness between directives decreases as the distance between them increases. We define the distance between directives $u, v \in V$ as the number of edges in the shortest undirected path between them and denote it as $dist(u, v)$. By choosing the shortest path we account for the worst case scenario of change propagation. Specifically, the shorter the distance, the greater the likelihood of impact due to change propagation.

In Figure 3.3, **Delete Email Template** ($d_{34}$) and **View Email Template** ($d_{35}$) are directives of the same parent **Email Template** ($n_{32}$) and so are highly related with $dist(d_{34}, d_{35}) = 2$. In contrast, **Delete Email Template** ($d_{34}$) and **Generate Report Across Semesters** ($d_{49}$) have a lower relatedness with $dist(d_{34}, d_{49}) = 10$ as they share a common ancestor only in the system mission ($n_1$) (the paths between the directives are highlighted in Figure 3.3). Thus, from the distance measure we conclude that $d_{34}$ is less likely to be affected by a change in $d_{49}$ than a change in $d_{35}$; in effect, a change in the facility to **View Email Template** also affects the ability to **Delete Email Template**, but has little or no impact on **Report Generation Across Semesters**. Consequently, the dependence between $d_{34}$ and $d_{35}$ is much higher than between $d_{35}$ and $d_{49}$; thus implying, $Cp(d_{34}, d_{35}) > Cp(d_{34}, d_{49})$. 
Therefore, for any two directives \( u \) and \( v \in V \) we deduce that the coupling between directives decreases with the increase in the shortest distance between them. We denote this relationship as:

\[
Cp(u, v) \propto \frac{1}{\text{dist}(u, v)}
\]

2. **Probability of Change:** The other factor that influences the coupling measure is the probability that a directive will change and thereby, cause a ripple effect. Minimal interconnections reduce the likelihood of a ripple effect phenomenon. We know that coupling between Capabilities is a function of coupling between their respective directives. As mentioned earlier, if \( u \) and \( v \) are directives then \( Cp(u, v) \) can be quantified by measuring the effect on \( u \) when \( v \) changes. However, we still need to compute the probability that such a ripple effect will occur. This requires us to compute the likelihood that a directive might change. Therefore, \( Cp(u, v) \) also needs to factor in the probability of directive \( v \) changing: \( P(v) \).

We use a simplistic model to determine the probability that a directive will change. Specifically, we consider the likelihood that exactly one directive changes among all other directives in a given Capability. Therefore, for a Capability \( n \) of \( \text{size}(n) \), the probability that an associated directive \( d \) changes is given by:

\[
P(d) = \frac{1}{\text{size}(n)}
\]

For example, in Figure 3.3, for node \( n_3 \) with size 6, the probability that \( d_{24} \) will change is 0.16667. Although, a Capability is architected to be an independent entity, a change in a directive can impact the related directives of other Capabilities. The magnitude of this impact is based on the structural distance between the directives, which is determined by the distance measure, discussed earlier. Thus, we define the coupling between directives as a function of distance and probability of change.

Formally, coupling between two directives \( u \) and \( v \in V \) is computed as:
Figure 3.3: Undirected FD Graph of Course Evaluation System
This metric computes the coupling between directives $u$ and $v$ as the probability that a change in $v$ propagates through the shortest path and affects $u$.

### 3.2.2.2 Coupling between Capabilities

Coupling between Capabilities is determined as a function of the coupling between their respective directives because a change in a Capability implies a change in one or more of its constituent directives. As with directives, the coupling measure for Capabilities is asymmetric, i.e. if $p$ and $q$ be Capabilities then $Cp(p, q) \neq Cp(q, p)$. However, it is possible that $p$ and $q$ share common directives. In such a case, we need to make a decision about the membership of these directives and ensure that they belong to exactly one Capability. This is reflective of the actual system, where any functionality is implemented only once and is not duplicated.

We strive to increase the cohesion of a Capability, and so associate directives based on their percentage of contribution to the overall cohesion of the parent entity. As an example, let $Ch(p) = 0.873$ and $Ch(q) = 0.957$ and let $d$ be the shared directive, such that it relevance values are: $Rel(d, p) = 0.3$ and $Rel(d, q) = 0.7$. Then the percentage contribution of $d$ to the overall cohesion of $p$ is 34.36% and for $q$ is 73.14%. Because of its higher contribution to the cohesion of $q$, we associate the directive $d$ with $q$ rather than $p$.

We denote the set of leaves (directives) associated with an internal node $n \in V$ as:

$$D_n = \{ x \mid \exists \text{path}(n, x); \text{outdegree}(x) = 0; n, x \in V \}$$

where $\text{path}(n, x)$ is a set of directed edges connecting $n$ and $x$. For example in Figure 3.3, the set of leaves associated with the internal node $n_8 \in V$ is $D_{n_8} = \{ d_{16}, d_{17}, d_{18}, d_{19} \}$. Now consider $Cp(n_8, n_9)$, which denotes the coupling between Item Core Operations ($n_9$) and Scale ($n_8$). Specifically, $Cp(n_8, n_9)$ quantifies the effect on $n_8$ when $n_9$ changes i.e. we need
to compute the effect on the directives associated with \( n_8 \): \( D_{n_8} = \{d_{16}, \ldots, d_{19}\} \), when the directives associated with \( n_9 \): \( D_{n_9} = \{d_{10}, \ldots, d_{15}\} \), change. The coupling between \( n_8 \) and \( n_9 \) is given by:

\[
Cp(n_8, n_9) = \frac{\sum_{d_i \in D_{n_8}} \sum_{d_j \in D_{n_9}} Cp(d_i, d_j)}{|D_{n_8}| \cdot |D_{n_9}|}
\]

where \(|D_{n_8}|\) is the cardinality of \( D_{n_8} \).

Generalizing, the coupling measure between any two internal nodes \( p, q \in V \), where \( \text{outdegree}(p) > 1 \), \( \text{outdegree}(q) > 1 \) and \( D_p \cap D_q = \phi \) is:

\[
Cp(p, q) = \frac{\sum_{d_i \in D_p} \sum_{d_j \in D_q} Cp(d_i, d_j)}{|D_p| \cdot |D_q|}
\]

where \( Cp(d_i, d_j) = \frac{P(d_j)}{\text{dist}(d_i, d_j)} \) and \( P(d_j) = \frac{1}{|D_q|} \).

\( P(d_j) \) is the probability that directive \( d_j \) changes among all other directives associated with the node \( q \); \(|D_q| \equiv \text{Size}_q\), the number of directives associated with \( q \).

### 3.2.3 Abstraction Level

The technique of abstraction is instrumental in influencing software engineering phenomenon such as software architecture [58] [59], design patterns [60], object-oriented frameworks [61] and others. In fact, this stems from the recognition that high abstraction level is key in the development and evolution of complex emergent systems [62] [63] [64]. Similarly, we strive to identify nodes at balanced levels of abstraction as Capabilities. According to the FD graph, the node at the highest level is the overall mission (see Figure 3.2). If we implement this as a Capability, then the entire system is composed of exactly one large-sized Capability - a retrograde to the original RE approach. Instead, we need to identify Capabilities of a size
such that its functionality is comprehensible by the human mind \cite{65}. For this we consider nodes at lower levels of abstraction, that depict more specifically, what functionality is expected of the system.

The question that now arises is “How low of an abstraction level should one define a Capability ?”. From the FD graph (see Figure 3.2) we observe that as the abstraction level becomes lower, the node sizes decrease but the coupling values increase. We estimate the size of a Capability as the number of its associated directives, for example, in Figure 3.2 $size(n_1) = 43$, $size(n_9) = 6$. In fact, size estimates determined from non-code entities such as requirements are known to be fairly representative of the actual functional size \cite{66}.

Given a choice between two nodes of different sizes, we choose to implement the smaller-sized node as a Capability. This is in agreement with Miller’s observation about the limited processing capacity of the human mind \cite{65}. A large-sized Capability may encompasses too many functionalities for a developer to process. The worst case scenario of such a Capability is the root node and in Figure 3.2 this is Course Evaluation System.

Intuitively, the implementation of a smaller sized Capability is less complex. However, there is an ongoing debate in the software engineering community about the relation between the size of a component and number of defects or defect density. Several empirical studies have resulted in conflicting conclusions regarding this relationship. For example, Basili et al. \cite{67} argue that smaller sized components are more reliable than large components. In contrast, others \cite{68,69,70} observe that there is no relation between size and defects. Most of the reported research evidence is based on sizes determined by examining low-level design or code artifacts. In contrast, we are concerned with Capabilities, which exist in a realm even prior to requirements specification. This implies that the assumption of small-sized Capabilities being less complex and more easily maintainable than their large-sized counterparts is not invalid. Furthermore, as Fenton and Neil \cite{71} contend, in the light of insufficient information it is too premature to discredit the fundamental principles of modularization.
Two possible scenarios may arise when lowering the abstraction level of a node in order to decrease its size:

- **Common Functionality**: In the former case, lowering the abstraction level of the large-sized Capability results in nodes that share a common functionality. For example, in Figure 3.4, which is a snapshot of the original FD graph of the Course Evaluation System, this is illustrated by decreasing the level of Evaluation Authoring ($n_2$) to Expert Template ($n_7$) and Customized Evaluation ($n_{29}$); both share the common node Items ($n_{55}$).

- **No Common Functionality**: This case involves the reduction of a single aggregate to smaller sized nodes that have no commonalities. Figure 3.5 illustrates an example of this type of node viz. Results Analysis ($n_{57}$), a single aggregate that reduces to Responses ($n_4$) and Reports ($n_{63}$).
Thus for each scenario, \textit{balanced} abstraction levels are determined by examining the trade-space of two aspects: node size and coupling values. We now discuss these scenarios in detail using the two examples illustrated in Figures 3.4 and 3.5.

3.2.3.1 Common Functionality

Let Evaluation Authoring ($n_2$) be a Capability of the Course Evaluation System (see Figure 3.4). The size of this Capability is too large, and hence, we attempt to reduce its abstraction level to its children \textit{viz.} Customized Evaluation ($n_{29}$) and Expert Template ($n_7$), which are of a relatively smaller size. However, we observe that these nodes share a common functionality in Items ($n_{55}$). This implies that one of the links, ($n_{29}, n_{55}$) or ($n_7, n_{55}$), needs...
to be broken in order to implement Items as a part of a parent Capability. Let \((n_{29}, n_{55})\) be broken, and \textbf{Items} be implemented as a part of \textbf{Expert Template}. Consequently, Capabilities \(n_{29}\) and \(n_7\) are content coupled \cite{48} because \(n_{29}\) may attempt to manipulate the \textbf{Items} part ingrained in \(n_7\). Thus, lowering the abstraction level of \textbf{Evaluation Authoring} results in Capabilities of decreased sizes but increased coupling.

We formalize the steps for determining the abstraction level of a Capability in the scenario where its children share common functionality:

- If the Capability \((p)\) under consideration is too generic and subsequently, of a large size then lower its abstraction level to its \(k\) immediate children, \(p_1, \ldots, p_k\).

- Determine all nodes that are common to two or more \(p_i, i = 1, \ldots, k\); example \(n\) is common to \(p_1, p_2\) if there exists links \((p_1, n), (p_2, n)\).

- For each common node \(n\) determine its unique parent, and break common links.

- Compute coupling between \(p_1, \ldots, p_k\) with the new set of unique children.

- If the increase in coupling is offset by the advantages of smaller-sized Capabilities then let \(p_1, \ldots, p_k\) be the new set of Capabilities replacing \(p\)

- If the detrimental impact of coupling supersedes the advantages of reduced size then let \(p\) remain as the Capability.

Thus, we determine the abstraction level of Capabilities by subjectively analyzing the trade space between reduced sizes and increased coupling.

3.2.3.2 No Common Functionality

The other scenario, when lowering the abstraction level of a node, is that there are no common functionalities. As mentioned earlier, we prefer Capabilities of decreased sizes. By this
argument, one may conclude that the smallest sized nodes in an FD graph serve as Capabilities; with respect to the Figure 3.2, this implies that nodes \( n_{60}, n_{32}, n_{31}, n_{9}, n_{8}, n_{41}, n_{4}, n_{50}, n_{5} \) are all Capabilities. However, we hasten to add that node size (or the abstraction level) is only one aspect in identifying a Capability.

In reality, we first select specific sets of nodes from the FD graph based on their values of cohesion and coupling. During this process, we observe that nodes of a very small size (at the lowest level of abstraction) display high coupling and thereby, are removed from further consideration. We then examine the sizes of the remaining nodes to identify Capabilities. It is at this point that we determine the balanced abstraction level of Capabilities. From among the choices of different Capabilities, we choose those that exhibit the smallest size. For example, in the Course Evaluation System experiment, we choose Responses \( (n_4) \) and Reports \( (n_{63}) \) over their parent Results Analysis \( (n_{57}) \) (see Figure 3.5). This is because although there is a marginal increase in coupling, nodes \( n_4 \) and \( n_{63} \) are of smaller sizes when compared to \( n_{57} \). We are willing to accommodate this negligible increase in coupling for the convenience of increased modularity.

It is prudent to discuss the seemingly contradicting criteria for determining the abstraction level in the previous scenario where nodes had shared functionalities and the present scenario, where nodes are relatively independent. In the first scenario, we choose Evaluation Authoring \( (n_2) \) over its smaller-sized children that exhibit high coupling. In contrast, in the current scenario we choose the children over their parent, although their coupling is higher. We explain this conflicting choice: In an FD graph that is a tree (implying no shared functionalities), the coupling measure decreases with the reduction in the abstraction level of a node; the root is least coupled and the directives are most coupled. However, the increments by which the coupling values differs decreases with lower levels. For example, the coupling value of a combination of nodes at the highest level of abstraction is \( Cp(n_6, n_{57}, n_2) = 0.345338 \), and that at a lower level is \( Cp(n_6, n_4, n_{63}, n_2) = 0.957083 \). The percentage decrease is 139.2%. This is higher than the percentage decrease of 17.56% for combinations with coupling \( Cp(n_6, n_{60}, n_4, n_{32}, n_5, n_8, n_{31}, n_{50}, n_{41}, n_3, n_9) = 7.99848 \) and
$C_p(n_6, n_{60}, n_{32}, n_4, n_{50}, n_5, n_{31}, n_{41}, n_3, n_{55}) = 6.80333$, described at the lowest and second lowest levels of abstraction. Thus, when determining the abstraction level for the present scenario, we choose modularity over coupling. In contrast, for the previous scenario, it is obvious that nodes $n_{29}$ and $n_7$ are highly coupled because of their shared commonality. Hence, our decision in choosing the parent $(n_2)$ supports minimally coupled Capabilities over the benefits smaller size.

The overall algorithm for identifying Capabilities is discussed in detail in Section 3.3.2

### 3.3 Capability Identification Algorithm

Capabilities are identified using the three criteria of cohesion, coupling and abstraction level as discussed in Section 3.2. Each node in an FD graph represents some functionality of the system and hence, is a potential Capability. Therefore, the set of nodes that exhibit optimum values are the required Capabilities. We observe that the values of cohesion, abstraction level and size can be easily computed for an individual node. In contrast, the coupling metric is only applicable to a group of nodes because it measures the inter-dependencies between them. Thus, to determine the set of nodes that exhibit minimum coupling, we must examine all possible combinations and permutations of nodes in the FD graph. However, this is an NP-hard problem. Therefore, to reduce the complexity of this computation we first identify all slices from the FD graph. A slice is a valid combination of nodes that is meaningful enough to be considered as a set of Capabilities. We then evaluate each slice using each of the three criteria to determine the optimum set.

#### 3.3.1 Slices

We strive to reduce the complexity of evaluating all possible combinations of nodes in an FD graph by considering only those sets which are valid. For example, in Figure 3.2 the set
\(\{n_{59}, n_{60}, n_{30}\}\) is an unsound combination of Capabilities as they are a redundant portrayal of only a part of the system functionality. We define slices as being valid combinations of internal nodes from an FD graph where each node of a slice is a potential Capability. Thus, we enumerate specific constraints in choosing a valid set of Capabilities from the FD graph. For an FD graph \(G = (V, E)\) we define slice \(S\) as a subset of \(V\) where the following constraints are satisfied:

1. **Complete Coverage of Directives:** We know that a Capability is associated with a set of directives, which is finally mapped to system requirements (see Figure 3.1). Consequently, this set has to encompass all the directives resolved from user needs. The leaves of the FD graph constitute the set of all directives in a system. We ensure that each directive is accounted for by some Capability, by enforcing the constraint of complete coverage given by \(\bigcup_{i=1}^{m} D_i = \{L\}\), where
   - \(D_i\) denotes the set of leaves associated with the \(i^{th}\) node of slice \(S\)
   - \(L = \{u \in V | \text{outdegree}(u) = 0\}\) denotes the set of all leaves of \(G\)
   - \(m = |S|\)

2. **Unique Membership for Directives:** In the context of directives, by ensuring that each directive is uniquely associated with exactly one Capability we avoid implementing redundant functionality. Otherwise, the purpose of using slices to determine Capabilities as unique functional abstractions is defeated. We ensure the unique membership of directives by the constraint \(\bigcap_{i=1}^{m} D_i = \{\phi\}\).

3. **System Mission is not a Capability:** The root is the high level mission of the system and cannot be considered as a Capability. The cardinality of a slice containing the root can only be one. This is because including other nodes with the root in the same slice violates the second constraint. Also, such a slice represents an RE approach of development where the system is developed only on the basis of requirements. As CE follows a non-reductionist approach we enforce the rule \(\forall u \in S, \text{indegree}(u) \neq 0\).
4. **Directive is not a Capability:** A leaf represents a directive, which is a system characteristic. A slice that includes a leaf fails to define the system in terms of its functionality and focuses on describing low level details. Hence, \( \forall u \in S, \text{outdegree}(u) \neq 0 \).

Using the above defined constraints we enumerate all possible slices of the FD graph. A slice is a *valid combination* of nodes; each unique combination of these nodes is termed as a *basic slice*. An example of a basic slice of Figure 3.2 is \( S = \{n_2, n_6, n_{57}\} \). Permutations of a basic slice are also valid; for example permutations of \( S \) such as \( S_1 = \{n_{57}, n_6, n_2\} \), \( S_2 = \{n_6, n_2, n_{57}\} \) are all valid slices. We distinguish between the permutations because the coupling measure is asymmetric, i.e \( Cp(n_2, n_6) \neq Cp(n_6, n_2) \). Consequently, the coupling value of the permutations of a basic slice varies. In contrast, the values of cohesion and abstraction level are specific to an individual node, and thereby, are invariant across the permuted slices. Thus, we reduce the complexity of identifying all possible Capabilities by first determining the slices of an FD graph, and then applying the cohesion, coupling or abstraction level measures. The algorithm for Capability identification is formally described in the following section.

### 3.3.2 Algorithm

We now describe the major steps of the algorithm to identify Capabilities, and subsequently, discuss each step with respect to our experiment with the **Course Evaluation System**.

#### 3.3.2.1 Construct the FD graph

The FD graph for the **Course Evaluation System** is illustrated in Figure 3.2. This graph is constructed in accordance to the rules enumerated in Section 3.1. User needs and domain information were collected from documents created during the incipient stages of the project, and also elicited from the members involved in the system development.
3.3.2.2 Determine all slices

We identified a total of 33 basic slices and 53296254 associated permutations for the FD graph (see Figure 3.2) of the Course Evaluation System. Recall that a basic slice is a valid combination of nodes that satisfies the constraints described in Section 3.3.1. A permutation of a basic slice also fulfills the constraints; the only difference being that it exhibits a different ordering of nodes. Both basic slices and their permutations are considered slices of an FD graph.

3.3.2.3 Compute the cohesion of each slice

We compute the cohesion value of a slice as an average of the cohesion of its constituent nodes. For example, the cohesion value of \{n_6, n_{57}, n_2\}, is 0.75947. This is an average of the cohesion of the individual nodes: \(Ch(n_6) = 1.000\), \(Ch(n_{57}) = 0.590909\) and \(Ch(n_2) = 0.6875\). The cohesion measure of an internal node is defined in Section 3.2.1. The following steps describe the computation of the cohesion for a slice.

1. Ensure that every directive has exactly one parent

   We know that cohesion of a node measures the degree to which its directives work towards fulfilling the same functionality. It is possible that a directive may be common to two or more functions in an FD graph. For example, let us hypothetically assume in Figure 3.2 directive \(d_{10}\) has two parents \(n_9\) and \(n_8\), then in the actual system development, we may be faced with the dilemma of choosing in which function the directive is to be implemented. As discussed earlier, a directive should be implemented only once to avoid redundant functionality. Thus, deciding which parent the directive is associated with is crucial for the system development.

   For those cases where a directive has more than one immediate parent, we propose the following steps to ensure that a shared directive is aligned with exactly one parent only.
(a) **Compute the cohesion of each shared parent node with its original set of directives**

For our hypothetical example this would imply that the cohesion of nodes $n_9$ and $n_8$ be computed. Note that the original directive sets are $n_9 = \{d_{10}, \ldots, d_{15}\}$ and $n_8 = \{d_{10}, d_{16}, \ldots, d_{19}\}$ i.e the cohesion of both nodes are computed using $d_{10}$ as part of their directive sets.

(b) **Determine the unique parent of a common directive based on its percentage of contribution to the cohesion of each of its parents**

Specifically we compute the percentage contribution of the shared directive ($d$) to each of its parents ($n$) using:

$$\frac{\text{Rel}(d, n)}{\text{Cohesion}(n)} \times 100$$

where $\text{Rel}(d, n)$ is the relevance value of common directive $d$ to its shared parent $n$ and $\text{Cohesion}(n)$ is the overall cohesion value of parent $n$ (computed in the previous step). Continuing with our example, let us say that the percentage contribution of $d_{10}$ to $n_9$ is 68.39% and that of $d_{10}$ to $n_8$ is 21.87%. Then, we associate $d_{10}$ with $n_9$ and disassociate it from the directive set of $n_8$. Thus, the recomputed directive sets are $n_9 = \{d_{10}, \ldots, d_{15}\}$ and $n_8 = \{d_{16}, \ldots, d_{19}\}$.

2. **Determine the set of directives associated with each potential Capability based on the recomputed directive set obtained in the previous step**

In an FD graph, directives are denoted by leaves and internal nodes are potential Capabilities. If there exists a directed path from an internal node to a leaf then that leaf is a directive associated with a potential Capability. For example, in Figure 3.2, items ($n_{55}$) is a potential Capability associated with directives $d_{10}, \ldots, d_{19}$. Note that we are concerned with determining the directive set of each internal node because directives are transformed to requirements and thereby, serve as input to the development phases. On the other hand, the Capabilities influence the high-level design of the system, and in some sense, defines the modular aspects.
3. **Compute the cohesion and size of each potential Capability**

For the graph in Figure 3.2 we compute the cohesion of all internal nodes, which are potential Capabilities, using the cohesion measures defined in Section 3.2.1. The size of a node is the number of its associated directives or in other words, the cardinality of a node’s directive set. For example, the cohesion and size of $n_{55}$ are 0.74 and 10, respectively. Observe that because the issue of unique parent for a shared directive is resolved in Step 2, the directive set of $n_{55}$ accounts for only one instance of directive $d_{10}$.

4. **The cohesion of a slice is the average of the cohesion of each of its constituent Capabilities**

Thus, the cohesion of a slice is computed by applying the above described steps.

### 3.3.2.4 Compute the coupling value of each slice

We compute the coupling value of a slice as an average of the coupling of its nodes. For example, the coupling value of slice $\{n_6, n_{57}, n_2\}$, is 0.345338. This is an average of the coupling values: $Cp(n_6, n_{57}) = 0.520661$, $Cp(n_6, n_2) = 0.228889$ and $Cp(n_{57}, n_2) = 0.286465$. The coupling measure of the pairs $Cp(n_i, n_j)$ are defined in Section 3.2.2.

As an aside, the question that may arise at this point is - *Why should the order of nodes matter when computing the coupling of a slice?* To answer this, we consider another slice $\{n_{57}, n_2, n_6\}$, a permutation of the same basic slice. The coupling value of this permutation is an average of $Cp(n_{57}, n_2)$, $Cp(n_{57}, n_6)$ and $Cp(n_2, n_6)$, i.e we attempt to estimate the coupling of a permutation as the effect of change on $n_{57}$ if $n_2$ or $n_6$ change and the effect on $n_2$ when $n_6$ changes. This implies that there is a specific implementation order that exhibits the least coupling, a key property that is utilized when optimizing Capabilities in Phase II of the CE process (see Figure 3.1). Thus, the asymmetric nature of the coupling measure helps determine an optimal implementation schedule.
3.3.2.5 Select slices that exhibit high cohesion or low coupling

The number of slices for the FD graph of a complex emergent system can well exceed $10^7$. However, we are only interested in those slices that have a relatively high cohesion or low coupling. Thus, we select such slices by applying the following steps:

1. **For each basic slice set choose permutations that exhibit minimum coupling**

   From the set of all permutations of a basic slice, we choose those permutations that exhibit the lowest coupling value. Alternative selection paths include choosing minimally coupled permutations across all the basic slice sets. Our algorithm follows the former strategy because of the computational complexity of computing and comparing the coupling values of 53296254 permutations (an NP-hard problem). With respect to our Course Evaluation System experiment, we obtained 303 permutations that exhibited minimal coupling within their respective basic slice sets. The number of permutations exceed the number of basic slice sets (33) because there is more than one permuted slice with the same minimal coupling value. For example, for the basic slice set \{n_3, n_4, n_5, n_6, n_7, n_{31}, n_{32}, n_{50}, n_{60}\} there are 12 permutations with the same coupling value of 6.0222.

2. **Rank these permutations by cohesion and coupling**

   We rank the minimally coupled permutations by high cohesion and low coupling. Rank 1 represents high cohesion or low coupling. Note that two slices with the same cohesion or coupling value will have consecutively numbered ranks. Thus, the numbering of the ranks range from 1 to 303, the total number of minimally coupled permutations obtained from the previous step.

3. **Choose those slices that rank among the top 3 percent - either in the cohesion or the coupling category**

   Three percent of 303 slices is approximately 10 slices. Of course, the choice of percentage can vary, however, because we are concerned with those slices that exhibit high
cohesion and low coupling, we choose a low percentage value. Consequently, we restrict our analysis to a few select slices that are highly ranked. For the Course Evaluation System the slices selected in this process are enumerated, with their ranks, in Table 3.2. Note that a slice ranked highly in cohesion may be poorly ranked with respect to coupling, and vice versa. For example, slice 2, is ranked 2 in coupling but only 19 in the cohesion category. This also explains why there are 17 slices as opposed to 10.

3.3.2.6 Select slices whose cohesion values are above the overall cohesion average, and whose coupling values are below the overall coupling averages

Compute the average cohesion and coupling of the top 10 slices. Eliminate those slices whose cohesion value is less than the average cohesion value, and whose coupling value is greater than the average coupling value. The remaining slices are most optimum in terms of high cohesion and low coupling. For our experiment, from Table 3.2 we see that only slices 1 and 3 have a cohesion value that is higher than the overall cohesion average of 0.729798294 and a coupling value lower than the overall coupling average of 2.152371118.

3.3.2.7 The slice with balanced abstraction level is the desired set of Capabilities

Balanced abstraction level is determined by analyzing the trade-offs between reducing the node size and increasing coupling. Two possible scenarios can arise when considering nodes of a reduced size- these nodes may or may not share common functionalities (see Section 3.2.3 for a detailed discussion). With respect to the experiment we examine slices 1 and 3, output from the previous step, and observe that reducing the node sizes of \( n_2 \) of slice 1 results in common functionality, the first type of scenario. As described in Section 3.2.3.1 and shown in Figure 3.4 reducing the size of \( n_2 \), the node with the largest size in slice 1, results in nodes \( n_7 \) and \( n_{29} \) that share common functionality and thereby, exhibit content coupling. So breaking a common link and implementing a shared functionality (\( n_{55} \)) as part
of exactly one Capability violates the definition of Capabilities, as being minimally coupled units.

On the other hand, let us attempt reduce the size of \( n_{57} \), another node in slice 1. This results in the set of nodes \( \{n_6, n_4, n_{63}, n_2\} \), which is the same as slice 3. We observe that this reduction in size is that of the second type of scenario (see Section 3.2.3.2), where nodes do not have common functionalities. It is possible to reduce \( n_{63} \) even further to \( n_{50} \) and \( n_5 \). However the resulting slice 7, \( \{n_6, n_4, n_{50}, n_5, n_2\} \), has a poor cohesion, as seen from Table 3.2. This is further evidenced by the fact that this slice failed to qualify through the preceding steps of the Capability Identification algorithm.

Thus, based on the three criteria of cohesion, coupling and abstraction level, we choose slice 3 as our desired set of Capabilities for the Course Evaluation System. Derivation of a Capabilities-based design from identified set of Capabilities, and the impact of introducing random changes on this design is discussed in Section 4.1.

### 3.4 Conclusion

Capabilities Engineering employs a unique algorithm and set of well-defined metric computations that exploit the principles of decomposition, abstraction and modularity to identify functional aggregates, i.e. Capabilities, which embody the desirable software engineering attributes of high cohesion, low coupling and balanced abstraction levels. Change-tolerance is achieved through the embodiment of such attributes as evidenced by the experiment discussed in Section 4.1. In addition, the CE process structures the progression from the informality of the problem space to the formality of the solution space using Capabilities, facilitating pre-requirement specification traceability (see Chapter 5). Consequently, such a systematic approach promotes change-reduction during system development. An empirical validation of this claim is presented in Section 4.2.
Table 3.2: Slices Ranked in top 10 (either cohesion or coupling) of Course Evaluation System

<table>
<thead>
<tr>
<th>Slice#</th>
<th>Nodes in Slice</th>
<th>Cohesion</th>
<th>Rank\textsubscript{Ch}</th>
<th>Coupling</th>
<th>Rank\textsubscript{Cp}</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{n_6, n_{57}, n_2}</td>
<td>0.75947</td>
<td>1</td>
<td>0.345338</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>{n_6, n_{41}, n_{57}, n_{29}}</td>
<td>0.718727</td>
<td>19</td>
<td>0.826258</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>{n_6, n_4, n_{63}, n_2}</td>
<td>0.744792</td>
<td>4</td>
<td>0.957083</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>{n_6, n_3, n_{59}, n_{57}, n_7}</td>
<td>0.724626</td>
<td>17</td>
<td>1.28542</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>{n_6, n_4, n_{41}, n_{63}, n_{29}}</td>
<td>0.715133</td>
<td>22</td>
<td>1.57121</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>{n_6, n_4, n_5, n_{50}, n_2}</td>
<td>0.700833</td>
<td>83</td>
<td>1.84978</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>{n_6, n_4, n_{50}, n_5, n_2}</td>
<td>0.700833</td>
<td>84</td>
<td>1.84978</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>{n_6, n_{41}, n_3, n_{59}, n_{55}, n_{57}}</td>
<td>0.711633</td>
<td>25</td>
<td>1.92188</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>{n_6, n_4, n_3, n_{63}, n_{59}, n_7}</td>
<td>0.720648</td>
<td>18</td>
<td>2.0232</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>{n_6, n_{60}, n_3, n_{30}, n_{57}, n_7}</td>
<td>0.749358</td>
<td>2</td>
<td>2.20319</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>{n_{60}, n_6, n_3, n_{30}, n_{57}, n_7}</td>
<td>0.749358</td>
<td>3</td>
<td>2.20319</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>{n_6, n_{60}, n_4, n_3, n_{30}, n_{55}, n_{57}}</td>
<td>0.734688</td>
<td>7</td>
<td>2.88388</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>{n_{60}, n_6, n_4, n_3, n_{30}, n_{55}, n_7}</td>
<td>0.734688</td>
<td>8</td>
<td>2.88388</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>{n_6, n_{60}, n_4, n_3, n_{30}, n_{63}, n_7}</td>
<td>0.742415</td>
<td>5</td>
<td>3.19762</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>{n_{60}, n_6, n_4, n_3, n_{30}, n_{63}, n_7}</td>
<td>0.742415</td>
<td>6</td>
<td>3.19762</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>{n_6, n_{60}, n_8, n_{41}, n_3, n_9, n_{30}, n_{57}}</td>
<td>0.728477</td>
<td>9</td>
<td>3.69549</td>
<td>24</td>
</tr>
<tr>
<td>17</td>
<td>{n_6, n_{60}, n_8, n_{41}, n_9, n_3, n_{30}, n_{57}}</td>
<td>0.728477</td>
<td>10</td>
<td>3.69549</td>
<td>25</td>
</tr>
</tbody>
</table>
Chapter 4

Empirical Validation

Our approach for developing Capabilities-based systems touts two primary objectives — change-tolerance, and change-reduction. In order to validate these objectives, we perform empirical experiments on real-world systems, the Course Evaluation System and Sakai system. In particular, we use the former system to test the hypothesis that a Capabilities-based design improves a system’s change-tolerance. On the other hand, the Sakai system is used to test if the utilization of a CE process does reduce inherent change. In this chapter, we describe these experiments in detail, and discuss results in relation to the stated hypotheses.

4.1 Improving Change Tolerance: Course Evaluation System

A principal objective of the CE process is to construct complex emergent systems to accommodate change with minimum impact. To statistically validate if this objective is achieved by the use of Capabilities we examine the effect of changing needs on the RE and CE-based designs of the Course Evaluation System. This system is a cardinal part of a much larger complex emergent system and at the same time, is also envisioned to be used as stand-alone
software. For our empirical investigation we choose to examine such a system because it exhibits the characteristics of being complex, emergent and having an extended lifetime, and at the same time, is feasibly sized for our analysis.

We empirically analyze and compare the effect(s) of needs’ change on the RE and CE-based high-level designs of a real world system. For this, we consider a Course Evaluation System, which facilitates the creation, distribution, and collection, of course-evaluation forms. To test our hypothesis that a CE-based system accommodates change with lesser impact than an RE-based system, we examine six different change scenarios. We trace the ripple-effect of each change and record the number of affected classes in the original RE-based design, and our CE-based design. We observe that the difference in the number of impacted classes is statistically significant, thus implying that the utilization of Capabilities-based approach, improves the change-tolerance of the Course Evaluation System, when subject to the dynamic factor of changing needs.

First, we translate our research objective into a hypothesis that is viable for statistical testing. Then, we explain the experiment setup and the construction of the RE and CE-based designs. Subsequently, we discuss individual change scenarios and the statistical tests. Finally, we analyze the results, discuss their validity and present the implications and limitations of this experiment.

4.1.1 Hypothesis

We are interested in validating the change-tolerant properties of Capabilities generated by Phase I of the CE process described in Figure 3.1. Note that at this point Capabilities are associated with directives, which provide sufficient information to specify various high-level designs of the system. Our goal is to evaluate if the use of a Capabilities-based design improves the change-tolerance of the Course Evaluation System.

Before we test the validity of our approach, we need to discuss two aspects:
The measurement of change-tolerance, and

The comparison of the RE and CE-based designs for the Course Evaluation System.

We quantify change-tolerance as the number of entities impacted by a change, where the impact refers to any modification of the affected entity. Our objective is to test if the difference in the number of entities affected by change in an RE and CE-based design is statistically significant. Specifically, to prove our research claim that Capabilities-based design improves the change-tolerance of the Course Evaluation System we seek to reject the null hypothesis ($H_0$):

$$H_0: \text{The change-tolerance of a system is independent of an RE or a CE-based design}$$

and subsequently, accept the alternate hypothesis ($H_1$):

$$H_1: \text{The change-tolerance of a system improves with the use of a CE-based design}$$

To prove that the use of Capabilities introduces change-tolerance in a system we analyze the original design, a product of the traditional RE process (RE-based design), of the Course Evaluation System. Figure 4.2 presents the original RE-based high-level design. We compare this with a design influenced by the Capabilities identified from the FD graph of the Course Evaluation System (CE-based design), illustrated in Figure 4.1. The identification of these Capabilities is described in detail in Section 3.3.2.

In the RE-based design of Figure 4.2, the most abstract level entities are classes; the design has a predominantly flat structure. The classes are designated to fulfill some functionality, a choice that is more often based on intuition and convenience, than on results of a systematic analysis. In contrast, the CE-based design of Figure 4.1 is reflective of the Capabilities identified from the FD graph of the Course Evaluation System and illustrates
Figure 4.1: High Level Design based on CE

Figure 4.2: High Level Design based on RE

a more modularized high-level design. Here the most abstract entities are denoted by the emboldened boxes, which can be designed as classes. These represent a definite functionality that is highly cohesive and minimally coupled and specified at a balanced abstraction level.

In order to compare the change-tolerance of the two designs we subject them to a set of needs’ change-scenarios and record the impact in terms of the number of affected entities, i.e. count of the affected classes in the RE and CE-based designs of Figure 4.2 and Figure 4.1, respectively. Because we are interested only in the number of impacted classes, we refrain from discussing the members of each class, and focus only on the impacted functionality.
In some sense, this is similar to analyzing the responsibility of a class, an avenue for future research work.

We reiterate that the null hypothesis \( (H_0) \) states that there is no significant difference in the number of impacted classes in either approach. The alternate hypothesis \( (H_1) \) states that there is a significant difference between the number of entities affected by change in the RE and CE-based designs. More specifically, \( H_1 \), reflecting our research claim tests if the number of impacted classes is lesser in a CE-based design in comparison with a RE-based design. Thus, rejecting the null hypothesis implies that a CE-based design constructs change-tolerant systems.

The independent variables are the treatments in an experiment. Here they are represented by the CE and RE-based approaches that influence their respective designs. The dependent variables are the number of impacted classes. We analyze these variables to check if they are influenced by the different approaches. Hence, if the alternate hypothesis is true then one can attribute a decrease in the impact due to change to the use of Capabilities.

### 4.1.2 Experimental Setup

We construct the high-level design of the **Course Evaluation System** from its data model and class diagrams and it is consistent with the overall architecture of the actual system. Figure 4.2 illustrates this design in terms of the main classes. The edges between the classes indicate the existence of coupling and these are indicative of the dependencies between the implemented classes. For example, the edge from `Template_Item` to `Scale` implies that a change in the latter impacts the former.

The high-level design of the CE system is depicted in Figure 4.1. The entities of this design are the Capabilities identified from the FD graph of the **Course Evaluation System** using the algorithm described in Section 3.3.2. Specifically, these are nodes \{n_6, n_4, n_63, n_2\} from Figure 3.2.
Table 4.1: \# Impacted Classes in RE/CE-based design

<table>
<thead>
<tr>
<th>Scenario#</th>
<th>#Impact-RE</th>
<th>#Impact-CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Our experimental design has one independent variable with two values. We explain this by way of an analogy. Consider an experiment that uses the number of errors committed by participants to assess the efficacy of two tools, then by analogy, for our experiment the number of errors is equivalent to the number of impacted classes, the participants are akin to the change scenarios and the tools (treatments) are the CE and RE-based approaches. Hence, we use the standard design of comparing paired treatments, and therefore, perform Wilcoxon Sign Rank test [72]. We use a non-parametric test as opposed to a parametric test because the latter assumes that the data follows a normal distribution, an assumption that may not be valid in our experiment. Furthermore, the same set of change scenarios is subjected to two different treatments, thus ensuring that the same dependent variable is being measured each time.

4.1.3 Change Scenarios

We briefly discuss the impact of six scenarios of changing needs on the RE and CE-based designs for the Course Evaluation System. These cases are realistic changes and are currently not incorporated as part of the original functional specification. Table 4.1 presents a comparison between the designs in terms of the number of impacted classes in the RE-based design (#Impact-RE) and the CE-based design (#Impact-CE) for each scenario.

1. Teaching Assistants should be able to define their evaluations
Presently, only faculty are allowed to customize their evaluations. This change would require that the permission settings be modified. In the RE-based design, permissions are set and checked within each class, whereas in the CE-based design there is a separate class for implementing this functionality.

2. **Generate a report about the popularity of course CSXXXX over last 10 years**
   
   In the RE-based design reports are closely tied with course evaluation, the answers, and the context in which they are assigned. In the CE-based design reports are treated as independent modules that generate information in different formats, where the information is independent of the source. To fulfill this need, information is derived from the response set recorded over the years.

3. **Need information about the handicapped accessible facilities for courses taught in Room XXX**
   
   It would be feasible to produce this information as an independent report. However, the RE-based design forces the retrieval of any such information only through the data set of the evaluation forms. In contrast, in the CE-based design reports can independently process information and are decoupled from the data sources.

4. **Responses in different formats (video/ audio/ . . .) will be accepted from a student. This is to ensure that the student is not restricted to submitting feedback only when an online evaluation is scheduled**
   
   In the RE-based design for a video/audio response to be acceptable the evaluation has to be modified. This is because the responses are derived only from the answers of the evaluation and so, the answers have to be redesigned to accept videos. This implies that the item types may need to be modified, because answers are directly associated with items. On the other hand, in a CE-based design responses are independent classes and so can accept alternate formats as feedback, without having to manipulate other classes.
5. Need to statistically analyze Prof. X’s teaching abilities

As evident from this need, responses may need to be statistically analyzed to compute different measures. For example, such information is useful for decision regarding tenure cases, offering the same course in future and others. In the RE-based design, an evaluation is not persistent, i.e. the same evaluation can be deployed for different courses. However, as in the CE-based design, it is more prudent to relate responses to the courses than to a particular evaluation, for garnering such information.

6. Henceforth, all essay-type feedback items are to be qualitatively judged. Thus, disassociate them from the existing 5-point scale

This change-scenario impacts the item-scale association. In the RE-based design, two different classes have been defined for “items”- one is associated with the expert-template and the other with the customized-evaluation. Consequently, this need impacts more than one item-scale association. Hence, as suggested by CE-based design it is more convenient to define a single class for items for decreased impact.

Thus, both designs are examined and the number of impacted classes are recorded for each change-scenario as presented in Table 4.1. We reiterate that an impacted class is one that has to be changed to accommodate a modified need. This echoes our definition of change-tolerance, the ability of a system to accommodate change with minimum impact; in the experiment the impact is measured in terms of the number of affected classes.

### 4.1.4 Analysis of Results

We now discuss the results of the statistical test, and subsequently, examine the validity of the experiment.

We use Minitab for performing the Wilcoxon Signed Rank test, the non-parametric alternative to paired t-test, on the data presented in Table 4.1. The output of this test is a P-value,
which indicates the probability that the population medians of the number of affected classes in the RE and CE-based designs are different because of chance. The computed P-value of 0.018 implies that there are non-random factors that cause the decrease in number of classes impacted in the CE-design when compared to the RE-design. Specifically, one can attribute this difference to the key factor of using RE and CE-based approaches to develop high-level designs. Consequently, the very small P-value compels us to reject the null hypothesis (at levels greater than 0.018) that the change-tolerance of the system is indifferent to either the CE or the RE approach.

Thus, the alternate hypothesis that the number of impacted classes in the CE-based design is significantly lesser than that of the RE-based design is true.

This result is in agreement with our research claim that the change-tolerance of a system improves with the use of a design based on Capabilities.

However, it is prudent to examine the different threats to the validity of this experiment prior to the acceptance of these results [73].

Internal Validity: The internal validity of an experiment is compromised if there are unknown factors that influence the dependent variable. In our experiment the dependent variable is the number of classes affected by change. A factor that may influence this is bias in the high-level RE/CE-design construction.

The RE-based design is reflective of the original data model and the class diagrams of the Course Evaluation System. In addition, this design, although a snapshot of the system at a particular point in time, is validated by the members in the project. Thus, there is negligible bias in the construction of the RE-based design.

The CE-based design is derived from the Capabilities identified in the FD graph illustrated in Figure 3.2. The detailed computations for identifying the set of Capabilities are presented in Section 3.3.2 along with the values of the three criteria of cohesion, coupling and balanced
abstraction levels in Table 3.2. Hence, the CE-based design is the result of an unbiased and systematic construction.

Conclusion Validity: This type of validity is concerned with incorrect conclusions drawn about the relationship between the treatment and the outcome. This could arise from violating assumptions of statistical tests, having low statistical power, unreliable measurements and others. The use of a non-parametric test ensures that we fulfill required statistical assumptions. The change-scenarios explained in Section 4.1.3 and the RE/CE-based designs illustrated in Figures 4.2 and 4.1 facilitate the individual verification of the number of impacted classes, thus ensuring reliability.

Construct Validity: Construct validity ensures that our research claim is actually investigated in the experiment. It answers the question if the Capabilities-based approach is really responsible for improving the change-tolerance of the system. As presented in Section 4.1.1, the hypothesis tested in the experiment is derived from our research hypothesis. We attempt to establish the relation between the change-tolerance of the system and the design utilized to develop it, by using the same set of change-scenarios on a RE and CE-based design, and explaining the impact of each change in detail. In addition, the changes introduced are realistic and representative of actual user demands. Construct validity also examines if the notion of change-tolerance has been measured accurately. We use the number of impacted classes as an indicator for change-tolerance, a measure that captures the impact of change in a high-level design.

External Validity: This type of validity is related to the ability to generalize the result that the use of a Capabilities-based design improves the change-tolerance of all complex emergent systems. We refrain from generalizing these results for all systems. Although, the Course Evaluation System exhibits the desired characteristics for validation, it is a relatively smaller-sized system in the class of complex emergent systems. Furthermore, we
need to quantify all aspects of design and development, for a result to be valid in the universe of all systems; an impossibility because software construction is and inherently sociological activity. An example of an unquantifiable aspect in our process is the FD graph. Although we provide objective construction rules, the FD graph of a system is not necessarily unique and is dependent on the indeterministic thought processes of the analyst and users of the system.

4.2 Change Reduction: *Sakai Experiment*

We use the **Sakai** Collaboration and Learning Environment, an open-source, enterprise-scale software application, to validate if the CE process tends to reduce change inherent in a development process. **Sakai** is being developed by an international alliance of several universities spanning four continents [74]. The current Sakai system consists of about 100,000 lines of code and its complexity is further compounded by distributed development. The high-level mission of this system is to realize specific research, learning, collaboration and academic needs of universities and colleges. The system is constantly evolving to accommodate the needs of its 300,000+ users. System increments that incorporate new functionalities are released on a yearly basis. Also, the overall system is envisioned to be used for an extended time-period. Hence, the Sakai system exhibits characteristics of complex emergent systems and appears suitable for the purpose of validating the other objective of change-reduction.

Change requests for the **Sakai** system are documented in a comprehensive repository. These requests encompass user needs, new system functionality, programming bugs, usability changes, and other types of issues. We consider requests related to system functionality, and examine if the use of a CE process may have inhibited the generation of such a change. For this, we classify each change and analyze the potential cause for its occurrence. Examples of such causes include modularization, missing domain information, incomplete needs, and others. Based on the cause we determine if characteristics of the CE process could have prevented
or, on the other hand, could have generated such a change request. Thus, to test our hypothesis that the CE-process generates lesser number of change-requests than an RE-process, we select a random set of change-requests and analyze their causes. We observe, that across a span of 18 months, the number of requests generated by the CE-process is significantly lesser than that of the RE-process.

In the following sections, first, we formulate the hypothesis to reflect our research objective. Then, we explain the experiment setup and the statistical tests. Finally, we discuss the results, analyze their validity and present the conclusions of this experiment.

4.2.1 Hypothesis

We are interested in validating the change-reduction properties of the CE process described in Figure 3.1. Our goal is to evaluate if the use of a CE process reduces the number of change-requests generated when developing the Sakai system.

We quantify change-reduction by measuring the number of change-requests caused by the development process. The objective is to test if the difference in the number of change-requests generated by the RE and the CE process is statistically significant. Specifically, to prove our research claim that the CE process reduces inherent change in the development of Sakai we seek to reject the null hypothesis ($H_0$):

$$H_0: \text{The number of change-requests generated during system development is independent of an RE or a CE process}$$

and subsequently, accept the alternate hypothesis ($H_1$):

$$H_1: \text{The number of change-requests generated during system development reduces with the use of a CE process}$$
To prove that the use of CE process introduces change-reduction in a system we analyze the documented change-requests of the Sakai system. A snapshot of the recorded requests is shown in Figure 4.3. Each change-request has a detailed description, comprehensive change history, and comments by the developers and stakeholders. Information regarding the components, and versions of Sakai that are affected by each request are also detailed.

In order to compare the change-reduction properties of the two development processes we examine a random set of changes, and categorize them according to their potential causes.
We then analyze if some aspect of the RE/CE process could have inhibited this cause and thereby, prevented the generation of such a change-request.

We reiterate that the null hypothesis ($H_0$) states that there is no significant difference in the number of change-requests generated by either approach. The alternate hypothesis ($H_1$) states that there is a significant difference between the number of requests caused by the RE and CE-based processes. More specifically, $H_1$, reflecting our research claim tests if the number of change-requests caused by the CE process is lesser in comparison with the number generated by the RE process. Thus, rejecting the null hypothesis implies that a CE process does reduce change when developing complex emergent systems.

4.2.2 Experimental Setup

We choose to examine the change request set of Sakai version 2.2. Because Sakai is a constantly evolving system, we scope our analysis to one particular version of the system. Consequently, the FD graph used in the CE process is constructed using feature list and user documentation specific to version 2.2. Sakai uses Jira [75] to track a variety of issues such as bug reports, tasks, and feature requests; a snapshot is shown in Figure 4.3. The time-stamp of an issue is the date on which it has been created in Jira. These requests are ordered by ascending order in time. We consider requests time-stamped within a sequence of two months as a single sample. For example, the first sample is requests recorded for the months of January and March 2005. There were none with a time-stamp of February 2005. Thus, in this manner, we consider 9 samples spanning from January 2005 to July 2006, each of varying sizes. These requests are independent of any specific ordering or aggregation by component. Hence, a selection of requests based entirely on the time-stamp provides a random set.

Our experiment examines if some aspect of the CE/RE process is responsible for generating the change-request. Thus, the processes are the treatments, and the number of change-requests are the dependent variables. Because we subject the same request to both the
RE and the CE process, we use the standard design of comparing paired treatments, and therefore, perform Wilcoxon Sign Rank test \[72\]. As with the Course Evaluation System experiment we use a non-parametric test as opposed to a parametric test because the latter assumes that the data follows a normal distribution, an assumption that may not be valid in our experiment.

4.2.2.1 Potential Causes

We examine each request and identify a possible cause for it. Broadly speaking, a request can be caused because of a bug (programming error), or is an issue caused by some technical constraint. It can also be a request for some enhancement of the existing tool - *viz.* new functionality. However, we are not concerned with new user needs because this kind of change has been validated in the experiment with the Course Evaluation System in Section \[4.1\]. We are specifically interested in examining those requests generated because of certain aspects of the requirements, and design phases. We identify three broad categories of potential causes, and relate them to characteristics of the CE process:

- **Missing Information:** This category can result from incomplete requirements, loss of information when transitioning from the problem to the solution space, ambiguous specification, and other similar factors. The FD graph of the CE process facilitates the visualization of information at different levels of abstraction. Consequently, directives furnish low-level details that are further mapped to more specific system requirements. On the other hand, visualizing nodes at a higher level of abstraction facilitates a high-level vision of the system, and thereby, preventing the inclusion of redundant functionality. An example of a change-request caused by missing information is described in Table \[4.2\].

- **Modularization:** Often, the decomposition of the system into components is influenced by factors of convenience rather than deliberate strategies. For example, factors
Table 4.2: Example: Missing Information Category

<table>
<thead>
<tr>
<th>Problem</th>
<th>RE Process</th>
<th>CE Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK3824—Calendar templates make assumptions about date formats</td>
<td>Lack of precise information regarding the formats supported in Calendar</td>
<td>Date formats are not localizable. There is an assumption that mm/dd/yy is the only format</td>
</tr>
</tbody>
</table>

such as geographic location of teams, and pre-existing code base are responsible for the modularization of the Sakai system. This kind of classification may result in lack of cohesive modules, and tight coupling among the components. In contrast, Capabilities are identified such that they are highly cohesive, and loosely coupled and exhibit balanced levels of abstraction. These characteristics lead to reduction in the kind of change requests generated because of poor modularization; such an example is illustrated in Table 4.3.

Table 4.3: Example: Modularization Category

<table>
<thead>
<tr>
<th>Problem</th>
<th>RE Process</th>
<th>CE Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK144—Currently irrelevant calendar.new &amp; calendar.delete permissions</td>
<td>Permissions and Schedule module are tightly coupled</td>
<td>Permissions were intrinsically designed to be a part of the Schedule tool causing the change-request</td>
</tr>
</tbody>
</table>

- **Unstructured Process:** Complex emergent systems are constantly evolving, and require the addition of new functionality over time. Introducing new components in a random manner can result in issues of scheduling, resource management, inconsistency, and others. In Sakai this is evident with the number of change-requests related to presentation, and consistency of the tools. The Capability set of the CE process provides an estimate of the expected functionality, and facilitates a platform for estimating
budget, standardizing formats, and allocating resources. Note that we compute the set of Capabilities for the Sakai system using the algorithm described in Section 3.3. The Capability set is comparable to the set of components in Sakai. An example of a change-request regarding consistency across different tools is detailed in Table 4.4.

Table 4.4: Example: Unstructured Process Category

<table>
<thead>
<tr>
<th>SAK3420-Inconsistency with button types in Membership</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROBLEM</strong></td>
</tr>
<tr>
<td>RE PROCESS</td>
</tr>
<tr>
<td>CE PROCESS</td>
</tr>
</tbody>
</table>

The list of causes responsible for such design issues is summarized in Table 4.5; each cause is described along with the element in the CE process that inhibits the corresponding cause.

Table 4.5: Potential Causes- Summary

<table>
<thead>
<tr>
<th><strong>CAUSE</strong></th>
<th><strong>DESCRIPTION</strong></th>
<th><strong>ELEMENT OF CE PROCESS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Missing Information</td>
<td>Incomplete requirements, loss of domain information</td>
<td>Directives, FD Graph</td>
</tr>
<tr>
<td>Modularization</td>
<td>Improper module classification, random component creation</td>
<td>Capability Identification Criteria</td>
</tr>
<tr>
<td>Unstructured Process</td>
<td>Intuitive design decisions</td>
<td>Capability Set</td>
</tr>
</tbody>
</table>

4.2.2.2 Change-Requests Analysis

We examined 117 change-requests, grouped into 9 samples based on the recorded timestamp. The percentage of change-requests generated by the RE/CE process is computed for
Table 4.6: Percentage of Change Requests Caused by RE/CE Process

<table>
<thead>
<tr>
<th>Sample #</th>
<th>Size</th>
<th>% Caused-by-RE</th>
<th>% Caused-by-CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>71.43</td>
<td>14.29</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>35.71</td>
<td>7.14</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>31.25</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>37.5</td>
<td>6.25</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>35.25</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>35.29</td>
<td>11.76</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>23.81</td>
<td>0</td>
</tr>
</tbody>
</table>

purposes of comparison. For example, Sample #6 has 16 change requests of which 37.5% are caused by the use of an RE process, as opposed to the 6.25% caused by the CE process. The samples, their sizes, and the percentage of change-requests generated by the RE and the CE process are presented in Table 4.6.

A detailed report of the change-requests and their associated analysis with regards to the RE and CE process is described in Appendix A.2. Such an analysis requires the computation of Capabilities of the Sakai system. Thus, a detailed description of the FD graph construction, and the Capabilities identification process is also presented in Appendix A.1. In the following section we discuss the results of this experiment, and correspondingly, confirm or refute our research hypothesis. In addition, we also discuss the validity of such an experiment.

4.2.3 Analysis of Results

We use Minitab for performing the Wilcoxon Signed Rank test, the non-parametric alternative to paired t-test, on the data presented in Table 4.6. The output of this test is a $P$-value, which indicates that the difference in the median number of change requests generated by CE process and RE process is by chance. The extremely small computed $P$-value of 0.0002 implies that 99.9998% of time there are non-random factors that cause the decrease in number
of requests. Specifically, one can attribute this difference to the key factor of using RE and CE processes. Consequently, the very small P-value compels us to reject the null hypothesis that the change-reduction is indifferent to either the CE or the RE process.

Thus, the alternate hypothesis that the percentage of change-requests caused by the CE process is significantly lesser than that of the RE process is true.

This result is in agreement with our research claim that number of change-requests generated during system development reduces with the use of a CE process.

However, it is prudent to examine the different threats to the validity of this experiment prior to the acceptance of these results [73].

Internal Validity: Error in estimating the dependent variable can affect the internal validity. The number of change-requests caused by the RE process is estimated based on documentation recorded by the development team, in the change repository. For the CE process, we construct the FD graph in collaboration with the Sakai community, and identify the set of Capabilities using an objective algorithm (see Section 3.3.2 and Appendix A.1). Hence, estimates of the dependent variable are devised using authentic, and unbiased data.

Conclusion Validity: Conclusion validity is challenged when violating assumptions of statistical tests, utilizing low statistical power, drawing incorrect relations between the treatment and outcome, and others. We use a non-parametric test to ensure the required statistical assumptions are fulfilled. In addition, specific aspects of the CE process that inhibit the generation of the change-requests are described in detail in Sections 3 and 4.2.2.1.

Construct Validity: Construct validity ensures that our research claim is actually investigated in the experiment. It answers the question if the CE process is really responsible for
change-reduction during system development. As presented in Section 4.2.1, the hypothesis tested in the experiment is derived from our research hypothesis. We attempt to establish the relation between the change-reduction and the development process, by examining the same set of change-requests in relation to the RE and CE processes. In addition, these changes are documented and explained by the development teams in the system repository. Construct validity also examines if the notion of change-reduction has been measured accurately. We use the number of change-requests as an indicator for change-reduction, a realistic estimate given that requests are an inherent measure of change in the system.

**External Validity:** This validity allows one to generalize the result that the use of CE process reduces change when developing complex emergent systems. We refrain from generalizing these results for all systems. Although, the Sakai is an example of a complex emergent system, we need to quantify all aspects of design and development, for a result to be valid in the universe of all systems. This is challenging given that software engineering has several intangible aspects that are difficult to quantify.

Thus, the experiments with the Course Evaluation and Sakai systems establish the change-tolerance and change-reduction properties of Capabilities and the CE process, respectively. Repeated experiments with other systems that display characteristics of complex emergent systems will further strengthen the validity of our approach.
Chapter 5

Pre-Requirement Specification

Traceability: Bridging the Complexity Gap through Capabilities

5.1 Introduction

Current traceability techniques falter when subjected to the dynamics of requirements evolution, user needs volatility, market vagaries, technology advancements and other change-inducing factors that plague software systems during their development cycles. Undoubtedly, the inability to precisely capture and represent the effect(s) of each change has contributed to the long history of system failures [2]. Although, the importance of traceability, and in particular requirements traceability (RT), was recognized in the early nineties [76], the research community is still grappling with the challenges of traceability, especially between requirements and their source needs. In large part, this is because of the difficulty in formalizing user needs, which are often unstructured information. Although, needs are informal expectations of users, they serve as the primary source for requirements specification. If a
software system is to exhibit the “right” functionality, then it is imperative that each system requirement satisfies one or more user needs.

Because system validation is performed against requirements, it is critical that we have the ability to trace requirements back to their source needs. This type of traceability is known as pre-Requirement Specification (pre-RS) tracing [76]. In fact, it is empirically established that inadequate pre-RS tracing is far more responsible than post-RS tracing (tracing requirements to design/code artifacts) for defective RT [77] [78]. We conjecture that this is because post-RS tracing works within the convenience of the solution space, mapping requirements to design or code entities.

However, pre-RS tracing is required to trace entities from the problem space (needs) to the solution space (requirements). The chasm between these spaces, i.e. the complexity gap [79], is too large of a leap for current traceability techniques. In addition, the effects of this gap — manifested as loss of domain information, misinterpreted requirements, misconstrued needs — are exacerbated during the development of large-scale systems. We term the intermediary region that includes the complexity gap as the transition space and use it to ease the giant leap from needs to requirements. Figure 5.1 illustrates the difference between the traditional requirements engineering approach and our solution approach in advancing from the problem to the solution space.

Our solution approach for pre-RS traceability is derived from Capabilities Engineering (CE), a process for architecting change-tolerant systems by constructing functional abstractions termed Capabilities in the transition space. Capabilities influence the basic composition of systems, and in some sense, impose a high-level architecture. Capabilities are formulated from user needs and mapped to system requirements. In the process, they occupy a position that is neither in the problem space nor in the solution space. More specifically, although Capabilities are derived from user needs, they are imbued with design characteristics of cohesion and coupling. This introduces aspects of a solution formulation, and thus, discourages the membership of a Capability in the problem space. On the other hand, Capabilities are
less detailed than entities that belong to the solution space. Consequently, Capabilities fit more naturally in the transition space. Furthermore, their formulation from the user needs and mapping to requirements imply that they have the potential to bridge the complexity gap; thus assisting the traceability between needs and requirements as depicted in Figure 5.1. Moreover, the inherent ability of Capabilities-based systems to accommodate change with minimum impact enhances the efficacy of traceability; random, unstructured ripple-effect impairs the strength of regular traceability techniques.

The Capabilities-based development approach strives to accommodate change with minimum impact, and therefore, incorporates pre-RS tracing in its process; traceability is the cornerstone of change-management. The use of the transition space facilitates the capture of domain information, and preserves relationships among needs and their associated functionalities during the progression between spaces. On the other hand, the characteristics of high cohesion and low coupling of Capabilities, support traceability in evolving systems by localizing and minimizing the impact of change. The ability to trace is unhindered by the system magnitude when utilizing a Capabilities-based development approach because traceability techniques are embedded into the process. Moreover, by considering traceability
as an integral part of the development effort, several issues relating to human factors can be resolved. For example, often the importance of traceability is undermined because it is regarded as a time-consuming activity when executed in isolation.

The remainder of the chapter is organized as follows: Section 5.2 discusses related work on traceability. In Section 5.3, we define each space, discuss the role of CE in facilitating traceability within each space and examine the spaces’ connectivity in terms of common linkages. Then, in Section 5.4 we present how the use of a Capabilities-based development approach addresses specific challenges of traceability. Our conclusions are surmised in Section 5.5.

5.2 Background

We first clarify the usage of the terms “requirement” and “requirement specification” in the context of our research. A common definition of RT is the “ability to describe and follow the life of a requirement, in both a forwards and backwards direction” [76]. Forward traceability is tracing a requirement to its design or code entities, and backward traceability is tracing a requirement to its sources [80]. According to these definitions a requirement is not necessarily a part of a specification document. Pre-RS tracing, however, refers to the traceability aspects of a requirement before its inclusion in a specification document. Consequently, there is an overlap between the different modes of traceability; this is graphically illustrated in [81]. This overlap stems from the fact that a requirement is iteratively refined until it is suitable for a formal specification. As a result, any statement that describes what is expected from the system, irrespective of its level of refinement, is termed a requirement. However, we make a distinction with respect to the terminology used. We consider a requirement as a statement that is formally recorded in a software requirements specification document [43]. Therefore, we consider the terms requirement and requirement specification as one and the same, and so use them interchangeably.
Several models have been constructed to assist pre-RS traceability. For example, Contribution Structures [82] consider the role of users, personnel, and others, in eliciting information to trace the origin of requirements. Similarly, Yu and Mylopoulos [83] factor in the influence of the relationships between stakeholders on the RT process. These methods emphasize the social aspect of requirements elicitation. On the other hand, Pohl [84] tailors the RE environment to capture traceability information between needs and requirements using PRO-ART. More general reference models for traceability have been developed by Ramesh and Jarke [85]. In the Capabilities-based development approach, traceability information is more of an implicit by-product.

In the following section we discuss how the relationship between needs, directives, Capabilities and requirements, as defined by the CE process, aids in pre-RS traceability and conceptually prove that Capabilities help bridge the complexity gap.

5.3 Role of CE in Pre-RS Tracing

Traditional RE transitions directly from needs to requirements, as illustrated in Figure 5.1. Needs are the primary source of information for system development. They are stated in a natural language form, and thereby, can often be ambiguous, vague and misleading. Hence, needs are unsuitable as input to the design phase; instead, we utilize system requirements derived from user needs. Thus, there is a transition from needs to requirements. These requirements are more formal, and display quality characteristics such as accuracy, unambiguity, testability, and others [86]. Although, formalization of requirements does reduce the possibility of misinterpretations, it usually fails to convey pertinent problem domain information. In fact, it has been recognized that the informality of a natural language has the advantage of communicating certain knowledge that formalization neglects to capture [87]. This loss of information has been identified as a key issue in RE [44].

We claim that there exists an intermediary space, which symbolizes a middle-ground be-
tween the extremes of formality and informality. This space provides an opportunity to metamorphose from the natural informality of the problem space to the rigid formality of the solution space in a more deliberate and systematic manner. In addition, the consequences of a direct leap from the problem domain to the solution domain — misinterpreting needs, missing requirements, loss of domain information and so forth — can be mitigated. We term the intermediary space that includes the complexity gap as the transition space. Again, Figure 5.1 graphically illustrates how CE uses the transition space to decrease the leap from problem to solution space. In other words, Capabilities assist in a smoother transition from needs to requirements.

Sections 5.3.1, 5.3.2, and 5.3.3 discuss in detail the problem, transition and solution spaces, respectively. In particular, we first define and describe the elements of each space, and then discuss how traceability is achieved within that space. Lastly, in Section 5.3.4 we present a unified perspective on the connectivity between the three spaces, and illustrate how pre-RS traceability is automatically supported by the activities of the CE process.

5.3.1 The Problem Space

The distinction between a space and a domain is often blurry and is used interchangeably. For example, the term problem space is used to indicate a conceptual region of relevance associated with the problem area [88] [89]. However, in some instances the term problem domain is also employed to imply the same [90] [26]. For the purposes of clarity, we utilize the notion of problem and solution domains as discussed by Hull et al. to describe spaces [91]. More specifically, we characterize a space in terms of three elements: the view, the domain and the resident entities. By this characterization, the problem space is composed of the entities: needs and directives. These entities are defined from a user’s view and are described in the language of the problem domain. Formally, the problem space is a collective aggregation of user view, problem domain, needs, and directives. Activities such as problem identification, decomposition, domain analysis [92] and others that help impart
a better understanding of the problem, are performed in this space. An application of the user view on the problem domain generates needs, which are, to a large extent, informal and unstructured. A pictorial representation of the problem space is shown in Figure 5.2.

Figure 5.2: Problem Space

- **User View**: This refers to the perspective of a stakeholder who is interested in the system to be developed. The user view includes both direct and indirect viewpoints.

- **Problem Domain**: The problem domain denotes the knowledge area(s) relevant to the problem being solved. For example, if the problem is to build an ATM, then the problem domain is banking.

- **Needs**: A need specifies what is desired of the system from a user’s perspective, and is stated in the language of the problem domain. It is obtained by applying a user view on the problem domain. It is composed of objects and operations of the problem domain as perceived by the user. Resulting needs cause the generation of other needs. User views can also activate other user views. Hence, every element feeds into the problem space as an input to produce more needs.

- **Directives**: We define a directive as a detailed characteristic of the system formulated
in the problem domain language. It can be regarded as a requirement with context information. In the problem space, the purpose of a directive is two-fold. First, it helps capture domain information. Second, it facilitates the progress from the problem space to the transition space.

An example of a user need, a directive, a Capability and a requirement, for a hypothetical course management system is described in Table 5.1.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need</td>
<td>Need a facility for students and faculty to share ideas, discuss questions</td>
</tr>
<tr>
<td>Capability</td>
<td>Discussion Forum</td>
</tr>
<tr>
<td>Directive</td>
<td>Provide a separate section for faculty to post important announcements</td>
</tr>
<tr>
<td>A Requirement</td>
<td>For the announcement section, the write permission must be enabled only for users designated as faculty.</td>
</tr>
</tbody>
</table>

Table 5.1: Examples in the context of a Course Management System

We begin the process of CE by discovering, eliciting and understanding needs from the user. The “needs” component of the CE process, and the subsequent decomposition to directives illustrated in Figure 3.1 is confined to the problem space. In the following section, we explain how the deliverable of the decomposition activity — the FD graph — aids traceability.

5.3.1.1 Traceability in the Problem Space

The FD graph, resulting from the activity of decomposition, assists in pre-RS traceability in the following ways:

1. By the virtue of construction, the FD graph represents, in some sense, dependency links between user needs. A change in a parent node affects one or more of its children.
Therefore, the effect of a change in a need is more easily traced to its dependent constituents. Subsequently, the FD graph facilitates traceability among needs. In contrast, present traceability methods are more concerned with the traces between requirements and its sources, rather than links within the sources themselves. As a result, a changed need is traced only to its associated set of requirements; there is insufficient information to assess the impact of change on other needs, and subsequently, their requirements. Thus, we claim that this type of traceability is important to understand the ripple-effect of change within needs, so as to completely record the effect on requirements. When utilizing the FD graph, explicit efforts to maintain traces among the source needs are unnecessary.

2. The FD graph presents an intuitive decomposition of functionality expected of the system to be developed. In the process of decomposition, nodes are described at different levels of abstraction. This kind of a visual representation encourages the user to focus on the functionality of the system, rather than on the low-level details. In addition, the hierarchical decomposition of the needs’ functionality permits a more systematic approach to stating what is expected of the system. This reduces the possibility of representing conflicting statements, redundant functions, and inconsistent expectations across different user classes. Furthermore, the FD graph, structures the informality of problem space, and thereby, provides an opportunity to automate the traceability process.

3. The leaves of the FD graph are directives, which are described in the language of the problem domain. The set of all directives represents the entire system functionality. Figure 3.1 indicates that directives are used to formulate Capabilities that occur in the transition space. More specifically, directives are the connecting links between the problem and transition spaces, and thereby, help preserve and transfer the problem domain information from the former to the latter. We claim that directives implicitly
aid pre-RS traceability, by serving as inter-connections between the disparate spaces. Section 5.3.2 elaborates on this aspect.

It is inevitable that a large part of the needs elicited from different sources is inconsistent or conflicting [94]. However, the use of an FD graph helps capture needs and represent desired system functionalities in a systematic manner. Furthermore, the graph also represents functions at different levels of abstraction, which aids in understanding their relative importance. For example, fulfilling the overall mission of the system is more important than implementing a directive. The structure of the graph facilitates traceability among needs by introducing aspects of formality in the highly irregular problem space.

Recall that the leaves of the graph are directives, which connect the problem and the transition spaces. In the following section, we discuss traceability within and between Capabilities, in the transition space.

5.3.2 The Transition Space

In the transition space, we introduce specific design characteristics of cohesion and coupling. We know that entities with these properties localize and minimize the propagation of change, which is crucial for promoting change-tolerance and also for structuring pre-RS traceability. We move away from the informality of the problem space by adopting a system view — introducing design aspects of cohesion and coupling — on the expected functionality of the system. However, we still utilize the problem domain but generate new entities termed Capabilities. Thus formally, the transition space is defined as a collective aggregation of system view, problem domain, and Capabilities. A pictorial representation of the transition space is shown in Figure 5.3. Note that directives are also present in the transition space because they facilitate the change from the problem to the transition space. They originate, however, in the problem space, and so belong there.

- **System View:** We define system view as the software engineering perspective that
guides the identification of Capabilities based on the design principles of high cohesion, low coupling, balanced abstraction levels, as constrained by schedule and technology. Figure 5.3 illustrates that initial Capabilities are formulated from directives. These initial Capabilities are iteratively optimized to produce an optimized set of Capabilities. In essence, formulation and optimization, the activities of the CE process as shown in Figure 3.1, incorporate the system view to produce Capabilities.

• **Capabilities:** We describe Capabilities as functional abstractions that exhibit high cohesion, low coupling and are defined at balanced levels of abstraction. The optimized set of Capabilities, chosen to develop the system, reflect the constraints of technology feasibility and implementation schedules.

The input to the transition space is an FD graph, which describes expected system functionalities at varying levels of abstraction, and directives that help realize these functions. The activity of formulation, uses these directives to identify initial sets of highly cohesive, minimally coupled Capabilities. Then, the optimization activity applies the constraints of schedule and technology to determine the final set of Capabilities.
5.3.2.1 Traceability in the Transition Space

We now discuss how the structure and characteristics of a Capability and its associated directives assist in pre-RS traceability in the transition space.

1. Capabilities are functional abstractions identified from the FD graph, and are associated with a set of directives. Capabilities provide stakeholders with a high-level perspective of the system functionality. In contrast, directives within each Capability furnish the more rudimentary details. We claim that the structure of Capabilities and their directives serve the pre-RS traceability interests of two disparate groups of users — high-end and low-end users — as identified by Ramesh [95]. High-end users work with complex systems that have a large number of requirements, and can easily become entangled in the details. However, Capabilities can be utilized to understand, from a high-level perspective, the expected system functionalities, and thereby, ensure that user expectations are satisfied. In contrast, low-end users are known to neglect pre-RS traceability because they are more concerned with detailed requirements. In such a case, these users can focus on directives to trace the origin of requirements because they are similar to detailed requirements but are stated in the language of the problem domain. Thus, we claim that Capabilities and their directives help overcome certain shortcomings of pre-RS traceability among different groups of users.

2. Capabilities exhibit high functional cohesion. Each associated directive, with varying degrees of relevance, is essential for realizing a Capability. A change in a directive may affect the parent Capability, and also other associated directives. Thus, Capability-directive links established by the FD graph can be utilized for the purposes of traceability. In other words, the property of high cohesion facilitates traceability within a Capability.

3. Minimally coupled Capabilities reduce the overhead of maintaining traceability information between each and every directive. This is because low coupling implies
decreased dependencies, and therefore, reduces the need for traceability paths between certain entities.

4. One can use information from the FD graph to record which directives are common to different Capabilities, and maintain traces for them. This is in agreement with the observation that a change in a detailed requirement is often less significant, and hence, it is more cost efficient to maintain traceability for critical entities \[95\]. The criticality of directives can be deduced from their relevance values, which are elicited for the computation of the cohesion measure.

In the transition space, we use the system view to formulate and optimize Capabilities from the FD graph generated in the problem space. We observe that directives behave as linkages between the two spaces, and also, preserve and transfer problem domain knowledge. These directives are transformed to requirements, which belong to the solution space. We now discuss the solution space, the CE activity of transformation (shown in Figure \[3.1\]) and traceability within the space.

5.3.3 The Solution Space

The solution space is usually understood as the conceptual realm associated with the technical aspects of developing the system. Often, as with the problem space, the terms solution domain and space are used interchangeably. However, we envision the solution space as being defined by the system view, which generates entities called requirements from directives. These requirements are described in the language of the solution domain. We formally describe the solution space as the collective aggregation of the system view, the solution domain, and requirements. In the solution space, activities related to developing the system, such as establishing requirements, modeling architecture, developing design specifications, coding, unit testing, integration and testing, system maintenance and other downstream development processes are performed. We restrict our focus up to the specification of re-
requirements, and therefore, the graphical representation of the solution space in Figure 5.4 presents only Capabilities and requirements as its entities. Capabilities that are present in the solution space, are the unchanged entities that originated in the transition space. Hence, Capabilities behave as connectors between the transition and solution spaces, just as directives do between the problem and transition spaces.

![Solution Space Diagram](image)

Figure 5.4: *Solution Space*

The element, system view, has already been defined in Section 5.3.2. Therefore, we now describe the other elements — solution domain and requirements — of the solution space.

- *Solution Domain*: Solution domain denotes the technical area(s) relevant to the system being developed. This assumes that a solution always implies the development of a system, which is true in software engineering. For example the solution domain provides technical concepts such as design patterns and architectural styles that are relevant to the design of an ATM.

- *Requirements*: A requirement is a statement specified in the language of the solution domain that states what is expected of the system and exhibits specific quality characteristics such as testability, verifiability, accuracy, unambiguity, and others.
The input to the solution space is an optimized set of Capabilities and their associated directives. Moving from the transition to the solution space entails a change in the domain language used to describe the entities. In particular, requirements are now stated in the solution domain language. These requirements are derived from directives. Recall, directives originate in the problem space, assist the change to the transition space and are now transformed to requirements in the solution space. Also, Capabilities identified in the transition space progress unchanged into the solution space, and in the process become associated with a set of requirements as opposed to a set of directives. We now examine the transformation activity and discuss traceability in the solution space.

5.3.3.1 Traceability in the Solution Space

Traceability aspects of the transition space are applicable to the solution space because of the similarity between directives and requirements. Both are low-level detailed entities that describe what is expected of the system. We enumerate the traceability advantages in the solution space obtained by the utilization of the CE process.

1. When transforming directives to requirements, one can use the relevance values to determine the criticality of requirements. Recall that relevance values indicate the importance of a directive in achieving the objective of its parent node. For example, a directive with a relevance value 1 is mission-critical, and therefore, requirements derived from this directive are most likely critical too. This information can be utilized to selectively capture certain traces. In fact, it has been recognized that not all requirements are equal and that it is cost-effective to maintain traces to and from critical requirements [95].

2. Pinheiro [81] describes inter-requirements traceability as capturing the relationships between requirements. The CE process assists this type of traceability in three different ways:
(a) A directive is transformed into one or more requirements. Because the requirements are derived from the same directive, they share a very strong relationship, and therefore, a change in one is most likely to affect the other requirements. Hence, there is an implicit inter-requirements traceability among requirements derived from the same directive source.

(b) In the solution space, Capabilities are associated with a set of requirements that are transformed from directives. Note that Capabilities are unchanged when progressing from the transition to the solution space. By definition, Capabilities exhibit high functional cohesion; every element is essential to attaining its objective. Therefore, requirements associated with each Capability are strongly related to each other because each requirement is working towards fulfilling the same Capability. This facilitates inter-requirements traceability within a Capability and alleviates the overhead of analyzing exponential number of relationships among all possible requirements.

(c) Minimally coupled Capabilities aid in selective traceability between the requirements associated with different Capabilities. Directives that are shared among Capabilities in the transition space result in requirements which are common in the solution space. As a result, traceability efforts can focus on these requirements, which have the potential to affect more than one Capability when changed.

3. Tracing is performed only when a need is perceived. For example, requirements are tagged with keywords, cross-referenced, etc., to facilitate future tracing. However, the same importance has not been extended to the tracing from needs to requirements. We conjecture from the observations above that the process of CE may ease the difficulty of tracing from needs to requirements and thereby, further assist pre-RS traceability.

Thus, in the solution space, requirements are specified for Capabilities that are to be developed. These requirements are obtained from directives by the activity of transformation. The nature of a Capability, and its directives or requirements, facilitate traceability in the
transition space and the solution space, respectively. Moreover, the properties of Capabilities assist in inter-requirements traceability. We now briefly discuss, from a more global perspective, how the different spaces are connected.

### 5.3.4 Connecting Spaces

The preceding sections have described and discussed the traceability aspects in each space. In this section, we adopt a unified perspective and examine the relationship between the spaces. This is essential to understand the potential for traceability from needs to requirements using the CE process. When making a transition from one space to another, either the domain or the view varies. For example, the *view* changes in the progression from the problem space to the transition space; the *domain* changes in the shift from transition space to the solution space. This is presented in Table 5.2 where the italicized entries indicate the changed element. In addition, we observe that the spaces are connected through common entities.

<table>
<thead>
<tr>
<th>Space</th>
<th>Domain</th>
<th>View</th>
<th>Entities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Problem</td>
<td>User</td>
<td>Needs, Directives</td>
</tr>
<tr>
<td>Transition</td>
<td>Problem</td>
<td>System</td>
<td>Capabilities</td>
</tr>
<tr>
<td>Solution</td>
<td><em>Solution</em></td>
<td>System</td>
<td>Requirements</td>
</tr>
</tbody>
</table>

Table 5.2: Connecting the Spaces

Needs are decomposed to directives in the problem space; these directives are utilized to identify Capabilities in the transition space. As mentioned earlier, directives carry with them problem domain information which is preserved in the transition space. Furthermore, they are also used to identify Capabilities, which pass unchanged into the solution space. Thus, Capabilities and directives behave as connectors between the problem and solution spaces. Figure 5.5 graphically illustrates the transitions between and common elements resident in the spaces. The solution approach of the CE process illustrated in Figure 5.5 strives to resolve the problem of the complexity gap represented in Figure 5.1. This is achieved by introducing Capabilities to bridge the chasm between the spaces. In addition,
the conventional overhead and difficulties associated with pre-RS traceability are alleviated with the utilization of the CE process.

5.4 Traceability Challenges

In this section we examine from a pre-RS traceability perspective how CE addresses specific challenges. In particular, we refer to the challenges enumerated in the Grand Challenges document [96] to structure our discussion.

5.4.1 Supporting Evolution

Large-scale systems constantly evolve during their lengthy development cycles. Consequently, there is an enormous overhead in ensuring that the traceability links depict current dependencies. Changes can occur in needs, directives, Capabilities or requirements.
However, we are specifically concerned with the impact of any such change on Capabilities or requirements because they are the formal inputs to the development phases. Therefore, we present how CE facilitates traceability to requirements in the event of different change scenarios.

- **Needs Change:** The FD graph illustrates a decomposition of functions, which essentially represents the user needs. The visual structure of the graph provides information about nodes that can be affected by a need change. For example, in Figure [3.2] if a need associated with node $n_2$ changes, then it is evident that nodes $n_{29}$ and $n_7$ are most likely to be affected because they are connected by decomposition edges to the parent node $n_2$. If any of the affected nodes are Capabilities, then the set of associated requirements are also impacted.

- **Directives Change:** In the event of a directive change, its relevance value can provide an estimate of the possible impact on the associated Capability. Furthermore, this change also affects those requirement(s) obtained from the changed directive. Also, coupling between Capabilities is computed as a function of the coupling between their respective directives. Therefore, low coupling implies that a change in a directive has a decreased likelihood of affecting directives in another Capability, or their derived requirements.

- **Capabilities Change:** The high cohesion and low coupling of a Capability contains the impact of change to within a Capability. In all likelihood, only the set of requirements associated with it are affected. The dependencies between Capabilities can be examined using the FD graph to analyze the paths of change propagation.

- **Requirements Change:** As with directives, requirement changes are contained within a Capability. Reduced coupling between Capabilities ensures that a change in a requirement has low impact on requirements associated with other Capabilities.
Thus, no special effort is required to maintain the traceability information about all links in the pre-RS phase. Instead, the design of the FD graph and the subsequent derivations of requirements from directives, provide a natural means for traceability. In addition, a Capability’s characteristics of high cohesion and low coupling, contain and reduce the impact of change, respectively. This serves to alleviate the burden of capturing all possible traces, which is otherwise difficult in an evolving system.

5.4.2 Link Semantics

The FD graph is constructed in the problem space in accordance to certain rules. In particular, different types of nodes and edges are interpreted in the context of the problem space. As a result, by the virtue of its definition, the FD graph provides both link semantics and indicates the granularity of nodes. This information is essential to understand the underlying traceability relationships.

- **Link Type:** The FD graph is currently concerned only with representing expected system functionality, and not with the depiction of non-functional needs. Therefore, we define the edges of the graph to represent decomposition, intersection or refinement relationships. These links are independent of any particular domain, and therefore, are fundamental in nature.

- **Granularity:** The root of the graph represents the overall system mission. In contrast, the leaves denote the directives, which are low-level detailed characteristics. The internal nodes occupy the middle ground depicting functionalities at different levels of abstraction. We observe that the abstraction level of a node increases with its decreasing distance from the root (distance is the number of edges in the shortest path). Thus, the natural hierarchical structure of the graph indicates the granularity of different entities.
5.4.3 Scalability

Current traceability techniques are more suited for tracing from and among structured documents of the solution space, than within the informality of the problem space. In addition, these techniques also have to scale up to maintaining traceability in large systems. Our solution approach, the CE process, provides inherent traceability information, which can be used to adequately manage the enormous complexity of large-scale systems, or to serve the interests of small-scale system development. In either case, the CE approach facilitates traceability within and between the different spaces. In particular, as discussed in Section 3.1.1, CE facilitates traceability in the problem space, which has been largely neglected hitherto.

5.4.4 Human Factors

Traceability is often viewed as an activity that is extraneous to the actual development process. Human beings regard traceability efforts as invasive and time-consuming because of the inability to generate automatic traces as a by-product of development. In addition, it is difficult to trace between artifacts created by different users. However, we claim that using a Capabilities-based approach reconciles these problems:

- **Integrating Traceability:** The FD graph is the basis for the decomposition and formulation activities of the CE approach. Additionally, its edges behave as links, and thereby serve as a means for tracing between needs, directives and Capabilities. Thus, unlike current techniques in the CE development approach there is little overhead in producing trace information.

- **Bridging Semantic Differences:** As discussed in Section 3.4, the difficulty of tracing artifacts across different spaces is alleviated by maintaining the linkages between spaces.
5.4.5 Cost Benefit Analysis

Complete and comprehensive traceability between every entity produced during the development process is theoretically desirable but practically infeasible. The cost of maintaining all possible traces is not commensurate with the advantages one may obtain. In particular, as discussed in Section 5.3.3.1, to derive maximal benefits of traceability it is more prudent to maintain links between entities that are mission-critical or those that are described at a high-level of abstraction. We know that Capabilities enable one to abstract back from the lower details and focus on larger aspects of complex systems. By the same token, the requirements associated with Capabilities permit the analysis of details pertinent to that function.

We certainly acknowledge that there is an overhead cost associated with CE. We conjecture that this cost is minimal when one considers that the inherent traceability provided by CE reduces the cost associated with upfront traceability effort. Additionally, the cost-savings achieved through the property of change-tolerance, the ease of downstream maintainability, and the facility to support critical traceability, argue that the introduction of Capabilities provides benefits that exceed the costs of traceability.

5.5 Conclusion

Empirical evidence suggests that pre-RS traceability is crucial for the success of RT. This type of traceability is concerned with capturing relationships between requirements and their sources, which are primarily user needs. However, this process is challenged by the vast disparity between the informality of a user need and the rigidity of a system requirement. We claim there exists an intermediary space called the transition space, which can structure the movement from one space to the other. More specifically, we identify highly cohesive, minimally coupled, optimized functional abstractions termed Capabilities in this space. Here the Capabilities are associated with a set of directives, which are obtained by the decomposition
of user needs. Hence, directives preserve and carry domain information from the problem to the transition space. However, it is the Capabilities that progress unchanged into the solution space, where their directives are transformed to requirements. Therefore, although, the spaces are dissimilar they are connected through common entities. By establishing such relationships, we are no longer constrained by the traditional techniques of traceability. The use of directives and Capabilities generates an inherent traceability mechanism from needs to requirements. Furthermore, the structured nature of this approach supports the development of automated tools to capture and analyze different trace paths. Thus, by the virtue of using a Capabilities-based development approach, the effort to directly relate informal needs and formal system requirements is reduced, the complexity gap between the spaces is bridged, and the ability to trace requirements back to their needs and vice-versa, i.e. pre-RS traceability, is provided.
Chapter 6

Conclusion

Our research strives to embody complex emergent systems with the property of change-tolerance by using Capabilities. To compartmentalize the impact of change these entities are designed to be as disjoint as possible yet are inherently connected in their effort to fulfill the expected functionalities of the system. Results of the empirical evaluations of complex emergent systems indicates the use of Capabilities improves the change-tolerance of a system, and promotes change-reduction during system development.

The CE process operates in a space prior to requirements specification, a region considered nebulous hitherto. The systematic construction of the FD graph forces clarity of ambiguous expectations, avoids redundant specifications, encourages the natural process of decomposition and preserves valuable domain knowledge. This graph serves as a highly structured input for the algorithm that utilizes the specific criteria of cohesion, coupling and abstraction level to identify Capabilities. Although, such criteria have been known to exemplify the fundamental principles of designing complex systems, it is Capabilities that exhibit their efficacy in the realm of software engineering.
6.1 Main Contributions

We contribute to the state of the art by evolving a methodological approach to produce change-tolerant software for real-world systems. That approach embodies an algorithmic specification detailing how needs are aggregated to produce balanced abstractions exhibiting low coupling and high cohesion. The CE process currently derives a functionally-oriented architecture. Using non-functional requirements as the basis, a retrospective examination of the Capabilities and the CE process provides additional insights as to how we can introduce additional architectural considerations earlier in the software development process. Finally, recognizing that the loss of domain knowledge is a major problem when traversing the complexity gap between needs and requirements, additional intellectual contributions are realized through our approach to Pre-requirement specification traceability derived from the CE process. The main contributions of our research to the Software Engineering body of knowledge are enumerated below:

6.1.1 Capabilities Engineering

We developed an alternate approach to traditional Requirements Engineering for constructing complex emergent systems, such that it promotes change-tolerance, and change-reduction. This approach, termed Capabilities Engineering, utilizes desirable software engineering characteristics of cohesion, coupling and abstraction levels and is based on the principles of the science of complexity. The CE approach is published in [97].

6.1.2 Capability Identification Algorithm

We developed and evaluated two diametrically polar algorithms to identify the Capabilities of a system. Investigation of these algorithms on a small-scale computer-based library system,
and Course Evaluation System resulted in a composite approach to Capability Definition. The algorithms are published in [42] [98].

6.1.3 Function Decomposition Graph

We designed a graph-based framework to represent the functionality of the system to be built. This graph embeds the users perspective, and consequently, provides an opportunity to integrate usability and software engineering in the incipient stages itself. Furthermore, the graph is constructed by the intuitive process of decomposition and provides a visual representation of the functional hierarchy. An example graph of a real system (Sakai) has been constructed in collaboration with the community [99] [100].

6.1.4 Measures

We formulated specific measures to compute the cohesion and coupling of a functional abstraction, well before the commencement of design. Hitherto, no such measures have been developed prior to the design phase.

6.1.5 Pre-Requirement Specification Traceability

Pre-Requirement Specification traceability is the activity of capturing relations between requirements and their sources, in particular user needs. The process of CE supports the evolution of traces, provides semantic and structural information about dependencies, incorporates human factors, generates traceability relations with negligible overhead, and thereby, fosters pre-Requirement Specification traceability. The work is published in [45].
6.1.6 Requirement classification

We characterize natural language statements in terms of their domain, view and the space they belong to. All statements are not equal and this is evident in their classification into needs, directives, Capabilities and requirements. In addition, this also assists in bridging the complexity gap between problem and solution space.

6.1.7 Optimal Scheduling Sequence

We have constructed an algorithm to generate the sequence of Capabilities - from all possible permutations - that exhibit minimum coupling. This optimal implementation sequence is in accordance to the principle of incremental development.

6.1.8 Empirical Analysis

The change-tolerance and change-reduction properties of Capabilities, and the CE process have been empirically validated using real world complex emergent systems - Course Evaluation and Sakai systems, respectively. The change-tolerant aspect of CE has been published in [101].

6.1.9 Model Driven Architecture

We explored the role of Capabilities in Model based development methods for reconfigurable systems [102].
6.2 Future Work

Building on the existing CE process, future work may seek to answer the pertinent follow-on questions:

1. What alternative/additional mathematical characterization(s) of Capabilities can be employed to better optimize the identification of Capabilities and to produce a more automated approach for accommodating the confounding impact of schedule and technology constraints, and

2. How do we integrate Capabilities with an existing development paradigm like Object-Oriented so that the resultant system continue to exhibit change-tolerant and other characteristics required of real-world systems.

6.2.1 Alternative / Additional Mathematical Characterization(s) of Capabilities

Currently, the Capability Engineering process has several computationally intensive aspects, e.g., the construction of the FD graph and determining Capability sets. Additionally, we expect that alternate measures of cohesion, coupling and abstraction level for identifying Capabilities exist and which can help reduce computational demands and further strengthen construct validity using those measures. We discuss these areas in the following sections.

6.2.1.1 Function Decomposition Graph

The FD graph is a core component in the computation of the Capabilities of a complex emergent system. The graph is created by an intuitive process of decomposition, beginning with the root node representing highest level mission of the system. The mission node is then partitioned into constituent sub-functions; a recursive decomposition process continues
on each sub-function until directives are defined. This process of constructing the FD graph is notably subjective, and understandably, for a given problem there may be several different FD graphs. The proposed research outlined below address these and other germane issues.

**Graph Construction:** Differing perceptions of users result in the variability between FD graphs for the same system. To reduce this variance we consider potential solutions that manipulate either the cognitive process or the graph definition. As part of the former category, one needs to homogenize the human-centric process of decomposition or develop predictive abilities to automatically generate the graph based on user preferences. On the other hand, by enforcing more stringent graph-construction rules we can elicit greater objectivity from the user.

Presently, the edges of an FD graph can be of the type decomposition, intersection or refinement. These edges are restricted to representing a parent-child hierarchy. There are, however, several other kinds of edges that can be used to represent temporal information, association relationships, data flow and so forth. We conjecture that such information can facilitate in consistently constructing a more representative FD graph.

**Relevance Values:** We assign relevance values to directives based on the risk impact categories. Specifically, one estimates the impact of the failure to realize a directive and assigns a value accordingly. One may consider alternate sets of relevance values based on different criteria such as usability, security, performance, and other quality characteristics. In other words the integration of non-functional aspects in the CE process provides an opportunity to examine and evolve the construction and assignment of other suitable sets of relevance values.

**Hierarchical Task Analysis:** Coordinating usability practices within a software engineering framework is an area of ongoing research. We conjecture that the FD graph provides a platform to interweave aspects from both disciplines. More specifically, the FD graph en-
compasses a user view of the system, and is defined in the problem space. Its close alignment with the stakeholders perception and description of expected functionality utilizes decomposition principles as employed by such as Hierarchical Task Analysis (HTA) [103]. HTA uses a top-down decomposition of goals to understand the tasks necessary to achieve a particular functionality. A comparison of task analysis techniques in HCI and the FD graph can provide insights about accommodating several issues within the Capability Engineering process. For example, an important issue is the level of decomposition. In an FD graph we stop at the directive level, which implies that any further decomposition only results in low-level details. This is similar to HTA which stops when all information needed for analysis is obtained. Furthermore, certain rules specify that if the product of probability of failure and the cost of failure is deemed acceptable, then the decomposition can cease. However, this is more of a conjecture — additional research is required to identify stringent stop rules.

6.2.1.2 Metrics

Capabilities are identified on the basis of their metric values of cohesion, coupling and abstraction level. These measures are computed using structural information provided by the FD graph. We discuss below several issues related to the construction, evaluation and applicability of these metrics.

**Goodness:** Software metrics are theoretically flawed when they fail to validate against the principles of measurement theory [104] [51]. Several axiomatic approaches have been proposed to evaluate both the cohesion and coupling measure [56] [53] [105]. Additionally, verification of a metric can help theoretically ascertain the construct validity [106] of the metric. A process model for software measurement methods has also been proposed by Jacquet and Abran [107] that can help determine if the values of these measures influence the change-tolerance of a given system. Thus, future work can entail the theoretical verification of these measures.
Alternate Metrics: Presently, the cohesion and coupling measures are graph-based computations that use only the structural information. However, we can construct alternate metrics that use information such as the semantics of a node or the history of system development.

- Presently, the cohesion metric is an weighted average of the relevance values of directives, and the size of functional abstractions. Relevance values are elicited for each directive based on its criticality to the parent function in question, and are assigned one of four values. However, instead of relying on users’ input, these values can be determined by analyzing needs/requirements of other systems developed in that domain. Doing so provides a facility to automate the assignment of these values - a considerable savings on time and effort.

- Coupling, measures the likelihood that a change in one entity will affect the other. In our construct, we use a simple probability model of change. On the other hand, one can employ a more sophisticated model that examines the history of change to determine this probability. Note that it is this component of the coupling measure that defines the asymmetric nature of coupling - a factor that allows us to examine alternate configurations of Capabilities for various scheduling sequences. Thus, the probability model can also incorporate information relevant to schedule constraints, or other external factors when estimating the dependency between entities.

- Abstraction level is key in the development and evolution of complex emergent systems, and so, we strive to identify nodes at balanced levels of abstraction as Capabilities. Presently, an ideal abstraction level is determined by visually examining the nodes and performing a trade-off analysis between decreased size and increased coupling. However, we foresee a more objective measure, similar to that of cohesion and coupling, to identify the optimal level. In addition, a formalization of the analysis between conflicting factors such as size, coupling, cohesion may additionally help improve the decision making process.
Metrics play a pivotal role in the CE process. The cohesion, coupling and abstraction values are the Capability identification criteria. By devising them to use all possible relevant information, we can ensure that the set of generated Capabilities are more change-tolerant. At the same time, one must assert the validity of the measures constructed, both from a theoretical and empirical standpoint.

6.2.1.3 Multi-disciplinary Optimization Approach

To determine Capabilities, all nodes of the FD graph are examined for their cohesion, coupling and abstraction level values. For a large-scale complex system, this requires the analysis of at least several hundred nodes. Consequently, assessing all possible permutation sets of Capabilities is an NP-hard problem. Also, to improve the change-tolerance of the system, we subject Capability sets to constraints of schedule and technology. Other external factors can also be introduced to incorporate more comprehensive scenarios of real-world development. To address the computational complexity and generality of the optimization model, we explore the following research avenues:

Computational Complexity: The asymmetric nature of coupling implies that permutations of a Capability set have different coupling values but exhibit the same cohesion, and abstraction level. This requires us to first generate all combination of nodes in the FD graph, and then enumerate the permutations for each combination. Thus, there are two aspects of exponential complexity here:

- Combinations: We reduce the search space of all possible combinations of nodes by defining constraints. For example, a directive or a mission cannot be considered as a Capability, just as nodes that share common directives. Such restrictions narrow the sets of nodes that need to be examined.

- Permutations: As the number of Capabilities in a combination increases, generating
all possible permutations becomes challenging. However, if we change the probability model of the coupling computation such that it becomes symmetric then the analysis is restricted only to the combination, and not its permutations. Also, the coupling measure is used in some sense to determine the optimal order of scheduling. We can now subject a combination of Capabilities to external constraints, as opposed to integrating them in the coupling computation, to reduce the complexity of computation.

Thus, by applying certain logical restrictions we can reduce the complexity of computation. However, such constraints should be formulated such that they preserve the definition of a Capability.

**Optimization Model:** Phase I of the CE process determines Capability sets that exhibit properties of high cohesion, low coupling and balanced abstraction level. At this juncture other factors such as effort, cost, schedule, can be examined to determine the most optimal set of Capabilities. We discuss below our interpretation of schedule and technology constraints, and the potential to expand the optimization model to accommodate any general constraint.

- **Schedule:** In the context of the CE process we view schedule as a function of implementation order and time. Order is the sequence in which Capabilities are to be developed. Time is the period within which a Capability of the system is to be delivered. This definition of scheduling Capabilities is reflective of the principle of incremental development, a risk mitigation strategy for large-scale system development [108, 20]. The time period within which a Capability is to be delivered is dependent on technology (among several other factors).

- **Technology Advancement:** There are two possible scenarios when incorporating technology in a system- obsolescence and infusion. The former involves replacing obsolete technology with new technology and the latter introduces new technology in the system, as a result of building new Capabilities.
Technology infusion is the introduction of the latest technology into a system. More specifically, it is the implementation of Capabilities such that they are incorporated with the state-of-art technology. However, a particular technology needed to develop a Capability may require additional time to achieve maturity. In such a case, it is judicious to examine if the Capability under discussion can be delayed until the technology is available for use. For example, if the set of Capabilities be \( C_1, C_2, C_3, C_4, \) and \( C_5 \) is awaiting a new technology, then there are different cases that can be examined:

1. Delay the development of \( C_3 \) and all its subsequent Capabilities (here \( C_4 \)) until the technology matures. In this case one has to ensure this delay is feasible for \( C_4 \) and if it is accommodated within the expected schedule of the system.

2. If delaying the development of \( C_4 \) is unfeasible then explore alternate configurations where \( C_3 \) is the final Capability to be developed. For this, we need to re-examine sets that are eliminated by the criteria of low coupling.

3. If it is impossible to postpone the development of \( C_3 \) then one is forced to implement it with the currently available technology. This gives rise to a potential situation where the technology can become obsolete. However, the effects of technology obsolescence are mitigated with the use of a change-tolerant Capability.

**General Model:** We see that the aspects of schedule and technology are closely intertwined, and thus, need to be considered as different dimensions of a single problem, rather than separate individual concerns. This mandates the use of a general model of optimization where the combined effect of several different factors can be analyzed. For example, a cost-effort estimation may have a greater influence in determining the Capability configuration than say, the schedule. Thus, the formulation of such a general model will prove useful in the development of systems of large-magnitude.
**Long-term Capabilities:** The CE process generates change-tolerant Capabilities for long-lived systems. However, these systems are still subject to change even after certain Capabilities are realized. In such a case it is prudent to examine if the generated set of Capabilities is still valid as the system evolves over time. In other words, at different points in time an analysis of the snapshots of the Capability sets can help determine if they are considerably different. This consideration leads to questions: Should Capabilities be constantly refactored or are they fundamental enough to be valid throughout the development lifecycle? Does the introduction of new functionality require the refactoring of existing set of Capabilities? In order to provide definitive answers it is imperative that the kinds of changes, and their time frame be determined.

### 6.2.2 Establishing the Paradigm Independence of Capabilities

Capabilities are functional abstractions that are generic enough to be paradigm-independent. Because their definition is inherently tied to principles of cohesion, and coupling, and are defined prior to requirements specification, we conjecture that Capabilities can be integrated easily into many different development paradigms. As a potential reference point we propose to examine CE relative to the OO paradigm.

The OO paradigm and the CE process share the principles of abstraction and decomposition. OO analysis (OOA) strives to identify entities that are fundamental to the domain under consideration, and yet are independent of the problem being solved. In OO design, such entities, better known as objects, are further refined to facilitate encapsulation and information hiding, modeling real-world properties. Similarly, CE identifies Capabilities that exhibit high cohesion, low coupling and a balanced abstraction level. These characteristics assist in defining a change-tolerant design for the system to be developed. Thus, the OO and the CE process manage complexity using techniques of modularity. However, because the Capabilities occur prior to requirements the input for OOA the CE process can be utilized
and integrated in the OO paradigm of development. In the following sections, we explore consequent issues, and discuss avenues for future research.

6.2.2.1 Fundamental Entities

The OOA model should consist of enduring classes of domain objects that are independent of any particular functionality. In essence, these are entities identified from the problem domain, and have the highest potential for reuse. This informal process is often driven by division of data, behavior, responsibilities, and/or noun/verb identification. In contrast, we propose to use Capabilities to define a structured approach to identify these fundamental entities.

- One such approach could be to construct use cases for each Capability. Then, for a set of Capabilities, compare use cases across the different Capabilities, and identify recurring elements. These elements constitute the fundamental entities.

- Moreover, because a Capability is designed to be minimally coupled and maximally cohesive, they are only related in the sense of striving to accomplish a common mission. Hence, the Capabilities of a system can be envisioned as separate problems by themselves. Thus, comparing different Capabilities is similar to comparing specific problems that occur in the same domain.

Such a rigorous process ensures that the identified entities are indeed fundamental to the domain being examined.

6.2.2.2 Use Case Definitions

Uses cases describe the interaction between the system and the actors. OOA generates use cases after needs analysis. This often includes discussions with stakeholders, interviews, the use of questionnaires, and examining lists of “requirements”. In this context one can explore
to what extent the FD graph of the system can serve as the basis for the construction of use cases. The investigation may be guided by the following observations and assumptions:

- **User View:** An FD graph describes the functionality expected of the system in the problem space and from users’ view. A use case also describes the external behavior of the system from a user’s perspective. Hence, one can explore a natural derivation of use cases from the FD graph

- **Abstraction Level:** Use cases can be developed for each Capability. The FD graph illustrates all other functions that constitute a Capability (as part of its decomposition and intersection). Hence, this graph can be used as a reference framework to understand the level of abstraction at which a particular use case is being defined. In some sense, Capabilities guide the level of descriptive detail for its associated use cases.

- **Validation & Verification:** The FD graph can be used to validate if the use cases are complete and depict all functionalities expected of the system.

- **Change-tolerant Characteristics:** A Capability is defined to exhibit high cohesion, and low coupling. By defining sets of use cases specific to each Capability there is a more structured categorization of these use cases. Moreover, these use cases work towards achieving Capabilities/ functionalities that are inherently change-tolerant. Thus, for the purposes of describing functionality, use cases associated with Capabilities are also change-tolerant in nature.

- **Relevance Values:** These values depict the criticality of a directive to its parent functionality. Such values can assist in annotating or prioritizing use cases with information about their criticality.

- **Graph Structure:** Nodes in an FD graph provide static information, and the edges between them depict dynamic behavior. The issues to explore are (a) whether we preserve the same kinds of edges as in the original graph, or introduce alternate edges that
better describe the graph from a more dynamic perspective, and (b) how the information from the graph structure can be utilized to understand and predict relationships between use cases in OOA, and perhaps, classes in OOD.

6.3 Summary

Capabilities Engineering employs a unique algorithm and set of well-defined metric computations that exploit the principles of decomposition, abstraction and modularity to identify functional aggregates, i.e. Capabilities, which embody the desirable software engineering attributes of high cohesion, low coupling and balanced abstraction levels. Change-tolerance is achieved through the embodiment of such attributes. The integration of the CE process with existing development paradigms, and the exploitation of enhanced traceability that accompanies it, are expected to promote change-reduction during system development.

Our research will lead to the advancement of how to design and build large-scale real-world systems. The resulting development methodology will have a positive impact on software systems requiring long-term development cycles, help minimize the detrimental impact of changes due to technology insertion and schedule constraints, and have a beneficial influence on down-stream maintenance activities.
Bibliography


Appendix A

Sakai Experiment

We validate the change-reduction properties of the CE process using Sakai, a complex emergent system. As explained in Section 4.2, we need to identify the set of Capabilities in order to assess if the use of the CE process generates or inhibits a particular change-request. Specifically, the Capability set helps better understand the categories of “Modularization”, and “Unstructured Process” as described in Table 4.5 and thereby, assist in the classification of a change-request. In addition, the directives associated with each Capability help determine if “Missing Information” is a potential cause for a request. Thus, in the following sections we elaborate on the Capability identification process (Section A.1), and present analysis of the nine samples of change-requests (Section A.2) of the Sakai system.

A.1 Capability Identification

Step 1: Constructing the FD graph

The FD graph of the Sakai system is based on features described in Sakai version 2.2, and is illustrated in Figure A.1. The graph is constructed in accordance to the rules enumerated in Section 3.1. User needs and domain information were collected from documents created
Figure A.1: *FD Graph of Sakai*

during the incipient stages of the project. In addition, the graph has been validated by the *Sakai* community in their bi-annual conference [99] [100].

**Step 2: Computing Cohesion and Coupling Values**

We compute cohesion using the measure described in Section 3.2. For the coupling computation between nodes $p$ and $q$ we modify the measure of probability to reduce the complexity of computation. This is described below:

$$C_p(p, q) = \frac{\sum_{d_i \in D_p} \sum_{d_j \in D_q} C_p(d_i, d_j)}{|D_p| \cdot |D_q|}$$
where \( Cp(d_i, d_j) = \frac{P(d)}{\text{dist}(d_i, d_j)} \) and \( P(d) = \frac{1}{|D|} \).

\( P(d) \) is the probability that a directive will change among all other directives associated with the system; \(|D|\) is the total number of directives in the FD graph.

Note that this probability of change (\( P(d) \)) estimates the likelihood that a directive \( (d) \) will change among all other directives associated with the system, as opposed to among those associated with a particular Capability. By using this probability computation the coupling measure becomes symmetric, \( i.e. \) \( Cp(p, q) = Cp(q, p) \). Consequently, the permutations of a slice exhibit the same coupling value as a basic slice. If we utilize the original coupling computation then we would have to generate all permutations of each slice. Because \texttt{Sakai} is such a large system, the number of nodes in a slice vary from 5 to 35. It is computationally infeasible to compute the permutations for slices with a large number of nodes. In addition, the focus of our experiment is validating the change-reduction properties of Capabilities, and not about the scheduling sequences determined by various permutations. Furthermore, the probability measure used here is still relevant to system development, and hence, our decision to utilize the above described modified coupling measure.

**Step 3: Identifying the Candidate Set of Slices**

A total of 119 slices were generated by the Capability identification algorithm. These slices were ranked by high cohesion and low coupling. Rank 1 represents high cohesion or low coupling. The top 10 slices in either category are enumerated in Table A.1.

Slices whose cohesion is above the average cohesion value (0.7414), and coupling is below the average coupling value (15.1955) of the slices displayed in Table A.1 are \#28, \#69, \#72, and \#77. These are the candidate slices. We now need to examine their abstraction levels to determine the optimal set of Capabilities.

**Step 4: Computing Balanced Abstraction Levels**
Table A.1: Slices Ranked in top 10 (either cohesion or coupling) of *Sakai* system

<table>
<thead>
<tr>
<th>SLICE#</th>
<th>COHESION</th>
<th>$Rank_{Ch}$</th>
<th>COUPLING</th>
<th>$Rank_{Cp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>0.763138</td>
<td>1</td>
<td>16.7122</td>
<td>25</td>
</tr>
<tr>
<td>80</td>
<td>0.761362</td>
<td>2</td>
<td>15.21</td>
<td>12</td>
</tr>
<tr>
<td>32</td>
<td>0.756663</td>
<td>3</td>
<td>15.2272</td>
<td>13</td>
</tr>
<tr>
<td>71</td>
<td>0.75595</td>
<td>4</td>
<td>16.6654</td>
<td>24</td>
</tr>
<tr>
<td>77</td>
<td>0.754251</td>
<td>5</td>
<td>13.877</td>
<td>6</td>
</tr>
<tr>
<td>72</td>
<td>0.75367</td>
<td>6</td>
<td>15.1007</td>
<td>10</td>
</tr>
<tr>
<td>118</td>
<td>0.750758</td>
<td>7</td>
<td>20.9061</td>
<td>86</td>
</tr>
<tr>
<td>78</td>
<td>0.750491</td>
<td>8</td>
<td>17.7316</td>
<td>37</td>
</tr>
<tr>
<td>28</td>
<td>0.749448</td>
<td>9</td>
<td>15.1219</td>
<td>11</td>
</tr>
<tr>
<td>59</td>
<td>0.748891</td>
<td>10</td>
<td>18.9575</td>
<td>55</td>
</tr>
<tr>
<td>114</td>
<td>0.692198</td>
<td>118</td>
<td>11.6842</td>
<td>1</td>
</tr>
<tr>
<td>49</td>
<td>0.720819</td>
<td>76</td>
<td>12.0659</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>0.726875</td>
<td>52</td>
<td>13.365</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>0.731024</td>
<td>42</td>
<td>13.413</td>
<td>4</td>
</tr>
<tr>
<td>69</td>
<td>0.746535</td>
<td>12</td>
<td>13.7457</td>
<td>5</td>
</tr>
<tr>
<td>17</td>
<td>0.713002</td>
<td>101</td>
<td>14.4214</td>
<td>7</td>
</tr>
<tr>
<td>82</td>
<td>0.735322</td>
<td>32</td>
<td>14.4265</td>
<td>8</td>
</tr>
<tr>
<td>51</td>
<td>0.735881</td>
<td>31</td>
<td>14.8885</td>
<td>9</td>
</tr>
</tbody>
</table>
Table A.2: Candidate Slices of Sakai system

<table>
<thead>
<tr>
<th>Slice#</th>
<th>Abstraction Level</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>2.69231</td>
<td>8.65385</td>
</tr>
<tr>
<td>69</td>
<td>2.66667</td>
<td>9.375</td>
</tr>
<tr>
<td>72</td>
<td>2.68</td>
<td>9</td>
</tr>
<tr>
<td>77</td>
<td>2.65217</td>
<td>9.78261</td>
</tr>
</tbody>
</table>

As described in Section 3.3.2.7, balanced abstraction level is determined by analyzing the trade-offs between node size and coupling values. The sizes and the abstraction levels of the candidate slices are presented in Table A.2.

We observe that slice 28 has the lowest average node size, the highest coupling value, and the most number of nodes (26). This corresponds to the notion that there is a relation between the decrease in the average node size, and increase in the number of nodes in a slice.

**Step 5: Selecting the Optimal Set of Capabilities**

We need to determine the optimal set from the candidate slices enumerated in Table A.2. Slice #77 seems most optimal given that its ranking with respect to cohesion and coupling is 5 and 6 respectively, as shown in Table A.1. However, this set also has the highest average node size of 9.7861. In addition, examining its set of nodes, we observe that node #16 - Collaboration (see Figure A.1) has a size of 64. This is extremely large compared to the slice’s average node size of 9.78261. Hence, we do not consider Slice #77 as an optimal set.

The other three slices have similar sets of nodes. Among them Slice #69 has the least coupling value. However, we refrain from determining an optimal set based these values because the strength of these numbers are determined only relative to the other slices; they do not represent an absolute value. For further support, we examine the set in the FD graph. We observe that it consists of a member node #25 - Resource Tool; most children of this node are member nodes of Slices #28 and #72, implying that the node is of a larger size than average. In addition, the cohesion rank of this slice is lower than that of the remaining candidate slices.
Thus, we are to choose an optimal set from Slices #28 and #72. They are hardly different with respect to the member node composition, and neither consists of a singularly large-sized node as with the previous candidate slices. At this point, we revert back to the rankings to determine if one is better than the other. We observe that Slice #72 is better than Slice #28 with respect to both the cohesion and coupling rankings. Thus, we choose Slice #72 as the optimal set of Capabilities.

We present the above selection of an optimal set with the caveat that there are only minor differences between the four candidate slices. Our experiment merely requires a Capability set for the purposes of modularization, and reinforcing the structured nature of the CE process. So we choose the optimal set on the basis of size, cohesion, and coupling values. However, one may use several additional constraints such as time, schedule, effort to determine a more optimal set of Capabilities.

### A.2 Change-Request Analysis

We now present the classification of each change-request (based on the category of causes) examined as part of the experiment on Sakai to validate the change-reduction properties of the CE process. Samples of change-requests are determined by their creation date. We examine requests spanning two sequential months (if a month has no requests, we consider the next consecutive month as part of the original sample). In total, we analyze 9 samples of varying sizes as illustrated in Table A.3.

As described in Section 4.2.2.1, each request is examined in terms of the possible causes for its generation. The major categories of causes are Missing Information, Modularization and Unstructured Process as shown in Table 4.5. In addition, a request can be a bug, or a feature enhancement. However, we do not consider these cases as the objective is to validate change-reduction properties of the RE and CE processes. We present our analysis of each change-request in a tabular format, where each sample is enumerated in a table. If the
Table A.3: Sample Sizes

<table>
<thead>
<tr>
<th>Sample</th>
<th>Size</th>
<th>Time Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>Jan, 05 - Mar, 05 (No requests created in Feb.)</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Apr, 05 - May, 05</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>Jun, 05 - Jul, 05</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
<td>Aug, 05 - Sep, 05</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>Oct, 05 - Nov, 05</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>Dec, 05 - Jan, 06</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>Feb, 06 - Mar, 06</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
<td>Apr, 06 - May, 06</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>Jun, 06 - Jul, 06</td>
</tr>
</tbody>
</table>

RE/CE process is responsible for generating a particular request, we denote a ‘1’ under the RE/CE process column respectively. A ‘0’ under the same indicates that the process would not have generated such a request. In addition, we also list the cause responsible for the creation of such a request. Tables A.4, A.5, A.6, A.7, A.8, A.9, A.10, A.11, and A.12 represent Samples 1, 2, 3, 4, 5, 6, 7, 8, and 9 respectively.

Table A.4: Sample 1

<table>
<thead>
<tr>
<th>Id</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-144</td>
<td>Currently irrelevant calendar permissions</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-185</td>
<td>NPE on login when user has no password</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-544</td>
<td>Schedule printing to PDF mangles international characters</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-686</td>
<td>Error handling for News Tool during setup</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table A.5: *Sample 2*

<table>
<thead>
<tr>
<th>Id</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-705</td>
<td>Update “Sakai Menubar” help</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-725</td>
<td>Enhance help: Presentation tool</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-808</td>
<td>Context-sensitive help</td>
<td>Modularization &amp; Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-945</td>
<td>News page gets internal scrollbars</td>
<td>Technology Request</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-958</td>
<td>Chat tool does not show recent messages</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-988</td>
<td>Presentation tool interface not consistent</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1051</td>
<td>Customized page order gets reset</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1944</td>
<td>Copy/Move Pool:top-level to match spec</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table A.6: *Sample 3*

<table>
<thead>
<tr>
<th>Id</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-1128</td>
<td>Field should appear if custom type is selected</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1334</td>
<td>Various Issues</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1428</td>
<td>Problems uploading/editing Resources</td>
<td>Modularization</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-1514</td>
<td>Email notifications for announcements fail</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1531</td>
<td>Make it possible to add an existing item to resources</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1554</td>
<td>Archive does not save gradebook</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table A.7: Sample 4

<table>
<thead>
<tr>
<th>ID</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-1620</td>
<td>Presentation tool flaky</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1660</td>
<td>Expand content collections</td>
<td>Technical Request</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1662</td>
<td>Help format inconsistent</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1680</td>
<td>Extra line break added in base64 encoded value</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1716</td>
<td>Removing Components from Sakai can cause unstable behavior</td>
<td>Modularization, Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1723</td>
<td>Out of memory error when accessing presentation tool</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1734</td>
<td>WebDAV client on Windows XP requires multiple logins</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1739</td>
<td>Ability to have more than one tool on a page</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-1760</td>
<td>Modularization (Related to Editors)</td>
<td>bug</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-2007</td>
<td>Schedule / import from outlook</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2045</td>
<td>Presentation area does not recognize Presentations folder</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2049</td>
<td>Next Slide link still shows on last slide in Presentation</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2097</td>
<td>Design spec does not match implementation for News tool</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Id</td>
<td>Change Request</td>
<td>Cause</td>
<td>RE</td>
<td>CE</td>
</tr>
<tr>
<td>------</td>
<td>-------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>SAK-2328</td>
<td>Replacing Resource does not update URL</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2516</td>
<td>Lining out the columns in Grading table</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2568</td>
<td>Presentation Tool description Needed</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2626</td>
<td>Remove IU-specific link</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2654</td>
<td>New event default date</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2730</td>
<td>Accessibility–Tools Accesskey doesn’t work consistently</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2761</td>
<td>Presentation Tool Help Content Incomplete</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2789</td>
<td>Revise and Delete permissions do not work as expected</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2878</td>
<td>To verb or not to verb?</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2887</td>
<td>Presentation tool gives exception when users join an ongoing presentation</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2889</td>
<td>Updates to Chat room do not change for student user</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2916</td>
<td>If a student logs in who has not been assigned to any section, under “View my sections” the message says no sections in this course</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2928</td>
<td>Sites do not display properly</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-2977</td>
<td>Edit Merged Schedule items from MyWorkspace Schedule</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3011</td>
<td>Presentation tool case-sensitive</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3032</td>
<td>In Roster tool, clicking on profile of student and then on homepage results in error</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Id</td>
<td>Change Request</td>
<td>Cause</td>
<td>RE</td>
<td>CE</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------</td>
<td>----------------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>SAK-3051</td>
<td>Duplicate Alert if invalid RSS field is entered</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3059</td>
<td>Score entry for unsubmitted assessment</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3062</td>
<td>Gradebook headers don’t match up with scores</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3151</td>
<td>My Profile - change wording of pull-down menu shown in search results</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3148</td>
<td>Samigo should take advantage of Sakai file upload mechanism</td>
<td>Missing Information</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-3172</td>
<td>Alert Message for too long of a Title needs fixing</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3297</td>
<td>Official Photo (University Id Picture) is not shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3420</td>
<td>Inconsistency with button types in Membership</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3517</td>
<td>Message about updating participants shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3552</td>
<td>Attaching html documents from Resources breaks internal relative URLs</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3568</td>
<td>Roster allows unauthorized views of ID photos</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3581</td>
<td>Email rejection message does not indicate which site the mail was not delivered to</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3590</td>
<td>Page navigation buttons unnecessarily shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3673</td>
<td>Resource Changed Email Notification Tweaks</td>
<td>enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3677</td>
<td>Dates are incorrectly formatted on any locale</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Table A.10: *Sample 7*

<table>
<thead>
<tr>
<th>ID</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-3703</td>
<td>Option show “Assignment List by Student” lists others besides students in spreadsheet</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3723</td>
<td>Multiple WebDAV protocol compliance issues</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3804</td>
<td>Long URLs do not wrap, creating horizontal scrollbar in Resources</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3807</td>
<td>Revising announcement changes list sort order</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3823</td>
<td>Calendar templates assumes date formats</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3825</td>
<td>Assumptions about date format</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3861</td>
<td>Profile</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3880</td>
<td>Resources display incorrect timestamp</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3913</td>
<td>When two entries with the same title are created concurrently, one entry is discarded</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3917</td>
<td>Roster / Size limitation on profile</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3825</td>
<td>Printable calendar does not show full time range for date</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3959</td>
<td>Finish button still enabled when there are no attachments</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4069</td>
<td>FCK Editor - spell check marked in FF</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4080</td>
<td>Import Materials from a File reveals an error</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4112</td>
<td>Translation breaks permissions</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4129</td>
<td>Samigo: published assessment disappear</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4244</td>
<td>Public view options should be consistent between tools</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3151</td>
<td>My Profile - change wording of pull-down menu shown in search results</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3148</td>
<td>Samigo should take advantage of Sakai file upload mechanism</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3172</td>
<td>Alert Message for too long of a Title needs fixing</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3297</td>
<td>Official Photo (University Id Picture) is not shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3420</td>
<td>Inconsistency with button types in Membership</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3517</td>
<td>Message about updating participants shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3552</td>
<td>Attaching html documents from Resources breaks internal relative URLs</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3568</td>
<td>Roster allows unauthorized views of ID photos</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3581</td>
<td>Email rejection message does not indicate which site the mail was not delivered to</td>
<td>Enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3590</td>
<td>Page navigation buttons unnecessarily shown</td>
<td>bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3673</td>
<td>Resource Changed Email Notification Tweaks</td>
<td>enhancement</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-3677</td>
<td>Dates are incorrectly formatted on any locale</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Table A.11: Sample 8

<table>
<thead>
<tr>
<th>Id</th>
<th>Change Request</th>
<th>Cause</th>
<th>RE</th>
<th>CE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAK-4132</td>
<td>Confusing location of Site Resources when attaching to Assignments</td>
<td>Modularization</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-4339</td>
<td>Gradebook, Add Assignment takes invalid date inputs</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4344</td>
<td>Preview of questions in qpools is inadequate</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4347</td>
<td>Removing attachment does not delete it from storage</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4407</td>
<td>References to Calendar in Schedule should be to Schedule</td>
<td>Unstructured Process</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4428</td>
<td>Question creation state remembered after canceling</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4441</td>
<td>Session timeout causes wiki data loss</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4508</td>
<td>Inconsistency in ‘Add Attachment’ Layout</td>
<td>Modularization</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-4511</td>
<td>Merged announcements don’t show in “Recent Announcements”</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4534</td>
<td>Conflicting messages when adding students</td>
<td>Modularization</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SAK-4556</td>
<td>The Publish confirmation dialogs are unformatted</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4580</td>
<td>Problem in Navigating a series of sub folders</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-4977</td>
<td>Chat room does not update</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5046</td>
<td>Cannot tab to “Display to public” button on Add Announcements Page</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ID</td>
<td>Change Request</td>
<td>Cause</td>
<td>RE</td>
<td>CE</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>SAK-5143</td>
<td>Copyright form keeps coming up when opening a copyrighted resource</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5146</td>
<td>Saving a topic as draft in a forum with draft status changes the forum to available</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5151</td>
<td>Creating a new user ‘admin’ is in the new user name field by default</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5162</td>
<td>Bad RSS feed address creates silent error during site edit</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5204</td>
<td>On Resources tool, every user operation causes page to redraw from top</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5216</td>
<td>Chinese incompliance in Syllabus</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5246</td>
<td>Resources from sites are not showing up in public view when they should</td>
<td>Modularization</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5248</td>
<td>Adding questions to a Part of an assessment, adds it as question 1</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5258</td>
<td>Email archive allows restricted names</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5322</td>
<td>User can alter URL to get to Add Item area in Resources when does not have permission</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5408</td>
<td>Grading when no questions have been answered</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5421</td>
<td>Profile remains searchable after user is deleted</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5502</td>
<td>The “Update” button on edit question pool page should be disabled until user has made a change</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5527</td>
<td>When students review their submitted assessment, “Feedback” label is seen even when there is no feedback to display</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5528</td>
<td>Announcement attachment removals saved when the edit canceled</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5574</td>
<td>How should we validate delivery dates?</td>
<td>Missing Information</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5616</td>
<td>Dropbox does not work if Resources Tool is not enabled in a site</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5663</td>
<td>Resources / email sent to all participants when group is designated</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5707</td>
<td>Last Modified Date on Assessments - Core Assessments for listing is not updated</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SAK-5748</td>
<td>End Time shown as one minute less than actual end time</td>
<td>Bug</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>