ABSTRACT

MASSEY, AARON KEITH. Legal Requirements Metrics for Compliance Analysis. (Under the direction of Ana I. Antón.)

Laws and regulations safeguard citizens' security and privacy. The Health Insurance Portability and Accountability Act of 1996 (HIPAA)[1] governs the security and privacy of electronic health records (EHR) systems. The U.S. Department of Health and Human Services (HHS), which is charged with creating, maintaining, and enforcing regulations pursuant to HIPAA, has required systematic changes in institutional privacy practices as a result of nearly 15,000 resolved HIPAA investigations.[2] HIPAA violations can result in serious monetary penalties. HHS recently fined one healthcare provider $4.3 million dollars for violations of the HIPAA Privacy Rule.[3] Ensuring EHR systems are legally compliant is challenging for software engineers because the laws and regulations governing EHR systems are written by lawyers with little to no understanding of the underlying technology.

This dissertation examines how software engineers can evaluate software requirements for compliance with laws and regulations. The main objective of this work is to help software engineers perform a legal compliance analysis for software intended to be deployed in domains goverened by law by developing empirically validated: (a) techniques for determining which requirements are legally implementation ready (LIR);[4] (b) metrics to estimate which requirements are LIR automatically; and (c) a prototype tool that supports identifying LIR requirements using legal requirements metrics.

My empirical studies suggest that the average graduate-level software engineer is ill-prepared to identify legally compliant software requirements with any confidence and that domain experts are an absolute necessity. When working together as a team graduate-level software engineers make extremely conservative legal implementation readiness decisions. Furthermore, we observe that the legal requirements metrics discussed in this dissertation can be used to improve legal implementation readiness decisions. These findings, along with legal and ethical concerns, make the study of legal compliance in software engineering a critical area for continued research.

[1] `Pub.L.No.104-191,110Stat.1936(1996).`

[2] `http://www.hhs.gov/ocr/privacy/hipaa/enforcement/highlights/index.html`

[3] `http://www.hhs.gov/news/press/2011pres/02/20110222a.html`

[4] Legally implementation ready requirements are requirements that have met or exceeded their obligations under relevant laws and regulations.

© Copyright 2014 by Aaron Keith Massey

All Rights Reserved

Legal Requirements Metrics for Compliance Analysis

by
Aaron Keith Massey

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2014

APPROVED BY:

Laurie Williams

Jon Doyle

Julia Earp

Anne Klinefelter

Ana I. Antón
Chair of Advisory Committee

DEDICATION

To the memory of my Father, David Lee Massey, who never found out that Nintendo didn't rot my brain after all.

BIOGRAPHY

Aaron Massey received a Bachelor's of Computer Engineering from Purdue University in 2003. While at Purdue, Aaron worked with Dr. Carla Brodley on network intrusion detection and information security. After graduating, Aaron worked for Advanced Micro Devices in Austin, Texas. During his time as a software engineer, Aaron developed device drivers for MIPS-based embedded Linux systems.

Aaron enrolled in the computer science program at North Carolina State University to transition back to computer security, an interest that endured from his time at Purdue. Aaron's interests broadened to include digital privacy, computer security, and legal compliance concerns in software engineering under the direction of his PhD advisor, Dr. Annie Antón, as a member of ThePrivacyPlace.org. He completed his both his Master's in Computer Science (2009) and his PhD in Computer Science (2012) at North Carolina State University.

During the course of his career as a graduate student, Aaron worked with Jim Harper, the Director of Information Policy Studies at the Cato Institute; Dr. Eugene Spafford, the Executive Director of the Center for Education and Research in Information Assurance and Security at Purdue University; and Dr. Travis Breaux, previously a Research Associate at the Institute for Defense Analysis and currently an Assistant Professor of Computer Science at Carnegie Mellon University.

Aaron is a 2008 Google Policy Fellowship recipient, the 2008-2009 Walter H. Wilkinson Research Ethics Fellowship recipient, a 2010 KPMG IAPP Summit Scholarship recipient, and a 2011 Outstanding Teaching Assistant at North Carolina State University. At Purdue University, Aaron was a Fessenden-Trott Scholar, a Dean's Engineering Scholar, and a National Merit Scholar. Aaron is a member of the Association for Computing Machinery (ACM), the Institute of Electrical and Electronics Engineers (IEEE), the International Association of Privacy Professionals (IAPP), and the USACM Public Policy Council.

Aaron Massey is originally from Noblesville, Indiana, which is just northeast of Indianapolis. He remains a fan of Indianapolis area sports teams.

ACKNOWLEDGEMENTS

"Big decisions threaten to narrow us, as if we would no longer be, after choosing, the person who could have chosen otherwise. Looking into the future, we see ourselves completely defined by one choice—to be someone's lover, to take such and such a job. We become flat characters in a novel, and die a little."

– James Richardson

Writing a dissertation is undeniably a big decision. It requires sacrifice and hard work. Once completed, a title is attached to one's name for the rest of their life. It threatens to completely define the title-holder. I have been blessed by the support that I've received since making the decision to write this dissertation. Some have labored mightily to educate my fantastically stubborn mind; others have repeatedly assured me that my own labors are not in any way narrowing. I can't possibly acknowledge them all here, and I'm sure I'll be mortified to learn after the publication of this document that I've left some rather important people off this list. My apologies in advance. Still, I must at least try to mention those who have supported me during this process. My heartfelt thanks to:

- My advisor: Annie Antón, who convinved me to ride this roller coaster and sat next to me in the front seat. I'm looking forward to the next ride in this amusement park.
- My Dissiertation Committee: Anne Klinefelter, Julie Earp, Jon Doyle, and Laurie Williams, for your expert guidance throughout my graduate career.
- Travis Breaux, for his excellent guidance, rational skepticism, and outstanding example.
- Paul Otto, for sharing his incredible expressive abilities and his friendship. You've made me a better author and a better collaborator.
- Jessica Schmidt, for inspiring me with her focused, clear vision. And for commisurating with me about the trials and tribulations of the Indianapolis Colts.
- Jeremy Maxwell, for reminding me of the priorities my father had.
- Andy Meneely, the first person I met on campus as an official NCSU grad student. Thanks for all the great conversations.
- Ben Smith, for never compromising and always being willing to indulge in discussion for the sake of argument.
- Mei Nagappan, for inspiring me with his constant ability to outperform his brother in every way.

- Pat Morrison, for being willing to share the experience of big decisions carried to fruition and for continuing to make big decisions courageously.
- Jim Harper, for teaching me more about Washington D.C. than I thought possible to learn in a single summer.
- Eugene Spafford, for demonstrating that an understanding of computer science and policy really can change the world.
- Peter Swire, for embracing technologists and patiently explaining the intricacies of the law to them.
- Chris Hoofnagle and Dan Solove, for tirelessly organizing the Privacy Law Scholar's Conference, the best event I've attended as a student.
- Max Davies and Wayne Weise, for being more than just my Odyssey of the Mind coaches.
- Matt Schoolfield, for pushing me to grow outside of a classroom.
- Logan Tetrick, for consistently having a unique outlook on life. You've taught me more than I could have ever learned in any school.
- Nancy Massey, my mom, for never complaining about all the field trips and extra-curricular activities I put you through. Thanks for understanding all those times I broke the family computer. I love you.
- Group Stupid, for never being flat characters in a novel.
- Allison Massey, my wife, for suffering through the ugly parts of my attempts to make something worth making. I wish our first year of marriage could have lived up to the spirit of Deuteronomy 24:5. I hope to spend the next 70 years making that up to you.

Finally, I would like to thank one special teacher who had the vision to start a Computer Club for students at Conner Elementary School way back in 1986. Mrs. Linville, thank you so much for taking time to introduce me to my first computer, the Apple IIe. You probably thought it was half pointless since all we ever wanted to do was play Dig Dug or Number Munchers, but I still remember writing that first mathematics flash-card program. I remember going to the Indiana State Media Fair, winning a first place ribbon, and, most importantly, seeing all the impressive things that people were doing with computers. I hope whatever children I may eventually have are lucky enough to have teachers like you.

Soli Deo gloria

TABLE OF CONTENTS

List of Tables . . . . . **viii**

List of Figures . . . . . **ix**

Chapter 1 Introduction **1**
1.1 A Historical Perspective 2
1.2 Laws and Regulations 4
1.3 Ethics 5
1.4 Legal Compliance, Ethics, and Software Engineering 7
1.5 Overview of Remaining Chapters 8

Chapter 2 Background and Related Work **9**
2.1 Terminology 9
2.2 Legal Requirements 10
2.3 Legal Theory 12
2.4 Requirements Triage and Prioritization 13
2.5 Measurement and Requirements Metrics 15
2.6 Chapter Summary 16

Chapter 3 Tracing Existing Requirements to Legislation **17**
3.1 iTrust Medical Records System 18
3.2 Methodology Overview 18
3.3 Terminology Mapping 19
3.3.1 Tracing Actor Definitions 22
3.3.2 Tracing Data Object Definitions 23
3.3.3 Tracing Action Definitions 24
3.4 Requirements Identification and Disambiguation 25
3.5 Requirements Elaboration 28
3.5.1 Document Each Requirement's Priority and Origin 28
3.5.2 Record Document Provenance 30
3.6 Tracing Requirements to the Legal Text 30
3.7 Evaluating Software Requirements for Compliance with Legal Texts 32
3.8 iTrust Case Study Results 33
3.8.1 Results From Terminology Mapping 33
3.8.2 Results from Requirements Identification and Disambiguation 36
3.8.3 Results From Requirements Elaboration 37
3.8.4 Results From Tracing Requirements to Legal Texts 39
3.9 Lessons Learned 41
3.10 Chapter Summary 44

Chapter 4 Defining Legal Requirements Metrics **45**
4.1 Legal Requirements Metrics 46
4.1.1 Dependency Metrics 49

4.1.2 Complexity Metrics . . . 49
4.1.3 Maturity Metrics . . . 50
4.2 Legal Requirements Triage Algorithm . . . 52
4.2.1 Calculation of Triage Scores . . . 53
4.2.2 Identifying Implementation-Ready Requirements . . . 53
4.2.3 Legal Requirements Triage Methodology . . . 54
4.2.4 Triage Results . . . 60
4.3 Chapter Summary . . . 61

Chapter 5 Validation Studies . . . 62
5.1 LIR Assessment Case Study . . . 63
5.1.1 Materials . . . 64
5.1.2 Software Engineering Participants . . . 66
5.1.3 Subject Matter Experts . . . 66
5.1.4 Legal Requirements Triage Algorithm . . . 68
5.1.5 Analysis Methodology . . . 68
5.1.6 LIR Assessment Case Study Results . . . 69
5.1.7 LIR Assessment Case Study Threats to Validity . . . 77
5.1.8 Discussion . . . 78
5.2 Replication Case study . . . 79
5.3 Wideband Delphi Case Study . . . 82
5.3.1 Wideband Delphi Case Study Methodology and Materials . . . 83
5.3.2 Wideband Delphi Case Study Participants . . . 85
5.3.3 Wideband Delphi Case Study Results . . . 85
5.3.4 Wideband Delphi Case Study Discussion . . . 91
5.4 Chapter Summary . . . 94

Chapter 6 Conclusion . . . 95
6.1 Threats to Validity . . . 97
6.2 Future Work . . . 99

References . . . 101

Appendix . . . 107
Appendix A Materials for User Study . . . 108
A.1 iTrust Medical Records System: Requirements for Technical Safeguards . 108
A.2 iTrust Requirements . . . 109
A.2.1 Authentication . . . 109
A.2.2 Account Information . . . 110
A.2.3 Account Creation and Deletion . . . 111
A.2.4 Emergency Situations . . . 112
A.2.5 Encryption and Backups . . . 112
A.3 iTrust Glossary . . . 112
A.4 Traceability Matrix . . . 113
A.5 HIPAA § 164.312 Technical Safeguards . . . 115

LIST OF TABLES

Table 3.1 Terminology Mapping 34
Table 3.2 Requirement Prioritizations 37
Table 3.3 Origin Categories 39
Table 3.4 Summary of Issues Identified 40

Table 4.1 Triage Metrics 48
Table 4.2 Triage Results ($D = 30, C = 30, M = 60$) 61

Table 5.1 Model Coefficient Signs 75

Table A.1 User Study Traceability Matrix 114

LIST OF FIGURES

Figure 3.1 Overview of the Traceability Methodology 20
Figure 3.2 Example of Direct and Indirect Tracings from Software Terms to Legal Terms 23
Figure 3.3 iTrust Use Case 4 27
Figure 3.4 iTrust Requirement 19 30
Figure 3.5 iTrust Requirement 21 32
Figure 3.6 iTrust Actor Hierarchy 34
Figure 3.7 Partial HIPAA Stakeholder Hierarchy 35
Figure 3.8 iTrust Use Case 3 38

Figure 4.1 Sample Hierarchical Legal Text 47
Figure 4.2 Sample Requirement 47
Figure 4.3 Red Wolf Requirements Management Screenshot 55
Figure 4.4 Legal Requirements Triage Methodology Overview 56
Figure 4.5 HIPAA §164.312(a)(1) 57
Figure 4.6 Sample XML Markup for HIPAA §164.312(a)(1) 58
Figure 4.7 Sample Case Study Requirement 59

Figure 5.1 Example Legal Compliance Scenario from Case Study Tutorial 67
Figure 5.2 iTrust Requirement 18 70
Figure 5.3 iTrust Requirement 26 71
Figure 5.4 Student Responses to the Legal Implementation Readiness Studies of Spring 2011 and Fall 2011 76
Figure 5.5 Assessment of Requirements by Participants Prior to Wideband Delphi Method 86
Figure 5.6 iTrust Requirement 11 88
Figure 5.7 Duration of Discussion for Requirements in the Wideband Delphi Case Study 90
Figure 5.8 HIPAA § 164.312(a)(1) 92
Figure 5.9 iTrust Requirement 27 93

Chapter 1

Introduction

"All that may come to my knowledge in the exercise of my profession or in daily commerce with men, which ought not to be spread abroad, I will keep secret and will never reveal."

–The Hippocratic Oath

This dissertation examines how software engineers evaluate software requirements for compliance with laws and regulations and proposes legal requirements metrics that improve legal implementation readiness decision making. The main objective of this work is to help software engineers ensure that software complies with laws and regulations by developing empirically validated: (a) techniques for determining which requirements are legally implementation ready (LIR);[1] (b) metrics to estimate which requirements are LIR automatically; and (c) a prototype tool supporting the identification of LIR requirements using legal requirements metrics.

To our knowledge, this work is the first to empirically examine the extent to which software engineers are able to accurately determine whether software requirements meet or exceed their legal obligations. *The goal of this research is to provide actionable insight on legal compliance decisions in the software requirements engineering process through the development of legal requirements metrics.* We focus our research in support of this objective on the following research questions:

RQ1 Can software engineers evaluate existing software requirements for legal compliance?

RQ2 Can software engineers generate traceability links from existing software requirements to the appropriate subsections of the legal text?

[1] Legally implementation ready requirements are requirements that have met or exceeded their obligations under relevant laws and regulations.

RQ3 Are simple metrics based on the structure of a legal text related to the legal implementation readiness of software requirements?

RQ4 Can simple metrics based on the structure of a legal text be used to estimate legal implementation readiness of software requirements?

We examine each of these questions in a series of studies. For RQ1 and RQ2, we develop and validate a method for evaluating existing software requirements for legal compliance. This method uses the Inquiry Cycle [63], ambiguity classification [5], and relationship identification [5] and takes as an input a legal text and a set of requirements that must comply with that legal text. It produces as an output a set of documented legal compliance concerns, and a traceability mapping of requirements to specific subsections within the legal text, which is used as an input for the research on RQ3 and RQ4.

To examine RQ3 and RQ4, we develop legal requirements metrics, which comprise simple, consistent attributes of a legal text and a set of requirements that must comply with that legal text, such as the number of sections or subsections within a particular regulation. Other attributes include the number of words, the number of cross-references to other sections of legislation, and the number of exceptions within a subsection of a legal text. These metrics are validated in three case studies using the Goal / Question / Metric paradigm [10]. We show that statistical models based on legal requirements metrics performed quite well at determining whether a requirement is LIR. As a part of our examination of RQ3 and RQ4, we developed Red Wolf, a prototype legal requirements management tool, to support using legal requirements metrics as a part of the software engineering process.

1.1 A Historical Perspective

Although centuries old, the well-known Hippocratic Oath[2] still influences our cultural understanding of ethics in healthcare. The Hippocratic Oath may be known best for its "do no harm" clause, but the privacy promise quoted above is extremely motivating to information security and privacy professionals, particularly in the field of medicine. However, healthcare is not the only area of society guided by ethical principles. Societies codify some ethical principles into laws and regulations governing numerous areas of life. In fact, every engineering endeavor is subject to laws, regulations, and ethical principles.

Governments enact laws and regulations to safeguard the security and privacy of their citizens. These laws and regulations reflect the ethical concerns of society and have always targeted engineering endeavors to ensure their safety. The Code of Hammurabi, named for the

[2] `http://www.nlm.nih.gov/hmd/greek/greek_oath.html`

sixth Babylonian king, dates back to almost 1800 B.C. and includes the following provisions for engineered structures [62]:

> If a builder build a house for a man and do not make its construction firm, and the house which he has built collapse and cause the death of the owner of the house, that builder shall be put to death.
>
> If it cause the death of the son of the owner of the house, that builder shall be put to death.
>
> If it cause the death of a slave of the owner of the house, he shall give to the owner of the house a slave of equal value.
>
> If it destroys property, he shall restore whatever it destroyed, and because he did not make the house which he built firm and it collapsed, he shall rebuild the house which collapsed from his own property.
>
> If a builder build a house for a man and do not make its construction meet the requirements and a wall fall in, that builder shall strengthen the wall at his own expense.

The Code of Hammurabi is one of the oldest documents known to humankind. It is significant that it began a tradition of legally regulating engineering projects that continues today. Furthermore, the proscribed punishments clearly indicate the seriousness of the law. Science and engineering have evolved greatly in the nearly four thousand years since Hammurabi's Code was first baked into clay tablets. Home owners typically are not concerned about their houses collapsing on top of them. People regularly fly across continents, communicated with friends hundreds of miles away, and perform myriad other tasks that would seem like magic to ancient Babylonians.

Software in particular has been likened to magic. Fred Brooks opens his essay titled "The Whole and the Parts" with the following [19]:

> The modern magic, like the old, has its boastful practitioners: "I can write programs that control air traffic, intercept ballistic missiles, reconcile bank accounts, control production lines." To which the answer comes, "So can I, and so can any man, but do they work when you do write them?"

Societal expectations increase with engineering progress. As Henry Petroski says [62], "With each success comes the question from society, from taxpayers, from government, from boards of directors, from engineers themselves as to how much larger, how much lighter, and how much more economically can the next structure be made." Societal concerns cannot be

eliminated through improved engineering; they can only be mitigated. Society's concerns are justified in primarily two ways: (1) Legally and (2) Ethically. Let us now examine each concern briefly as it relates to modern software engineering.

1.2 Laws and Regulations

Regulatory compliance has been the primary driver of information security in industry since 2005 [77]. Ernst and Young's 2006 annual survey of almost 1,300 organizations found that "privacy and data protection" was the key motivating factor for information security practices in 42% of the organizations [76]. In 2007, this figure increased to 58% [77]. By 2010, 81% of executives indicated that "managing privacy and personal data is very important or important to their organization" and 82% said that "achieving compliance with regulations is very important or important to their organization" [78]. Unfortunately, past research has noted several reasons why it is difficult to build software systems that comply with regulations [15, 42, 55, 60]. In particular, complying with privacy and data protection policies is especially challenging [5, 17, 46].

The healthcare industry is an exemplar of the need for engineering techniques that improve regulatory compliance. Perhaps partly to fulfill the ancient privacy and security promises of the Hippocratic Oath in the modern age, the United States passed the Health Insurance Portability and Accountability Act of 1996 (HIPAA).[3] The resulting HIPAA regulations govern patient health information usage by providing detailed security and privacy procedures to support the practical needs of insurance companies, healthcare providers, law enforcement and other organizations that have a bona fide need for access to patient health information. In short, HIPAA regulates organizations employing EHR systems, such as those promised by Presidents Bush and Obama. HIPAA includes serious penalties for non-compliance and raises the importance of solid requirements engineering and software design for legally compliant software. Penalties for non-compliance with HIPAA are severe: for a non-criminal violation of the HIPAA Privacy Rule, violators could be fined up to $25,000 per year per violation.

The U.S. Department of Health and Human Services (HHS) writes and enforces the final HIPAA regulations. Although passed by Congress in 1996, HHS issued the final security rule on February 20, 2003. This rule took effect on April 21, 2003 with a compliance date of April 21, 2005 for large covered entities and April 21, 2006 for small covered entities. Thus, software engineers had only two years to ensure their systems were HIPAA compliant.

In his 2004 State of the Union address, President George W. Bush set the goal of providing every American citizen the ability to have Electronic Health Records (EHRs) by 2014 [22]. In

[3] Pub. L. No. 104-191, 110 Stat. 1936 (1996)

February 2009, President Barack H. Obama signed into law his first major legislative act as President: the American Recovery and Reinvestment Act (ARRA)[4], which reserves $17 billion exclusively for the development of EHR systems [68]. President Obama's goal with ARRA is the same as President Bush's stated goal: to provide EHRs to Americans by 2014. Such high-level bi-partisan agreement marks the development of EHRs as a clear national priority for the United States.

Herein, we focus on the legal compliance concerns that arise in the healthcare industry of the United States. We focus on the healthcare industry generally, and HIPAA in particular, for five reasons. First, HIPAA is a non-trivial legal compliance concern. Second, engineers face significant engineering challenges in addition to legal compliance challenges when building and maintaining EHRs. Third, HIPAA compliance is a timely concern. Many organizations are currently facing the challenge of building EHRs and maintaining legal compliance. Fourth, although HIPAA compliance is a current concern, laws and regulations will continue to evolve, and software engineers will continue to need techniques and tools to improve their legal compliance decisions. The recent passage of the ARRA demonstrates this. Fifth, the extent to which laws and regulations across domains are similar is unknown. We cannot make broad conclusions about "the law" because it is possible if not likely that significant differences in the legislative and regulatory process would make broad conclusions impossible. However, HIPAA appears similar to laws and regulations in other domains, and we believe the research presented in this dissertation can be extended to cover those domains.

1.3 Ethics

The Code of Hammurabi was law in ancient Babylon, but the Hippocratic Oath was not law: it was a statement of professional ethics. The tradition of the Hippocratic Oath continues today, and modern medical professionals swear to upload a similar oath that also contains a privacy promise:[5]

> I will respect the privacy of my patients, for their problems are not disclosed to me that the world may know. Most especially must I tread with care in matters of life and death. If it is given me to save a life, all thanks. But it may also be within my power to take a life; this awesome responsibility must be faced with great humbleness and awareness of my own frailty. Above all, I must not play at God.

Ethics is the philosophical field encompassing the study of moral values—the study of right and wrong behavior. The medical profession is not the only profession that honors a code of

[4]Pub. L. No. 111-5, 123 Stat. 115 (2009)

[5]`http://www.pbs.org/wgbh/nova/body/hippocratic-oath-today.html`

ethics. As engineers, we must seek to understand how ethics applies to the engineering Members of the National Society of Professional Engineers[6] must "conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession." process. The IEEE Code of Ethics[7] asks engineers "to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment" and "to improve the understanding of technology, its appropriate application, and potential consequences."

The Association for Computing Machinery (ACM), the world's oldest scientific and educational computing society, has a longer, more detailed Code of Ethics.[8] It contains provisions to "Respect the privacy of others" and to "Honor confidentiality," both of which mirror the intent behind similar clauses from the Hippocratic Oath. The ACM Code of Ethics, however, goes further with respect to legal concerns:

> **2.3 Know and respect existing laws pertaining to professional work.**
>
> ACM members must obey existing local, state,province, national, and international laws unless there is a compelling ethical basis not to do so. Policies and procedures of the organizations in which one participates must also be obeyed. But compliance must be balanced with the recognition that sometimes existing laws and rules may be immoral or inappropriate and, therefore, must be challenged. Violation of a law or regulation may be ethical when that law or rule has inadequate moral basis or when it conflicts with another law judged to be more important. If one decides to violate a law or rule because it is viewed as unethical, or for any other reason, one must fully accept responsibility for one's actions and for the consequences.

This section contains a critical ethical statement: Software engineers must not only know and respect the law, but they must also know when ethical obligations require violation of the law. In addition to potentially immoral or inappropriate laws and regulations, laws may simply be unintelligible or impossible to fulfill technologically. Laws and regulations may be written by lawyers who are not be well-versed in technology issues. Engineers have an ethical responsibility to identify these and other situations where compliance with the law is immoral, inappropriate, or even impossible. Thus, requirements engineers, must include relevant laws and regulations in their engineering process.

It is our view that any particular technological artifact exists in an inherently value-free state. For example, consider a hammer—if used to build a home, then the hammer serves an ethically valuable, or "good," purpose. If used as a weapon to attack an innocent person,

[6] `http://www.nspe.org/Ethics/CodeofEthics/index.html`
[7] `http://www.ieee.org/about/corporate/governance/p7-8.html`
[8] `http://www.acm.org/about/code-of-ethics`

then the hammer serves an ethically questionable, or "bad," purpose. Some scholars would disagree with the statement that technological artifacts exist in an inherently value-free state. Such scholars claim that ethical value is imparted to the artifact during the design process.

Regardless of whether one believes a technological artifact is ethically value-free or not, the decisions made during the design process still have some ethical value. For example, consider an engineer attempting to make a hammer. If the engineering knowingly chooses to use a cheap, easily broken material for the handle, then the hammer may be unsafe to use for any purpose, whether building a house or attacking an innocent person. The process of engineering such an artifact must be deemed ethically questionable. Accepting that technological artifacts are value-free does not absolve ethical responsibility for decisions made during the engineering process. Thus, engineers must understand the ethics of their engineering decisions regardless of their view of the final product (i.e. Engineers cannot "pass the buck" with regard to the ethics of the products they build.).

1.4 Legal Compliance, Ethics, and Software Engineering

To ensure software systems are designed ethically and in compliance with law, requirements engineers must specify legally compliant system requirements that satisfy professional ethics concerns. Specifying legally compliant requirements is challenging because legal texts are complex and ambiguous by nature. Requirements engineers responsible for building regulated systems must:

- Understand subtle and difficult-to-define concepts, such as privacy, which are often found in laws and regulations.
- Reason effectively about the relationship between software requirements and specific subsections of the laws and regulations to which they apply to ensure proper coverage of the software requirements.
- Specify requirements with enough clarity to support implementation as well as ensure due diligence with respect to the law. Due diligence refers to the reasonable efforts that engineers make to satisfy legal requirements.

Software engineers need techniques, processes, and tools that help them uphold their legal and ethical responsibilities. This dissertation proposes one such process and one such tool. We begin by asking the question: How do software engineers make legal compliance decisions? One common approach to domain-specific decisions in requirements engineering is to hire or consult with domain experts [52]. Lawyers serve as extremely capable domain

experts for legal compliance concerns. However, as we have discussed, ethical considerations prevent software engineers from passing the full weight of their decisions to their lawyers. Software engineers have an ethical responsibility to understand the law and make appropriate professional decisions accordingly.

1.5 Overview of Remaining Chapters

The remainder of this dissertation is organized as follows: Chapter 2 discusses prior research relevant to this work; Chapter 3 describes the research methodology used to evaluate existing requirements for legal compliance and trace them to the pertinent subsections of the legislation to which they must comply; Chapter 4 defines each legal requirement metric evaluated as a part of this dissertation and outlines a basic algorithm for combining them to generate a legal implementation readiness decision; Chapter 5 presents the validation methodologies and results used to evaluate the utility of the legal requirements metrics defined in Chapter 4; and Chapter 6 concludes this dissertation with a discussion of the limitations of legal requirements metrics, including threats to validity, a summary of contributions and potential future work in this area. Important terminology is defined throughout this dissertation with definitions referenced in Section 2.1.

Chapter 2

Background and Related Work

"When you [. . .] start out in history, without knowing that you're starting out in history, very often you don't have a sense of what came before you, how it got there, and you certainly don't know what's going to come after."

– Paula Scher

In this chapter, we examine prior academic research in legal requirements, legal theory, requirements triage and prioritization, and requirements metrics. We begin with a discussion of key terminology used throughout this dissertation.

2.1 Terminology

Terminology is always important in academic discussions but is even more important when the discussion spans multiple domains as in this dissertation. Below we present an overview of those critical definitions to which we will refer throughout this dissertation.

Electronic Health Record (EHR) refers to a digital document or set of documents that store information related to an individual's healthcare.

iTrust Medical Records System refers to an open-source EHR system designed in a classroom setting where it has been in continual development by students at North Carolina State University since 2004.

Legal Compliance refers having a defensible position in a court of law [15].

Legal Requirements Metrics refer to the eight measures developed in Chapter 4 and used to determine which requirements in a set are legally implementation ready.

Legally Implementation Ready refers to the concept of a requirement meeting or exceeding legal obligations as expressed in relevant regulations. This is often abbreviated as LIR.

Requirements Triage is the process of determining which requirements should be implemented given available time and resources [25].

2.2 Legal Requirements

Bringing software into legal compliance is a challenging software engineering concern that has become the subject of increased requirements engineering research in the last five years [60]. The need to resolve ambiguities, to follow cross-references, and to understand a rich set of domain knowledge make legal compliance particularly challenging for requirements engineers [60]. Often, engineers rely on abbreviated summaries of legal texts that attempt to make dense legal terminology more accessible for requirements engineers to achieve legal compliance [60].

Laws and regulations may continue to be amended by administrative agencies or interpreted in judicial proceedings for years after their initial passage [60]. For HIPAA, the U.S. Department of Health and Human Services (HHS) has discretion over the frequency of updates to healthcare information technology regulations. Legal norms suggest that the HIPAA regulations could be revised as often as every year [11]. However, HHS is not the only government body that has the authority to change HIPAA. The U.S. Congress amended HIPAA through the HITECH Act, which includes data breach notification requirements and expands enforcement authority to cover business associates. Furthermore, U.S. court rulings for cases involving HIPAA could impact interpretations of the regulation.

Logical models of legislation support reasoning about and interpretation of the obligations that legislation entails. Some researchers focus on building logical models of regulation directly from actual legal texts [18, 55, 46, 52]. These models do not address requirements triage or prioritization, but requirements generated from these models could serve as inputs to our legal requirements triage technique. Of these approaches, only Maxwell et al. traces requirements directly to elements of the legal text [52].

Researchers are exploring automated processes for generating traceability links for specifically for the purpose of regulatory compliance [12, 24]. Cleland-Huang et al. use a machine learning approach to automatically generate traceability links for regulatory compliance [24]. However, their links trace from a requirement to an information technology responsibility identified in the law rather than to a specific subsection of the legal text itself. Berenbach et al. describe techniques for just in time regulatory traceability [12]. They intentionally trace requirements to higher level segments of the legal text rather than to individual subsections

[12]. To complete the process, a requirements engineer must be involved manually [12]. These approaches may reduce the effort required to trace requirements to specific subsections of a legal text, which is required by the algorithm presented in this dissertation, even though they do not currently produce such a mapping.

Barth et al. employ a first-order temporal logic approach to model HIPAA, the Children's Online Privacy Protection Act (COPPA)[1], and the Gramm–Leach–Bliley Act (GLBA)[2]—they use this model to demonstrate legal compliance [8]. May et al. employ an access control approach to model laws and regulations, also using HIPAA as an example [55]. Massacci et al. use goal-oriented modeling to analyze Italian data protection laws [46]. Maxwell et al. employ production rules, an artificial intelligence technique, to model regulations for requirements engineers and check requirements for compliance [52, 53]. In their work, requirements engineers query a production rules model and receive specific answers to questions regarding a modeled regulation [52]. Maxwell et al. also use the HIPAA regulations to illustrate their approach [52, 53, 54].

Breaux developed an XML markup of legal texts that formally describes the structure of the legal text [15]. Breaux applied this XML markup to HIPAA for the purposes of supporting requirements acquisition [15]. In this dissertation, we extend Breaux's XML markup to denote cross-references, exceptions, and definitions. These are not identifiable by the structure of the legal text and must be manually identified. In addition, we changed the names of XML elements to have semantic meaning related to the existing structure of the legal text. For example, instead of using a generic 'div' tag, we used tags related to the type of the node, such as 'subpart' and 'section.' Kerrigan and Law also developed an XML markup to describe the structure of legal texts. However, their markup was designed to build a logic-based regulation compliance checker [39].

Legal texts often contain exceptions that can indicate one aspect of a legal text takes precedence over another. Exceptions in the law are another challenge for requirements engineers [60]. Breaux et al. developed the FBRAM (Frame-based Requirements Analysis Method)—a methodology to build requirements that achieve needed legal compliance. The FBRAM also yields a comprehensive prioritization hierarchy for an entire legal text [15, 17, 18]. This hierarchy can identify which requirement takes precedence in cases of legal exceptions [17]. As a result, it is more useful for proving that a set of requirements has the same legal precedence as a set of legal regulations than for prioritizing requirements for the purposes of software construction or maintenance. In this dissertation, we employ legal exceptions as a metric for performing requirements triage rather than legal precedence.

[1]Pub. L. No. 105-277, 112 Stat. 2681 (1998)
[2]Pub. L. No. 106-102, 113 Stat. 1338 (1999)

2.3 Legal Theory

Asking a lawyer to explain the law is similar to asking a philosopher to explain the truth: there are many possible interpretations of both given different assumptions and background information. Although laws and regulations are developed with as much precision as possible, they are still imperfect. Legal texts may contain ambiguities or omissions that require interpretation by lawyers and judges. Legal academics recognize several theories of interpretation for legal texts, and the field as a whole is known as statutory interpretation [20, 28, 29, 31]. In this dissertation, we examine both how requirements engineers perform statutory interpretation and how a basic form of statutory interpretation can be estimated for requirements engineering.

Legislative intent is one theory of statutory interpretation wherein an interpreter may seek to resolve an ambiguous or unclear statue by examining the original goals of the legislative or regulatory body that wrote the statue [20, 28, 29, 31]. For example, the interpretor may examine speeches made in support of the bill that was eventually passed into the law under interpretation. The interpretor may also examine amendments proposed, and whether those amendments were accepted or rejected. These legal artifacts provide some insight as to the intent of the legislative body when constructing the law. Although this is a commonly used theory of statutory interpretation, it requires domain expertise outside of the traditional requirements engineering background.

Legislative history is a theory of statutory interpretation wherein an interpreter examines only the artifacts produced as a part of the legislative process that created the statute under examination [20, 28, 29, 31]. This is a narrower theory than legislative intent in that it only seeks to examine the historical artifacts produced as a part of the legislative process. Legislative history may provide some insight into the intent behind the statue, but it is not without potential legal concerns. For example, legislative history may provide conflicting viewpoints that fail to clarify the meaning of the statue [20]. Once again, this approach requires domain expertise outside that of the traditional requirements engineering background.

Purposivism, or the purposive approach, seeks to interpret statues based on an understanding of the goal of the legislative body that created the statute [20, 28, 29, 31]. This theory is related to legislative intent, but focuses exclusively on the goals of the legislative body rather than using artifacts of the legislative process. For example, many legal statutes have a preamble that introduces and explains the purpose of the statue. This preamble may not explicitly allow or disallow any particular act, but it can serve as a framework for interpreting the statute [29]. However, this too requires domain expertise outside that of the traditional requirements engineering background.

In contrast with legislative intent, textualism is a theory of statutory interpretation wherein an interpreter avoids any source of information other than the text of the statute itself

[20, 28, 29, 31]. Textualism is based on the structure and meaning of the words as they would be used by a skilled author [29]. In a textualist interpretation, any ambiguities, conflicts, or discrepancies must be resolved using the surrounding context rather than external sources such as goals or other legislative artifacts.

In this dissertation, we rely on a textualist theory of statutory interpretation for three reasons. First, the text of the law or regulation is always the initial source for an interpretation. Legal scholars rely on other theories of statutory interpretation primarily when the meaning of the legal text is ambiguous or unclear [29]. Furthermore, because we focus on identifying requirements that meet or exceed their legal obligations, rather than on requirements that definitively result in a known violation of the law, we need only to identify requirements that meet a strict, textual interpretation of the legal text. Any remaining requirements could be examined further using other theories of statutory interpretation. Second, other theories of statutory interpretation rely on summaries and histories that are not technically written into law. Otto et al. identify such sources as potential pitfalls for requirements engineers seeking to build legally compliant software systems [60]. Third, textualism is most similar to traditional requirements engineering techniques. Requirements engineers are familiar with resolving ambiguities, conflicts, and discrepancies in natural language. Importantly, if legal texts and requirements are assumed to be internally consistent in their use of language, then mappings of terms from one domain to another could also remain consistent and repeatable metrics based on natural language could be used to estimate legal implementation readiness. Chapters 3 and 4 describe our use of mapping and our development of metrics for legal compliance, respectively.

2.4 Requirements Triage and Prioritization

Quickly identifying implementation ready requirements is important for meeting legal compliance deadlines when using agile or iterative software development practices, which do not complete a comprehensive requirements analysis prior to beginning implementation. Requirements triage is the process of determining which requirements should be implemented given available time and resources [25]. The word 'triage' comes from the medical field where it refers to the practice of sorting patients based on which patients would most benefit from medical treatment [25, 67, 43]. Disaster victims can be triaged into three categories: those that would only recover if they receive medical attention, those that would recover regardless of treatment, and those with no hope of recovery [67, 43]. In this paper, we develop legal requirements triage to divide a set of requirements into the following three sets: (a) legal requirements that are implementation ready; (b) legal requirements that require further refinement; and (c) non-legal requirements.

Duan et al. created a process for creating a list of prioritized requirements from a set of customer requests and business requirements using semi-automated requirements prioritization and triage techniques [27, 43]. Their system, called Pirogov, uses an automated clustering algorithm that groups requirements [27, 43]. A requirements engineer then prioritizes these groups and creates weightings for the categories of requirements [27, 43]. A prioritization algorithm then creates a final prioritization [27, 43]. In this dissertation, we present an automated triage and clustering algorithm for legal requirements rather than customer requests or business requirements.

Triage alone is insufficient for requirements prioritization because it does not provide a complete ordering of requirements based on implementation readiness. Within the set of LIR requirements, some requirements may still depend on others. Requirements prioritization is the process of ordering requirements according to some previously agreed upon factor of importance [38]. Herein, we focus on prioritizing according to implementation readiness because it is critical for iterative software development processes.

Karlsson classifies two primary approaches to requirements prioritization: pairwise comparison techniques and numeral assignment techniques [36]. Pairwise comparison techniques involve ranking the relative priorities of each pair of requirements in the system [36]. Karlsson et al. show that pairwise comparison techniques require substantial effort upfront due to the quadratic growth of comparisons as the number of requirements increases [38]. Avesani et al. identify Analytic Hierarchy Process (AHP) as the reference method for pairwise requirements prioritization in case study-based requirements engineering research [7], and numerous researchers adapted AHP for requirements prioritization [33]. AHP provides heuristics for prioritizing elements from multiple domains based on multiple criteria [65]. In addition, AHP has mathematical support for ranking elements using priority vectors [65]. In the context of legal requirements, pairwise requirements prioritization techniques are at a disadvantage because the prioritization must be completely recalculated when a change in law occurs and the calculation effort grows quadratically with the number of requirements. In this dissertation, we discuss the use of pairwise prioritization in Section 4.2.3.

More recently, Herrmann et al. performed a systematic review of requirements prioritization techniques based on benefit and cost predictions [33]. They classified requirements prioritization techniques based on six groups: (1) fixed value priority; (2) grouping requirements; (3) relative value priority; (4) pairwise comparisons; (5) discrete scale ranking; and (6) benefit intervals [33]. These groups are not distinct, discrete groups, but sometimes overlapping classes of prioritization techniques. For example, it is possible to use both relative value priority rankings and pairwise comparisons.

The Herrmann et al. study showed that requirements prioritization during requirements refinement was an under-researched area of study [33]. In addition, Herrmann et al. found

that dependencies between requirements are often neglected when producing a requirements prioritization despite the fact that the requirements engineering community considers such dependencies to be important [33]. Herein, we address these needs in the context of legal requirements prioritization.

The numeral assignment prioritization technique is a ranking system in which requirements engineers assign numeral values to represent the priority of software requirements [36]. One such approach consists of assigning a discrete value to a requirement on a scale from 1 (optional) to 5 (mandatory) [36]. Karlsson showed that numeral assignment prioritization techniques take more time and produce less reliable prioritizations than pairwise comparison techniques [36]. In our prior work, we showed that assigning values to legal requirements based on the structure of a legal text can produce a useful legal requirements prioritization while requiring little training or domain knowledge [49]. In this dissertation, we introduce eight metrics for assigning numeral values to requirements representing legal implementation readiness characteristics of those requirements, and we describe a prototype tool that supports the automated calculation of these values.

2.5 Measurement and Requirements Metrics

Metrics are used in many domains such as insurance contracts and policy documents to empirically and objectively analyze natural language text. These metrics are useful for accurately examining aspects of natural language documents without requiring extensive analysis. Consider the Flesh-Kincaid Grade Level model [30]. In this model, the readability of a document can be accurately calculated without consulting numerous linguistics experts by measuring the readability of texts written in English by examining, for example, the average syllables per word and the average length of a sentence [30]. Despite the simple nature of their construction, metrics like the Flesch-Kincaid Grade Level metric prove useful in practice. Flesh-Kincaid, in particular, is included in numerous word processing systems and is a United States government standard [66]. Other models also exist to achieve this end, including the Gunning-Fog Index [32], the SMOG Index [57], and the Automated Readability Index [40].

Metrics have also been used to measure software readability. Buse and Weimer's metric model for software readability is based on simple constructs albeit within source code rather than natural language text [21]. For example, they measure the average number of identifiers per line of code and the average number of arithmetic operations per line of code. Using these and other metrics, their model of software readability is useful in examining software quality and provides insight on the design of programming languages.

Metrics must be validated to be empirically useful [41]. Meneely et al. conducted a systematic literature review of metrics validation in software engineering [58]. Their examination

identified forty-seven criteria for validating metrics, which sometimes conflicted or overlapped with one another [58]. They concluded that no standard currently exists for validating metrics in software engineering, and therefore researchers should choose a strategy that ensures the metrics they use or develop are validated for their intended use. In this dissertation, we attempt do precisely this—confirm that the legal requirements metrics are valid for use in determining whether or not a requirement is LIR.

2.6 Chapter Summary

Clearly, an examination of legal compliance in software systems requires background knowledge in many fields. The development of legal requirements metrics, presented in Chapter 4, draws from all of the related work discussed in this chapter. The iTrust case study described in the next Chapter served as the origin for the ideas presented in this dissertation.

Chapter 3

Tracing Existing Requirements to Legislation

"The lurking suspicion that something could be simplified is the world's richest source of rewarding challenges."

–Edsger Dijkstra

Although the focus of the U.S. Health Insurance Portability and Accountability Act (HIPAA) is broader than computer-based systems, it was intentionally constructed to cover them. Healthcare facilities have been using computers to manage basic health information, such as patient admission and billing, for decades [23]. However, most of these computer-based systems currently do not use software designed and built to support Electronic Health Records (EHR)[1] systems [26]. EHR systems store, update and transmit a patient's health records digitally. They are designed to reduce error rates in the administration and billing of medical services by storing and transmitting continuously updated medical records [23]. EHR adoption has progressed slowly: a recent survey published in the New England Journal of Medicine found that only 4% of respondents had "an extensive, fully functional EHR system," while 13% had a more basic EHR system [26]. Clearly, many new systems must be developed, deployed, and evaluated for legal compliance to provide every American with access to an EHR system.

Requirements engineers evaluating EHR systems for legal compliance must be able to establish and maintain traceability from software artifacts to the relevant governing legal texts[2] to demonstrate due diligence in satisfying security and privacy requirements [60]. Herein, *due diligence* refers to careful efforts taken to satisfy a legal requirement or to fulfill an obligation.

[1]Note that Choi et al. use the term electronic medical records (EMR) system rather than electronic health records (EHR) system [23]; we treat these terms as interchangable for the purposes of this dissertation.

[2]This dissertation uses "legal texts" to refer to both legislation and regulations.

The remainder of this chapter describes a case study in which we develop a methodology for tracing software requirements to legal texts for the purpose of evaluating legal compliance concerns. Our case study examines the iTrust Medical Records system, which is detailed in Section 3.1. We provide an overview of our methodology using some simple examples from our iTrust case study in Section 3.2. We then describe our methodology in detail throughout Sections 3.3, 3.4, 3.5, 3.6, and 3.7.

3.1 iTrust Medical Records System

We apply the methodology described in this chapter to the iTrust Medical Records System, an open-source EHR system designed as a prototype to be used in the United States healthcare industry. Although iTrust is being designed for future use in medical settings, to date it has been used exclusively in a classroom setting where it has been in continual development by students at North Carolina State University since 2004. Each semester, students are responsible for adding and testing both new and existing functionality. From an educational standpoint, the project exposes students to realistic challenges in software engineering: working within non-trivial code bases, managing information security and privacy issues, and using modern integrated development and testing environments [72].

The iTrust developers expressed their requirements as Unified Modeling Language (UML) use cases in consultation with both a practicing physician and a professional from the North Carolina Healthcare Information and Communications Alliance. Notably, the initial iTrust requirements called for HIPAA compliance. The iTrust system shares many characteristics with other existing software systems that must comply with laws and regulations. Financial services, industrial management, education, and many engineering industries also use computer systems extensively and are heavily regulated. Such systems must be evaluated and updated with each enactment or amendment of relevant laws and regulations to achieve compliance.

This chapter refers to iTrust as both an illustrative example system and as a part of our case study. For example, in Section 3.2, we outline our methodology for tracing existing requirements to legislation and highlight examples of this methodology using our iTrust case study. We provide an overview of the results of our iTrust case study in Section 3.8.

3.2 Methodology Overview

This methodology relies on manually generating traceability links[3] from software requirements to specific subsections of the legal texts. The process of manually generating these links enables

[3] Automated techniques may serve this function in the future [24].

engineers to produce a basic evaluation of their legal compliance concerns. A flowchart depicting an overview of the traceability methodology activities and their constituent steps can be found in Figure 3.1 pictured on Page 20.

The four traceability methodology activities shown in Figure 3.1 are described as follows:

Terminology mapping takes as inputs two sets of terms, which can include actors, data objects, or actions, and produces as an output a mapping between those sets. The first set of terms consists of the actors, data objects, and actions used in the software requirements. The second set of terms consists of the actors, data objects, and actions described in the relevant legal text. Section 3.3 describes this activity in detail.

Requirements identification and disambiguation takes as an input the set of requirements defined for the software system and the terminology mapping output produced during terminology mapping. This activity produces as output a different set of requirements that cover the original software system and requirements implied by the legal text with which the system must comply. Section 3.4 describes this activity in detail.

Requirements elaboration takes as an input the set of requirements produced during the requirements identification and disambiguation process. The output is a set of disambiguated and elaborated requirements that software engineers can trace to specific subsections of the legal text without extensive legal domain knowledge. Section 3.5 describes this activity in detail.

Tracing requirements to legal texts establishes traceability links from each requirement produced during the requirements disambiguation and elaboration activity to the relevant statements in the legal text with which each requirement must comply. Section 3.6 describes this activity in detail.

Each of these methodology activities must be completed for all system requirements prior to progressing to the next methodology activity. For example, all terms from the requirements must be mapped to the relevant terms from legislation as described in Section 3.3 prior to proceeding to the requirements identification and disambiguation step described in Section 3.4.

3.3 Terminology Mapping

In the first traceability methodology activity, analysts create a terminology mapping. During terminology mapping, analysts identify terms (actors, data objects, and actions) in both software documentation and legal texts. The associations between these software terms and

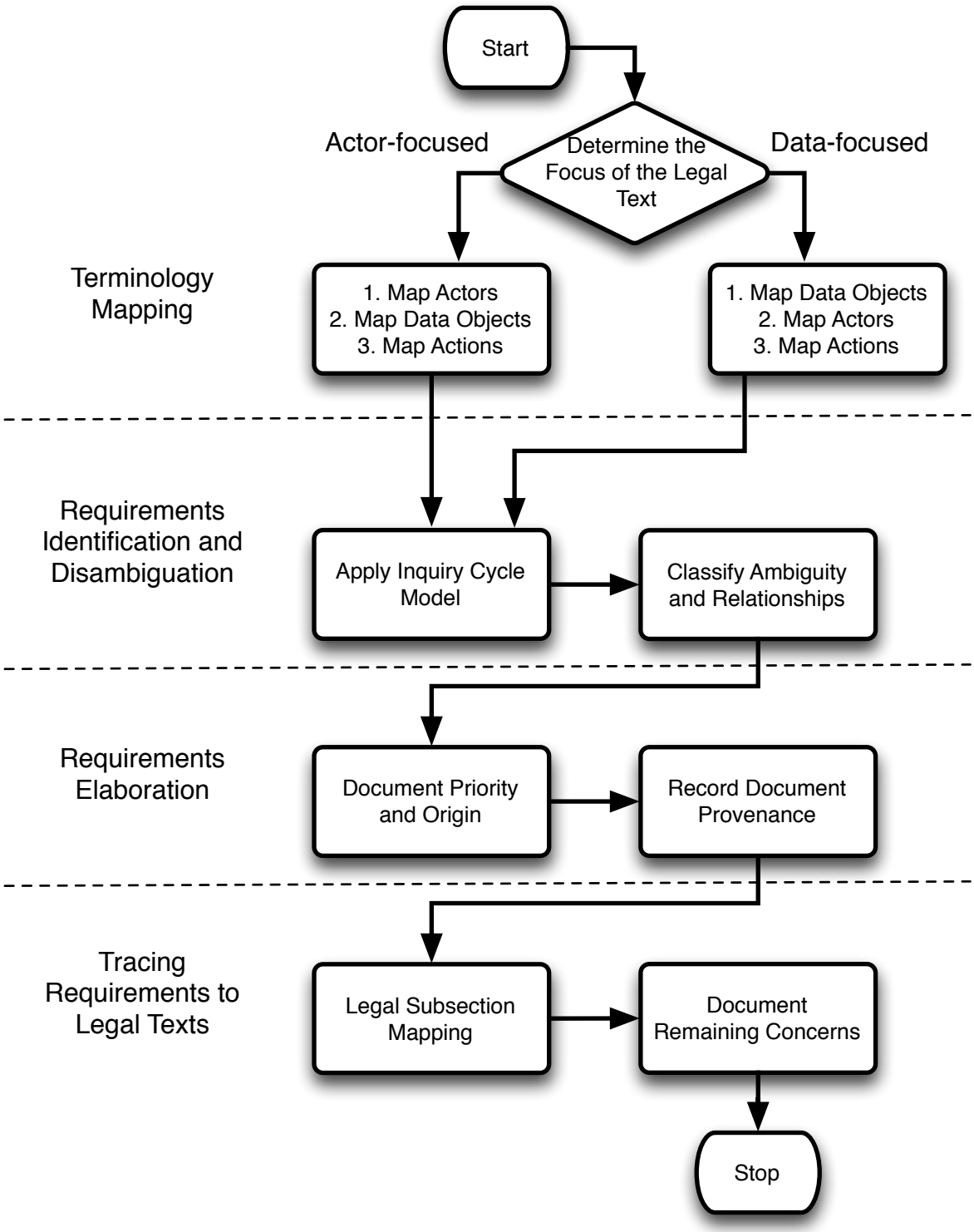

Figure 3.1: Overview of the Traceability Methodology

legal terms are then recorded. This terminology mapping provides an essential minimal level of traceability for establishing due diligence in security-sensitive and privacy-sensitive systems.

Legal domain experts and requirements engineers must first determine whether the legal text under analysis focuses on actor roles or data objects. Some legal texts, such as HIPAA, focus primarily on actor roles whereas others, such as the Graham-Leach-Bliley Act,[4] focus on data objects. HIPAA's focus on actor roles became apparent after constructing the actor hierarchy shown in Figure 3.7 and discussed later in this Chapter.

The three terminology categories that must be mapped are: Actors, Data Objects, and Actions, which we define as follows:

Actors / Stakeholders: Individuals or organizations explicitly mentioned in either the software documentation under analysis or the legal text with which compliance must be achieved. This does not include individuals broadly considered to be stakeholders (e.g., customers, developers) unless they are explicitly mentioned in the law or in the software documentation.

Data Objects: Information elements explicitly mentioned in either the software documentation under analysis or the legal text with which compliance must be achieved. For example, because HIPAA explicitly defines Protected Health Information (PHI), PHI is considered a data object.

Actions: Tasks performed by actors, which may or may not be legally compliant. Actions are explicitly mentioned in either the software documentation under analysis or the legal text with which compliance must be achieved.

Terminology mapping must be done systematically based on the focus of the legal text to reduce the effort required in the mapping process. Once determined, the primary focus area should be the first terminology type mapped. Actions should always be mapped last because they are dependent on actors and data objects. For example, if the legal text being studied is actor-focused, then the mapping order should be actors, followed by data objects, and then actions. If the legal text is data object-focused, then the mapping order should start with data objects, followed by actors, and then actions. Herein, we examine legal compliance with HIPAA, which is actor-focused. Thus, the terminology mapping is described in the following order: (1) Actors, (2) Data Objects, and (3) Actions. We now discuss the mapping of actors, data objects, and actions in turn.

[4]Pub.L. 106-102, 113 Stat. 1338

3.3.1 Tracing Actor Definitions

Mapping actors provides a basis for checking that the specified actions for the system actors are consistent with the corresponding legal actions for the actors specified in relevant law. It may also uncover important security, privacy, and legal conflicts, particularly when multiple mappings occur. For example, 'surgeons' may inherit legal obligations through their general classification as 'medical professionals,' such as the requirement to report evidence of potential child abuse incumbent upon all medical professionals. If this obligation is not recognized by the software system, then surgical residents may not be able to report suspected child abuse properly. Conflicts must be resolved by adapting actors in the software requirements to those of the legal text for the resulting system to be able to demonstrate legal compliance. Software requirements actors must be adapted because the legal specification is presumed to be unchangeable from the requirements engineer's perspective.

During legal requirements analysis, engineers sometimes need to clarify concepts with legal domain experts. Without a mapping of system actors to legislative actors, engineers and legal domain experts may believe they are discussing the same actor role when in fact they are discussing two or more different actor roles. Therefore, identifying actors in a system and mapping them to the corresponding actors specified in the relevant legal text is an important step in the process of establishing compliance with laws and regulations governing security and privacy. Unfortunately, mapping actors from one domain to another is complex because a single actor in a requirements document may fulfill zero, one, or many roles in a legal text. In addition, multiple actors in a requirements document may map to the same actor in a legal text.

Prior research has shown that it is helpful to create a stakeholder[5] role hierarchy to clearly delineate roles [17]. Creating this hierarchy helps analysts identify all actors both explicitly and implicitly referenced within the source materials. Hierarchies that appropriately express the legal actor roles as they relate to one another aid analysts in mapping from legal roles to those defined in software. Creating a stakeholder hierarchy from a legal text may further require a legal expert to properly disambiguate relationships among actors.

Consider a simple example wherein the requirements document has a "surgical resident" role and the legal text has a "medical professional" role as a general classification with both a "surgeon" role and a "pediatrician" role as sub-classifications in the hierarchy. The requirements document term "surgical resident" best maps to the legal term "surgeon" because their definitions are most similar. We call this type of mapping a direct mapping because the definitions of the two terms can be textually substituted for one another. In addition, the

[5]Note that we use a more specific definition of stakeholder than Breaux and Antón. We only consider stakeholders that are explicitly mentioned in the requirements document and in the relevant legal texts, whereas Breaux and Antón use stakeholder to refer to any entity with a 'stake' in the outcome [17].

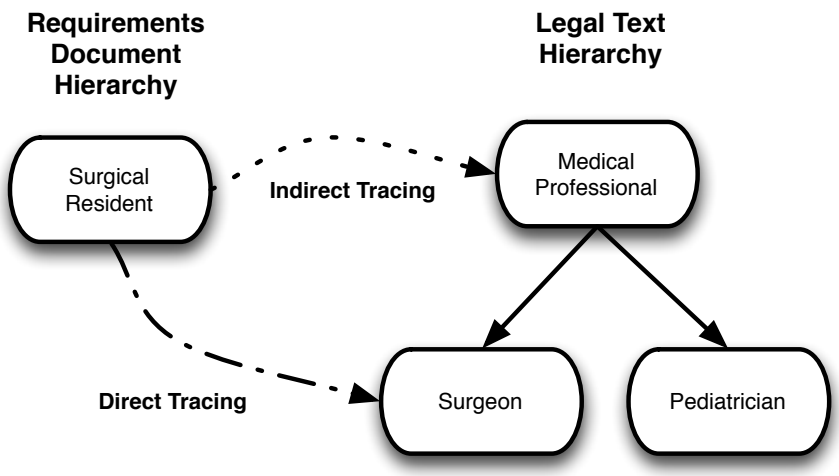

Figure 3.2: Example of Direct and Indirect Tracings from Software Terms to Legal Terms

requirements document term "surgical resident" maps to the legal term "medical professional" because the "surgeon" role is also legally responsible for the rights and obligations of its more general classifications in the hierarchy. We call this second type of mapping an indirect mapping. For the remainder of this chapter, when we use the generic term mapping we are referring to direct mapping because they must be actively created whereas indirect mappings are inherited[6] through the hierarchy. Figure 3.2 depicts this example.

In our case study, the results of which are described in Section 3.8, the requirements document contained the terms "Licensed Health Care Professional" and "Designated Licensed Health Care Professional." The former referred to a physician approved to view a patient's medical records, whereas the latter referred to a physician with an established relationship with a patient who is also approved to view a patient's medical records. Because both of these roles allowed a physician to access a patient's medical records, they were both mapped to the HIPAA role of "Healthcare Professional."

3.3.2 Tracing Data Object Definitions

The next activity entails identifying and mapping the data objects. Even if the legal text with which the software must comply is actor-focused, it still may be necessary to create a data object hierarchy for the legal text and map data objects from the requirements document to the legal text. This decision should be made in consultation with a legal expert. In HIPAA, identifying and mapping data objects is more straightforward than identifying and mapping

[6]The inheritance to which we refer is similar to that found in UML class diagrams with the exception that legal rights and obligations, as opposed to data or methods, are inherited by the subclass.

actors because HIPAA is primarily actor-focused. Given our prior experience with HIPAA [16, 17, 18], we determined that each iTrust data object could be mapped to a single data object in HIPAA for the purposes of our high level requirements analysis without the need for data object hierarchies.

Data object hierarchies may still be used, even with HIPAA, if requirements engineers and legal experts believe the complexity of either the requirements document or the legal text calls for such a mapping. For example, consider the term "Approved Diagnostic Information" to be a requirements document definition meaning "the set of diagnostic information a patient allows any licensed health care professional to view." In addition, consider Protected Health Information (PHI) to be a requirements document definition meaning "any spoken, written, or recorded information relating to the past, present, or future health of an individual." This definition for PHI could be textually substituted for that found in HIPAA §160.103, with key differences being exceptions listed in HIPAA and defined in other regulations.

These definitions of approved diagnostic information and PHI are not perfectly compatible since the sets of information are not equivalent. For example, a conversation with a physician would be considered PHI, but may not be considered as approved diagnostic information. A data object mapping may clarify the legal implications of this example situation. Storing a recording of every conversation between patient and doctor is outside of the scope of an EHR system. A summary of procedures performed, however, clearly falls within the scope of an EHR system. In the iTrust system, a summary of procedures performed could be considered approved diagnostic information. Determinations of this nature must be made through the data object mapping process.

In our case study, the requirements document referred to a "Medical Identification Number," which was a unique identifier for all individuals, including patients and staff, in the iTrust system. This data object mapped to the unique identifier required by HIPAA.

3.3.3 Tracing Action Definitions

After both actors and data objects have been identified and mapped, the actions themselves must be identified and mapped. Actions are tuples that consist of an actor, an operation, and a target, where the target can be an actor, a data object, or null. Actions are defined more explicitly in some legal texts than others and may or may not form a hierarchical structure. Requirements engineers must identify all actions described by the software requirements and map them to those described in the relevant legal texts.

Security, privacy, and legal compliance issues detected while performing action mapping usually result in actors or data objects being refined. Consider the action that takes place when a patient views his or her medical records using an EHR system. The action tuple consists

of the patient as the actor, the medical record as the target, and 'viewing' as the operation (e.g. <patient, medical record, viewing>). In HIPAA, patients generally have the right to view their medical records, but there are some notable security exceptions. For example, patients are not allowed to review psychotherapy notes regarding their own mental health. In this example, requirements engineers must refine the definition of the medical record data object because the definition makes no distinction between psychotherapy notes and other diagnostic information. This refinement must target the data objects because of the distinction discovered between psychotherapy notes and other medical record information. Refining or removing the "viewing" operation would result in legal non-compliance because patients do have the legal right to view some parts of their medical records.

In our case study, the iTrust requirements document consisted primarily of use cases. Since use cases identify specific uses of the system, we easily identified numerous candidate actions to map. However, identifying a single action described by HIPAA to map those actions to was more challenging. Some actions described by the iTrust requirements document could represent numerous actions described by HIPAA. For example, use case 10 described the process of "entering or editing a personal health record." This action corresponds to several actions as defined by HIPAA, including access control, disclosure of PHI, and logging transactions. Therefore, several actions identified in the iTrust requirements document, including use case 10, mapped to more than one action described in HIPAA. More details from our case study results are detailed in Section 3.8.

3.4 Requirements Identification and Disambiguation

After performing a terminology tracing from software artifacts to legal texts, software requirements must be identified and disambiguated from all available sources. Our methodology adopts three analysis techniques: the Inquiry Cycle model [63], ambiguity classification [5], and relationship identification [5]. During the analysis activities described in this section, any newly identified requirements are added and annotated accordingly, noting the origin for that new requirement.

To improve our understanding of and disambiguate the existing requirements, we first ask the following questions—as in the Inquiry Cycle model—about each requirement and record the answers [63]:

Q1: What does this mean? (*what-is* question type)

Q2: Why does it exist? (*what-is* question type)

Q3: How often does this happen? (*when* question type)

Q4: What if this doesn't happen or wasn't here? (*what-if* question type)

Q5: Who is involved in this description or action? (*who* question type)

Q6: Do we have any questions about this user role or use case? (*follow-on* question type)

Q7: Does this user role or use case have any open issues? (*assumption* documentation)

Q8: Does this user role or use case have legal relevance? (*what-kinds-of* question type)

Q9: What is the context of this user role or use case? (*relationship* question type)

Every requirement is annotated with the answers to these questions as well as any additional open issues, such as security and/or legal issues, or potential engineering challenges. The primary mechanism by which the requirements are disambiguated is by answering these questions.

Consider iTrust Use Case 4: "Enter/Edit Demographics," shown in Figure 3.3. Use Case 4 describes how patient demographic information can be added to iTrust. Questions 1 through 5 from our Inquiry Cycle prompted us to record relevant contextual information that was not previously explicit in the requirement. Clearly and explicitly recording the answers to these questions disambiguates the requirement.

Next, we identify and classify any ambiguities in the requirements. We classify the identified ambiguities as follows:

A1: Ambiguity is a natural language statement in which terms need further qualification or refinement [5].

A2: Incomplete ambiguity is "a specialized form of ambiguity that results from terms being left out of the documentation" [5].

Consider again Use Case 4 from Figure 3.3. This use case demonstrates both ambiguity and incomplete ambiguity. The use case does not clearly state the difference between entering and editing information. They are described as if they are the same action taking place throughout the same interface. However, the use of two terms implies that they are separate actions. This is a clear case of ambiguity. In addition, terms such as demographic information are not explicitly and completely defined, which is a form of incomplete ambiguity.

Once ambiguities have been identified and classified, we identify the kinds of relationships that exist between requirements. Understanding these relationships is critical for legal compliance, requirements prioritization, as well as to properly address security and privacy issues. Regulations can be filled with exceptions, follow-on obligations or refrainments, and simple prioritizations [17]. We employ the relationship types proposed by Antón et al. [5] as follows:

UC4 Enter/edit Demographics Use Case

4.1 Precondition: UC1 has completed and a patient has been created. The iTrust user has authenticated himself or herself in the iTrust Medical Records system [UC3].

4.2 Main Flow: Demographic information is entered and/or edited [S1, S2]. Once entered, the enterer/editor is presented a screen of the input to approve [E2].

4.3 Sub-flows:

[S1] A patient or personal health representative may enter or edit their own demographic information including their security question/answer according to data format 5.1. When answer to the security question is typed in, the answer should not appear on the screen (similar to how a password normally appears) and the answer should be confirmed (by the patient or personal health representative) before it is saved. [S4, E1].

[S2] HCP must enter the MID of a patient and then enter or edit demographic information with the exception of the patient's security question/password according to data format 5.1 [S4, E1].

[S3] An HCP may enter or edit their own demographic information according to data format 5.2 [S4, E1].

[S4] The enter/edit demographics transaction is logged [UC5].

4.4 Alternative Flows:

[E1] The system prompts the patient or HCP to correct the format of a required data field because the input of that data field does not match that specified in data format 5.1 or data format 5.2, as appropriate.

[E2] The enter/editor reviews the input and finds an error, he or she does not confirm the selection. He/She provides the correct input and submits again.

Figure 3.3: iTrust Use Case 4

R1: Constrains. Item A constrains Item B.

R2: Depends. Item A depends upon Item B.

R3: Operationalizes. Item A operationalizes Item B.

R4: Supports. Item A supports Item B in some fashion.

For Use Case 4, we identified and recorded dependencies with other requirements, such as authentication requirements, auditing requirements, and logging requirements. In addition, we identified and recorded an operationalization relationship because Use Case 4 operationalizes a high level requirement of maintaining medical records for patients. By explicitly recording these relationships, we know that Use Case 4 represents a central element in the iTrust system.

We focused exclusively on identifying relationships from one requirement to another and did not consider relationships from requirements to legal texts in this activity because relationships from requirements to legal texts are captured in the terminology mapping activity described in Section 3.3. When requirements are properly extracted from legal texts, all relationships identifiable in the legal texts will be mirrored in the requirements. We discuss the traceability connections between requirements and legal texts in Sections 3.6 and 3.7.

3.5 Requirements Elaboration

The requirements elaboration activity requires the analyst to: document each requirement's priority and origin, add mapping of initial user roles to new user roles, and move original supplementary material to appendices of the new requirements document. Requirements engineers should also any unresolved security and privacy issues in the annotations created during requirements identification. Newly identified requirements should be documented and annotated with newly identified issues.

3.5.1 Document Each Requirement's Priority and Origin

To support prioritization each requirement must be labeled with one of the following priority levels:

Critical: These requirements are the core elements of the software system; without these requirements, the system simply cannot function at all. This prioritization category includes security and privacy requirements that protect core functionality.

High: These requirements are not core elements in that they will not cause the system to fail if not met; these requirements may be elaborated with additional domain knowledge. This prioritization category contains all remaining security and privacy requirements.

Medium: These requirements are added by the requirements engineers, based on their own domain knowledge; in addition, these requirements elaborate the primary benefits of the software system.

Low: These requirements primarily address minor system aspects that enhance a user's experience during use of the software system.

Requirements engineers must document any specific legal implications uncovered during this activity. Note that requirements with no identified legal issues may still have legal implications. For example, a requirement that has no identified legal issue within the HIPAA regulations may still have a legal implication with respect other laws. The intent is to capture known legal implications to prioritize the requirements accordingly, not to ensure that all possible legal implications are documented.

The origin of a requirement helps to answer basic requirements questions such as Q2: "Why does it exist?" as introduced in Section 3.4. Each requirement must also be annotated with one or more origins to denote its provenance as follows:

Original (Use Case Number): Requirements extracted from the baseline iTrust requirements document [73]. The use case from which the requirement was extracted is denoted in parentheses.

Legal (HIPAA Regulatory Section Number): Requirements with a known legal issue. The regulatory section from which the issue originated is denoted in parentheses.

Common Domain Knowledge (Developer Name): Requirements commonly found in EHR systems that support basic healthcare information technology.

Development Experience (Developer Name): Requirements commonly found in web applications that support basic web application maintenance and usage.

The domain knowledge and development origins are defined herein in terms of the healthcare domain and web application development, respectively, due to the nature of the iTrust system; these would be adapted for other domains and development approaches as needed.

Based on the results from requirements identification and disambiguation in the previous step (see Section 3.4) and the results from this step, we identified and documented four system requirements from Use Case 4 (see Figure 3.3). Requirement 19, found in Figure 3.4, was one of those requirements. Note that it was found to be of "Critical" priority because it is a core element of the system, and it was identified as originating with the description of Use Case 4.

R-19: iTrust shall allow a physician, designated physician, administrative assistant, or medical administrator, each using their authorized accounts, to read or update unrestricted diagnostic information and patient record information.
Priority: Critical
Origin: Original (UC4)

Figure 3.4: iTrust Requirement 19

3.5.2 Record Document Provenance

Existing large software systems may have extensive documentation and may need to comply with several laws and regulations. In addition, legal texts, security policies, and privacy policies may evolve over time. In these cases, legal requirements analysis may need to be conducted iteratively or in stages. Requirements engineers should record the provenance of original documentation materials to support future iterations or compliance efforts. This documentation maintains traceability to the original elements of the software documentation when it is not possible to immediately update the requirements through the identification process. Furthermore, existing design documentation or source code may still refer to the original user roles; a mapping from original user roles to new user roles will prove useful when these software artifacts are updated to reflect the new requirements.

Although it is important to update supplementary material—such as original use cases or diagrams—to achieve legal compliance, it may not be practical for requirements engineers to make such changes to existing documentation while focusing on evaluating and improving the legal compliance of the requirements. To facilitate traceability to the original documentation, requirements engineers must move this supplementary material to appendices. (If the document is available in an electronic form, hypertext links provide excellent traceability support.) In addition, recording the provenance of original documentation supports security and privacy efforts by providing engineers with the ability to analyze both the new requirements and the original requirements for the same security flaws.

3.6 Tracing Requirements to the Legal Text

In the fourth activity, requirements engineers must attempt to resolve all remaining issues in the annotations and explicitly trace each requirement to the relevant regulatory section. Ideally, requirements engineers and any consulting legal domain experts will be able to accurately identify each requirement with its legal implications along with the specific origin of that

requirement. The legal origin of some requirements may have been identified in the previous activity, but requirements engineers must focus exclusively on tracing requirements to the relevant regulation as a separate activity.

Ideally, requirements are derived from law before any software requirements are specified. However, if the requirements did not originate from law and the regulatory compliance must be constructed given the existing artifacts. Because requirements engineers are not legal domain experts, they may not be able to locate the precise regulatory section to which a requirement may pertain without consulting a legal domain expert. If a legal domain expert is not available, the requirements engineer must document any remaining engineering, security, or legal issues in this step, defined as follows:

Engineering Issues: Issues relating to the software engineering process itself, including requirements, design, implementation, and testing.

Security Issues: Issues relating to the security and privacy of information stored in or services provided by the software system.

Legal Issues: Issues with legal implications that cannot be traced by the requirements engineer to the relevant regulatory section of the legal text.

Consider the example action discussed in Section 3.3.3 in which a patient uses iTrust to view his or her medical records. If a requirements engineer has completed all the steps in Sections 3.4 through 3.5 and still has an annotated issue regarding this requirement, then that issue must be classified and clarified for future consultation with a legal domain expert. The requirements engineer may have Engineering Issues related to implementing this feature because it may require additional password management for patients. The requirements engineer may have a Security Issue related to ensuring that patients are not able to use the feature to view medical records belonging to other patients. Although such a Security Issue may also be a Legal Issue, the requirements engineer may wish to document a specific Legal Issue regarding psychotherapy notes if they are unable to find the specific section within HIPAA regulating patient access to psychotherapy notes.

Requirement 21, found in Figure 3.5, was another requirement resulting from our analysis of Use Case 4 (see Figure 3.3). Our analysis of this requirement raised a legal concern we were unable to trace to a specific subsection of the legal text. We documented this concern as clearly as possible so that it can be examined and resolved later by a legal domain expert.

R-21: iTrust shall allow a patient, using their authorized account, to read or update their demographic information, the demographic information for the patients they represent, their list of personal representatives, the list of personal representatives for the patients they represent, their list of designated physicians, and the list of designated physicians for the patients they represent.
Priority: Critical
Origin: Original (UC4)
Legal Issue: We are not sure whether a personal representative should be allowed to remove himself from a list of personal representatives.

Figure 3.5: iTrust Requirement 21

3.7 Evaluating Software Requirements for Compliance with Legal Texts

The objective in applying our methodology was to improve the security and privacy of the iTrust software requirements and to evaluate their compliance with the HIPAA regulations. The artifacts available to us include the HIPAA regulations and version 12 of the iTrust software requirements specification. The requirements specification includes a brief introduction containing 12 glossary items, 27 use cases classified as functional requirements, and four non-functional requirements (one of which calls for HIPAA compliance) [73]. As tends to happen when a collection of use cases are employed as a substitute for an actual requirements specification [4], these use cases function as both requirements and design descriptions of the intended functionality within the iTrust system. In addition to the uses cases, the iTrust requirements specification includes a data field formats section that describes the attributes of the data stored in the database.

The iTrust requirements we identified using our methodology—along with the original use cases—are available in the iTrust Medical Records System Requirements Specification, a wiki-based document [74]. We identified 73 total requirements, of which 63 were functional requirements and 10 were non-functional requirements. The final requirements specification contained five sections, not counting appendices, as follows: Introduction, Description of User Roles, Functional Requirements, Non-Functional Requirements, and Glossary. The final requirements specification has automatic version control and enables one-click access to an index of term mappings, software requirements, use cases, and external references. We now present our findings from each analysis activity—for each of the activities in Figure 3.1, we separately present the findings in the following subsections.

3.8 iTrust Case Study Results

As preliminary validation for the methodology described in this chapter, we applied the methodology to the iTrust Medical Records System. We now describe our experiences and results from each step of our methodology in turn.

3.8.1 Results From Terminology Mapping

The iTrust documentation discusses eight actors: Patient, Administrator, Healthcare Professional (iTrust HCP), Licensed Healthcare Professional (LHCP), Designated Licensed Healthcare Professional (DLHCP), Unlicensed Authorized Personnel (UAP), Software Tester, and Personal Representative. In contrast, HIPAA encompasses a larger spectrum of actors than iTrust; there are at least 27 distinct actor roles discussed in the HIPAA regulations. To create a terminology mapping, each actor in the iTrust system must be mapped to the appropriate and/or corresponding actor(s) in the legal texts.

Figures 3.6 and 3.7 portray the complete stakeholder hierarchies for the iTrust system and the sections of the HIPAA regulations studied, respectively. These hierarchies operate in a similar, but not equivalent, fashion as class diagrams. Instead of inheriting all data or methods, inheritance operates on rights and obligations and with exceptions in these hierarchies. For example, in the HIPAA stakeholder hierarchy (Figure 3.7), an inmate does not inherit all the rights and obligations that a person would have. However, a Covered Health Care Provider (CHCP) may inherit all the rights and obligations for both a HIPAA HCP and a Covered Entity.

An actor with multiple roles in a legal text must account for duties stemming from each role. For example, in Figure 3.7, a Small Health Plan must also satisfy the requirements for a Health Plan. It is therefore critical to map each iTrust actor to each role that actor is responsible for in HIPAA. We identified both the direct and indirect actor mappings, and added them to the previously mentioned index of actor mappings. For example, when a single iTrust actor maps to multiple HIPAA actors, each of the HIPAA actors were added to the resulting term mapping for the iTrust actor. The results of this effort can be seen in Table 3.1.

Identifying the iTrust actors proved much easier than identifying the data objects or actions because the original iTrust requirements document was comprised of use cases, which describe user interactions with the resulting system. The HIPAA regulations do not explicitly define data objects for all data sets on which actors perform actions. In contrast, HIPAA does not clearly define the relationship between two data sets and the boundary between them. HIPAA broadly defines the following sometimes-overlapping data sets: electronic protected health information, health information, individually identifiable health information, and designated record set.

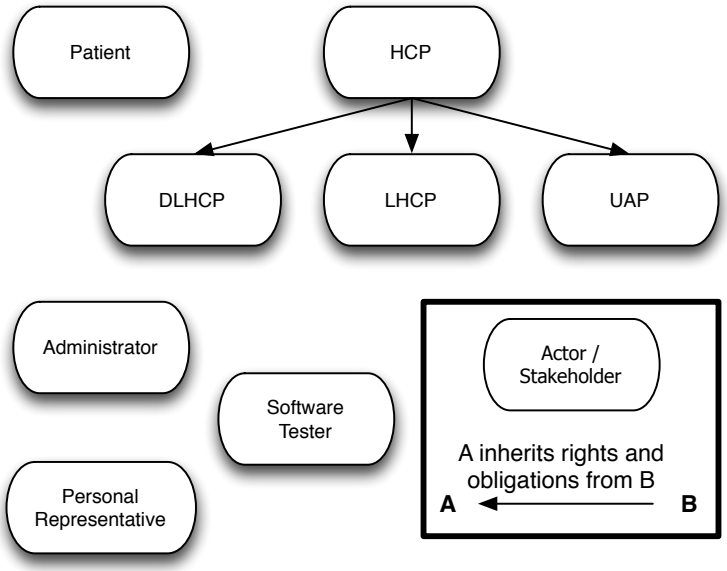

Figure 3.6: iTrust Actor Hierarchy

Table 3.1: Terminology Mapping

iTrust Actor	**Direct HIPAA Stakeholder**	**Indirect HIPAA Stakeholder**
Health Care Professional	CHCP	HCP, Person, CE, workforce
Patient	Individual	Person
Administrator	CHCP, Employer	HCP, Person, CE, Workforce
Licensed Health Care Professional	CHCP	HCP, Person, CE, Workforce
Designated Licensed Health Care Professional	CHCP	HCP, Person, CE, Workforce
Unlicensed Authorized Personnel	CHCP	HCP, Person, CE, Workforce
Software Tester	CE	Workforce
Personal Representative	Individual, Other Persons	Person

Because HIPAA's definitions of data are not always discrete sets, the same data object easily can fall into more than one category. Data object classification depends, in part, on the action being performed and the actor performing that action. For example, a standard health record could be considered as Protected Health Information (PHI) or as a part of a Designated Record Set, which is defined in §164.501 as "a group of records maintained by or for a Covered Entity (CE)." These broad data sets apply to a variety of healthcare providers because every CE does not collect the same patient health information elements. For example, psychotherapy notes are not present in all CE practices' data records.

Aligning the different actions been iTrust and HIPAA enables the requirements engineer to specify the security and privacy requirements so that all actions are legally compliant.

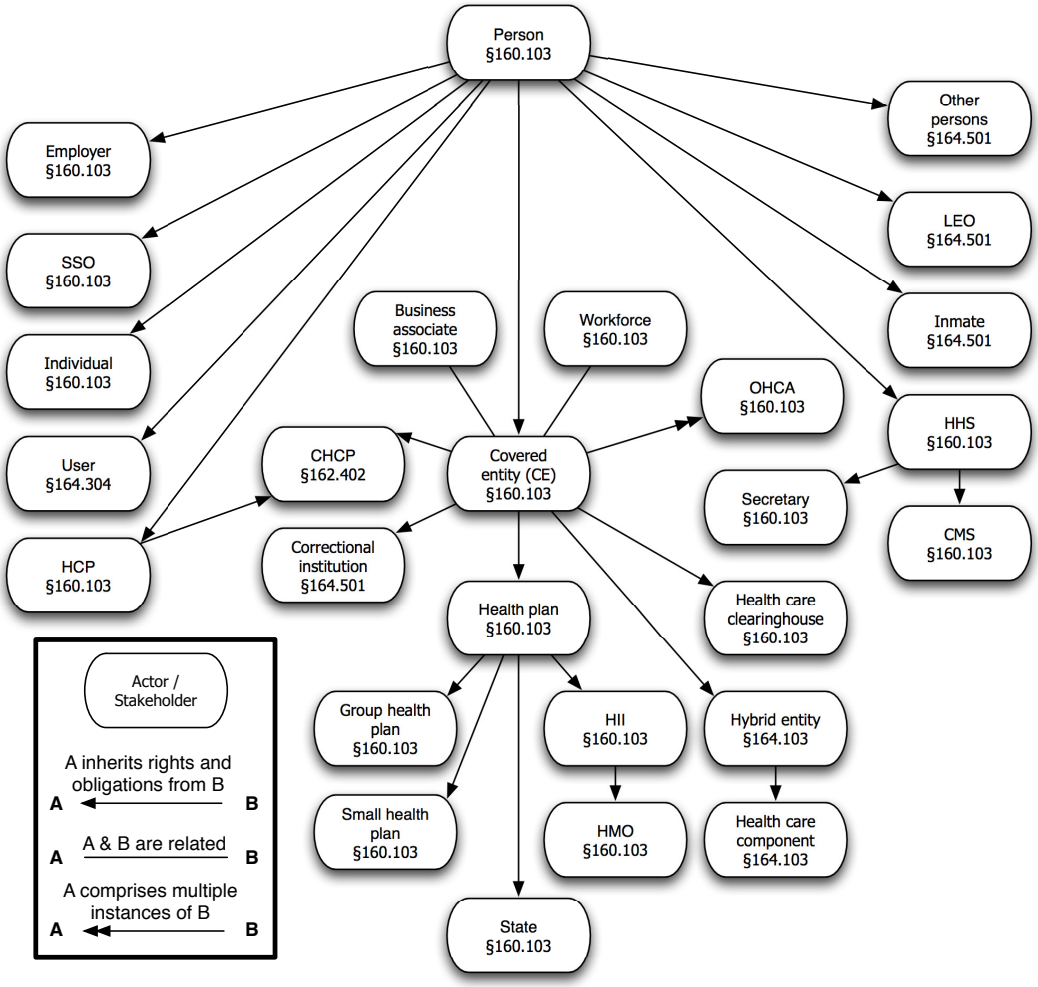

Figure 3.7: Partial HIPAA Stakeholder Hierarchy

Identifying security and privacy actions in HIPAA is more difficult than identifying actions in iTrust, however, because regulations are intended to be general, whereas use cases are intended to describe specific usage scenarios. For example, iTrust has an action "enter or edit demographic data" that could map to several different HIPAA actions including access control, disclosure of PHI, or transactions. In several cases, actions are explicitly defined in HIPAA. In §160.103, for example, a transaction is defined as "the transmission of information between two parties to carry out financial or administrative activities related to health care." This section also defines eleven types of information transmissions, some of which have different conditions that must be satisfied depending on the information transferred. For example, the action "coordination of health benefits" would have all the conditions for transferring PHI (defined in §160.103) as well as all the conditions for transferring payments (defined in §164.501). These conditions must maintain the same precedence in the requirements and the software system as they have in legal texts; care also must be taken to ensure that these conditions do not conflict with one another during secure operations. In contrast, iTrust has fewer and less detailed transactions, which are logged as the completion of an iTrust action.

Actions described in the software requirements that do not map or conflict with actions described in the legal text should be annotated as either a missing requirement or an unrelated element of the system. For example, iTrust has a Software Tester actor that performs a "determine operational profile" action, which is not explicitly defined in the HIPAA regulations. The data for this action is aggregate system usage information per user type. This action serves as part of system maintenance and testing and therefore may be considered as an unrelated element to establish legal compliance. It could be argued that including such a feature mixes software maintenance processes with software products, but operational profiles are neither prohibited nor permitted by HIPAA.

3.8.2 Results from Requirements Identification and Disambiguation

In Section 3.4, we explain how requirements identification and disambiguation entails applying the Inquiry Cycle model, identifying ambiguities, and recording relationships between requirements. In the iTrust system, this requires us to examine the glossary terms, introductory statements and the 27 use cases found in the iTrust requirements specification. From this analysis, we identified 53 new software requirements. Because we focused on security and privacy requirements stemming from the HIPAA regulations, requirements beyond the scope of an EHR were tabled for future analysis.

Recording the rationale and context for each iTrust requirement is challenging for security, privacy, and legal compliance until one can justify each requirement by tracing it to a specific section in the law. Business rationale is important for legal compliance because business

requirements may conflict with legal requirements. If such a conflict occurs the requirements engineer should use the legal rationale to guide the resolution of the conflict. Business rationale was missing from the original iTrust requirements specification, so we did not encounter this type of conflict. However, we did develop legal rationale for the requirements identified in this activity.

Consider iTrust's third use case, UC3. This use case describes a scenario in which users are authenticating themselves to the iTrust system. This use case is replicated in Figure 3.8 for reference.

3.8.3 Results From Requirements Elaboration

Prioritizing the security and privacy requirements proved the most challenging activity in our iTrust study. HIPAA includes prioritizations and exceptions, but these prioritizations and exceptions are distributed throughout the regulatory text as opposed to being collected and clearly defined in a single location. We classified the prioritization based on engineering issues into categories: Critical, High, Medium, and Low. We next identified any requirements with clear legal issues and marked them as higher priority than the other requirements in their respective categories. Table 3.2 displays the number of requirements in each priority classification.

The original iTrust requirements specification did not contain the origin or rationale for each use case. We expressed the origin each newly identified requirement to clarify legal issues, but we were limited by this lack of source information in the original documentation. For example, we were unable to identify any use cases that were explicitly influenced by the original consulting physician and professional from NCHICA. Table 3.3 shows the number of requirements that originated from each of the categories.

We also updated requirements and definitions extracted in the second activity as we analyzed our annotations from the previous activity and found statements that needed further elaboration.

Table 3.2: Requirement Prioritizations

Priority	**Number of Requirements**
Critical	37
High	19
Medium	5
Low	2

UC3 Flow of Events for the Authenticate Users Use Case

3.1 Preconditions: UC1/UC2 has completed and a user has been created.

3.2 Main Flow: A user enters their MID and their password to gain role-based entry into the iTrust Medical Records system [E1, S4] or requests that their password be changed [S1]. A session that has been inactive for too long is terminated [S3]. An authenticated session ends when the user logs out or closes the iTrust application.

3.3 Sub-flows:

[S1] Present security question [S2, E1] and obtain answer.

[S2] If answer to security question is correct, allow user to change their password.

[S3] Electronic sessions must terminate after a "pre-determined" period of inactivity. Allow the administrator to set the length of this period of time and ensure that all authorization is disabled after a period of inactivity that exceeds this length.

[S4] The transaction is logged [UC4].

3.4 Alternative Flows:

[E1] The user may try three times. After three failed attempts with a user ID in a given session, disable the user for 15 minutes (see comments in the source code).

Figure 3.8: iTrust Use Case 3

Table 3.3: Origin Categories

Origin	Number of Requirements
Original Use Cases	48
Legal	3
Domain Knowledge	14
Development Experience	8

3.8.4 Results From Tracing Requirements to Legal Texts

During our study, we ensured that every requirement was traced to the relevant HIPAA regulatory section to the statement level. Additionally, we recorded open issues and questions throughout our examination of the iTrust documentation. For example, iTrust required a privacy policy to be implemented as a part of the system:

Requirement 65 Description: iTrust shall have a privacy policy, which is linked off of the login screen.

In revising this requirement, we were unsure who should be able to edit this privacy policy. As a result, we annotated the requirement with the following question and answer:

Question: Should the privacy policy be editable in iTrust by a medical administrator role?
Answer: Unknown at this time.

We were unable to answer this question because although HIPAA §164.520 regulates "Notices of Privacy Practices", it is unclear whether a particular user role in an EHR system should have exclusive access to create or edit these policy statements. We must consult with legal and healthcare domain experts to answer this question.

Finally, we attempted to categorize all remaining annotations and issues as engineering issues, security issues, or legal issues (as defined in Section 3.6). Categorization may help resolve the issues in the future by providing direction to requirements engineers on which type of domain expert to consult. Table 3.4 illustrates the number of each type of issue we documented during our study of the iTrust system:

The 16 engineering issues reflect open questions regarding the future design and development of iTrust as an EHR system. For example, Requirement 15 has an engineering issue:

Requirement 15 Description: iTrust shall allow a user, using their authorized account, to request their current password be emailed to their recorded email address and shall require

the user's user ID be provided to satisfy this request.

Engineering Issue: Users may also forget their user names. This is currently not handled by the system.

To address this engineering issue, requirements could specify a user name recovery mechanism. These requirements must be analyzed for security and legal compliance.

The four security issues relate to security design decisions, which must be addressed in a future iteration of the iTrust code. Requirement 15, shown above, also has a security issue as described below:

Security Issue: If a user has forgotten their password and has a new email, they cannot access their account in any fashion.

To address this security issue, requirements could be created to allow users to notify an administrator regarding their inability to access their own accounts. The requirements for this new feature must also be analyzed for security and legal compliance.

The 23 legal issues comprise questions about the precise legal status of a particular element in iTrust. For example, HIPAA allows minors or patients who are incapable of making medical decisions for themselves to have personal representatives that act on their behalf. Requirement 21, which has a legal issue, handles the management of personal representative lists as described below:

Requirement 21 Description: iTrust shall allow a patient, using their authorized account, to read or update their demographic information, the demographic information for the patients they represent, their list of personal representatives, the list of personal representatives for the patients they represent, their list of designated physicians, and the list of designated physicians for the patients they represent.

Legal Issue: We are not sure whether a personal representative should be allowed to remove himself from a list of personal representatives.

If a personal representative were able to remove his or her name from the list of personal

Table 3.4: Summary of Issues Identified

Type of Issue	Number of Issues
Engineering	16
Security	4
Legal	23

representatives, then patients may be incapable of making medical decisions for themselves and also no longer have a personal representative. A legal domain expert must address this legal issue and similar legal issues for which we are lacking domain knowledge; we are consulting with lawyers as we continue our work on the iTrust system.

It is important to note that each of our legal issues could also be considered security issues because we limited our focus to legal issues involving security and privacy. However, we chose to record them as legal issues because of the importance of demonstrating due diligence and avoiding legal non-compliance.

During this activity, we also constructed new non-functional requirements to reflect the broader issues and maintenance goals of iTrust. For example, the non-functional requirement below is an ongoing maintenance goal:

Requirement 67 Description: iTrust code shall adhere to the Java Coding Standards.

3.9 Lessons Learned

We now highlight four specific lessons learned through the development and application of our legal compliance methodology. In particular, we discuss lessons regarding the use of actor (or stakeholder) hierarchies, complications resulting from unresolved ambiguity, requirements prioritization, and tool support.

Lesson 1: Actor hierarchies are essential for security and legal compliance. Building actor hierarchies for both the requirements document and the legal text allows the requirements engineer to clearly and unambiguously state the legal rights and obligations for actors in the resulting software. Without explicitly constructing these hierarchies, the rights and obligations for actors that have indirect mappings to actors in the legal text are much harder to identify.

The differences between the iTrust Health Care Professional (iTrust HCP) and the HIPAA term Health Care Professional (HIPAA HCP) exemplify the usefulness of these hierarchies. For reference, Figure 3.6 depicts the iTrust actor hierarchy and Figure 3.7 depicts the HIPAA stakeholder hierarchy. In iTrust, Health Care Personnel (iTrust HCP) refers to any health care professional allowed to access health records; this iTrust actor maps to multiple actors in HIPAA. Specifically, the iTrust HCP could map to either the HIPAA Covered Health Care Provider (HIPAA CHCP)[7] or the HIPAA Health Care Providers (HIPAA HCP), defined in §162.402 and §160.103 of the HIPAA regulations, respectively.

Both of these mappings are necessary because the HIPAA HCP and the HIPAA CHCP are each assigned different sets of rights and obligations by the law. For example, when we

[7]The CHCP definition is not within the scope of our analysis of HIPAA. However, HIPAA compliance would require considering this definition.

directly map the iTrust HCP to the HIPAA CHCPs, then the iTrust HCP also indirectly maps, as explained in Section 3.3.1, to HIPAA HCP, Persons, and Covered Entities (CE), each of which are defined in §160.103 and shown in Figure 3.6. As a result, the iTrust HCP must satisfy the rights and obligations for HIPAA CHCPs as well as HIPAA HCPs, Persons and CEs.

Alternatively, when we map the iTrust HCP to the HIPAA HCP, then the iTrust HCP directly maps to the HIPAA HCP and indirectly maps to the Person actor. In this case, the iTrust HCP does not have to satisfy the rights and obligations for HIPAA CHCPs or CEs, but it would have to satisfy the rights and obligations of the HIPAA HCP and Person.

A system designer implementing the requirements for an iTrust HCP could easily overlook some of these responsibilities if they are not explicitly mapped. For example, a HIPAA CE is responsible for obtaining authorization to disclose PHI to an employer because it is outside the normal course of treatment, payment, and operations. This is the sort of responsibility that should have a single actor with an explicitly defined role in an EHR system, which correlates to multiple HIPAA actors. By explicitly noting this relationship, we improve the likelihood that system developers will implement it properly.

Lesson 2: Unresolved ambiguity can lead to security and privacy non-compliance. Identifying, classifying, and resolving the ambiguity found in the requirements is one of the most challenging and rewarding aspects of our methodology. To resolve ambiguity properly, requirements engineers must be able to infer the intended meaning in the statements. Evaluating each requirement to determine if it contains ambiguity or incomplete ambiguity makes it easier for requirements engineers to identify ambiguity at all.

For example, UC4, which was discussed in Section 3.4 provided examples of both ambiguity and incomplete ambiguity. Figure 3.3 on page 27 describes this use case. Upon initial inspection, this use case appears quite detailed, but several terms, including demographic information, are not clearly defined in the rest of the document. Each ill-defined term is an instance of incomplete ambiguity reflecting a need for clarification.

We also identified two clear instances of ambiguity upon further inspection. First, the entire use case details two separate actions (entering and editing demographic information), but does not clearly state if they are conducted using the same interface. Second, error condition 2, denoted as [E2], does not state clearly whether or not the second submission of demographic information is checked for errors by the iTrust system. By analyzing and classifying the ambiguities in UC4, we were able to identify and improve requirements with clarity.

Lesson 3: Prioritizing requirements aids in identifying critical security requirements. As part of our requirements elaboration, we prioritized requirements according to the following categories: Critical, High, Medium, and Low. For example, Requirement 14 was classified as a Critical priority requirement, which indicates that it may have serious security implications and it should be built early in the implementation phase. The description for this requirement

is "iTrust shall generate a unique user ID and default password upon account creation." If iTrust were to generate a duplicate user ID, then a patient could access another patient's medical records, which is a clear HIPAA violation under §164.321 and a serious security issue.

Lesson 4: Requirements engineers need tool support for determining legal compliance. The single most frustrating aspect of extracting requirements from use cases was the lack of tool support. Berenbach also highlights the difficulty in maintaining UML through requirements evolution owing to the lack of tool support [13]. Since we chose not to include use cases in our requirements, we did not face this issue. Potts et al. identify tool support as helpful, but it is not a major focus of their work [63]. Our experiences affirm that tool support would be helpful. Many of the tasks of writing and formatting requirements were repetitive, tedious, and easily could have been automated to improve our efficiency. Throughout our experience with the iTrust requirements we compiled the following list of features we would like to see in such a tool:

- Automatic generation of links to glossary terms.
- Automatic ordering of both requirements and glossary terms.
- Support for generating documents tailored to different audiences (e.g., a document showing only the requirements that deal with a particular user role or a document showing only requirements of a particular priority).
- Support for denoting issues in the wiki, which could be automatically and optionally removed from the final document.
- Support for foldable nodes for sections in the requirements document similar to that found in many source code editors.
- Support for a mechanism that allows easy defining of glossary terms and keywords.
- Support for wiki-style editing and formatting.
- Support for searching particular elements of the document while ignoring others (e.g., searching just the issues and not the requirements or the glossary terms).

These features address several key elements identified by Otto and Antón as supporting legal compliance within requirements engineering [60]: Automatic ordering of both requirements and glossary terms supports prioritizing requirements. The use of wiki-style editing supports management of evolving legal texts. The use of hyperlinks supports tracing from requirements to legal texts. Including a mechanism that allows easy defining of glossary terms supports the creation and management of a data dictionary. Support for searching particular

elements of the document while ignoring others enables semi-automated navigation and searching.

3.10 Chapter Summary

This chapter presents our methodology for tracing existing requirements to relevant legislation. This methodology provides engineers with the ability to evaluate an existing set of requirements for legal compliance. It also guides engineers so that they can systematically generate traceability links from requirements to the specific subsections of the legislation to which they must comply. As a result of developing this methodology, it became clear that the traceability links themselves are extremely useful for for evaluating software requirements for legal compliance. In the next chapter, we explore legal requirements metrics as a first attempt to determine whether traceability links alone prove useful for this purpose.

Chapter 4

Defining Legal Requirements Metrics

"If you cannot measure it, you cannot improve it."

–Lord Kelvin

The American Recovery and Reinvestment Act (ARRA)[1] included provisions for data breach notification that must be implemented as early as January 1, 2011 for some organizations and by January 1, 2013 for all covered organizations. Compliance deadlines and serious penalties for non-compliance raise the question of where software engineers should begin. In the context of legal requirements, the order in which requirements should be designed and implemented in software should be influenced by legal domain knowledge [60]. One way to reason about this order is to perform requirements triage. Requirements triage is the process of determining which requirements should be implemented given available time and resources [25]. Legal requirements triage allows software engineers to focus on requirements that have legal implications early in the engineering process, avoid expensive refactoring, and demonstrate legal due diligence by incorporating laws and regulations efficiently throughout the engineering effort.

To begin implementation, developers need a set of implementation ready requirements. In the context of legal requirements, implementation order should be influenced by legal domain knowledge [60]. A requirement is considered to be *Legally Implementation Ready* (LIR) if it meets or exceeds its legal obligations as expressed in relevant regulations. Software engineers must be able to discern those requirements that are implementation-ready from those that require further disambiguation and/or elaboration.

When new laws and regulations take effect, or existing laws and regulations are amended, software engineers need to quickly assess the ways in which those legal changes will impact

[1] Pub. L. No. 111-5, 123 Stat. 115 (2009)

software system requirements. Otto and Antón highlight the need to manage the evolution of laws and regulations over time in order to facilitate legal compliance efforts [60]. For example, in 2009 Congress amended HIPAA with the HITECH Act. Among the HITECH Act's provisions were new data breach notification requirements. In addition, administrative agencies update regulations regularly in response to new laws and changing conditions. Again using the HITECH Act as an example, both HHS and the Federal Trade Commission promulgated new regulations pursuant to the HITECH Act within six months of the act becoming law.

In this chapter, we introduce and define our legal requirements metrics in Section 4.1. We also present an example algorithm for combining them to generate a legal implementation readiness decision in Section 4.2. Our legal requirements metrics comprise simple, consistent attributes of software requirements and a legal text. These metrics were designed to function as a part of a legal requirements triage process by quickly determining from a set of requirements traced to subsections of a legal text to which they must comply which requirements were LIR and which were not. The software requirements must have previously been mapped to specific sections of the legal text to which it must comply as described in Chapter 3.

4.1 Legal Requirements Metrics

Legal texts are hierarchical documents. Without knowing the meaning of a legal text, we can still infer some meaning from its structure [29]. Consider the structured text in Figure 4.1. It is a hierarchical document with three levels. We are able to infer that levels (i) and (ii) are more specific than levels (a) and (b) because they are deeper in the hierarchy. For these metrics, the only non-hierarchical elements of the legal text that must be identified are cross-references and exceptions, which enable further inferences based on the structure of the legal text. For example, a cross-reference indicates that the section in which it occurs may include additional, possibly unspecified, requirements. In addition, an exception indicates that that the associated legal text is meant to be interpreted differently than the rest of the section.

The use of the structure of a legal text to determine part of its meaning is an accepted part of the textualism theory of statutory interpretation [20, 28, 29, 31]. This theory is considered a basic standard that, if met, provides some assurance of compliance with the legal text. Therefore, if our metrics are able to measure the extent to which a software requirement matches the structure (e.g. subsection count) and meaning (e.g. cross-references) of the legal text, then they will accurately estimate the legal implementation readiness of the requirement. Recall from Chapter 2 that there is no single "correct" interpretation of the law. Rather there are many different theories for interpreting the law. Thus, legal compliance is closer to a probability than a binary state. The goal of our triage metrics is to estimate, rather than

a. Lorem ipsum dolor sit amet.

   (1) ***Except when:*** ed do eiusmod.

   (2) Incididunt ut labore et dolore.

      i. Magna aliqua.

      ii. Ut enim ad: ***Section (f)(2)***.

   (3) Quis nostrud exercitation.

      i. Fugiat nulla pariatur.

      ii. Consectetur adipisicing.

   (4) Ullamco laboris nisi ut aliquip.

b. Ex ea commodo consequat.

c. Duis aute irure dolor.

Figure 4.1: Sample Hierarchical Legal Text

precisely determine, which requirements are legally implementation ready and which need further refinement.

Our legal requirements triage metrics are classified into three categories: *Dependency*, *Complexity*, and *Maturity*. Dependency metrics estimate the extent to which a requirement may be dependent on other requirements, which may be potentially unknown or unstated. Complexity metrics estimate the engineering effort required to implement a particular requirement. Maturity metrics estimate a requirement's readiness for implementation. Definitions for the metrics themselves can be found in Table 4.1.

Name: Sample Requirement

Description: Occaecat cupidatat non.

Legal Subsections: (a)(1) and (a)(2)(ii)

Figure 4.2: Sample Requirement

Table 4.1: Triage Metrics

Category	**Variable**	**Name**	**Description**
Dependency	S_M	Subsections Mapped	The number of subsections to which a requirement maps.
Dependency	C	Cross-references	The number of cross-references found within subsections to which a requirement maps.
Complexity	N_W	Number of Words	The number of words found within subsections to which a requirement maps.
Complexity	N_S	Number of Sentences	The number of sentences found within subsections to which a requirement maps.
Complexity	S_C	Subsection Count	The number of subsections recursively counted within the subsections to which a requirement maps.
Complexity	E	Exceptions	The number of exceptions within subsections to which a requirement maps.
Maturity	S_D	Subsection Depth	The deepest-level subsection to which a requirement maps (SC_D) minus the number of subsections to which that requirement maps (S_M).
Maturity	S_F	Subsection Fulfillment Percentage	The percentage of mapped subsections in the highest-level sections to which a requirement maps.

4.1.1 Dependency Metrics

Dependency metrics estimate the extent to which a requirement may depend on other requirements, which may be potentially unknown or unstated at the time of the analysis. The first Dependency metric computes the number of subsections mapped (S_M) from each requirement to the relevant elements of a legal text. Recall that requirements traced to specific sections of a legal text are a required input to determine the value of our legal requirements metrics. To calculate S_M count each mapping of a requirement to a subsection once. Consider the sample requirement from Figure 4.2, which maps to sections (a)(1) and (a)(2)(ii) from Figure 4.1; its S_M score is two because the sample requirement maps to two subsections.

The second Dependency metric computes the number of cross-references (C) found within the subsections to which the requirement is mapped. Because subsection (a)(2)(ii) of our sample hierarchical document is a cross-reference, as identified by the phrase *Section (f)(2)*, $C = 1$ for the requirement from our previous example. It is important to note that identifying cross-references in a legal text is far easier than understanding the legal meaning of a legal text. In fact, requirements engineers do not even need to be able to identify which legal text the cross-reference refers to; simply knowing that it is a cross-reference is enough for the purposes of this metric. It is worth noting that identifying cross-references may provide other benefits, such as the ability to identify conflicting software requirements [51].

When combining metrics in an algorithm to determine legal implementation readiness, they must be normalized and weighted according to the particular legal requirements triage needs at the time of the triage. Normalization and weighting allow a requirements engineer to reduce their risk of misidentifying a requirement as LIR when it actually has legal compliance concerns that should be addressed. A requirements engineer seeking to identify only those requirements where dependencies within the legal text are most likely addressed would heavily weigh the Dependency metrics. Although this is discussed in detail in Section 4.2, it is worth noting here that the Dependency metrics S_M and C are normalized as percentages of the total number of subsections in the legal text (S_T) and the total number of cross-references in the legal text (C_T), respectively. After normalization, the maximum possible value for $\frac{S_M}{S_T}$ is one and the maximum possible value for $\frac{C}{C_T}$ is one.

4.1.2 Complexity Metrics

Complexity metrics estimate the engineering effort required to implement a particular requirement. Herein, we define four Complexity metrics: number of words (N_W), number of sentences (N_S), the subsection count (S_C), and the number of exceptions (E). The number of words (N_W) and number of sentences (N_S) are calculated by summing the total number of words and sentences respectively found in the sections of the legal text to which the requirement

maps. Sentences can be detected using Manning and Schütze's algorithm [45]. For the sample requirement in Figure 4.2, $N_W = 22$ and $N_S = 5$ because that requirement contains 22 words and five sentences.

The subsection count (S_C) is calculated by recursively counting the number subsections beneath the subsection to which a requirement maps until all subsections have been counted once. This count includes the subsection to which the requirement is initially mapped. Consider the sample requirement in Figure 4.2. It maps to two subsections: (a)(1) and (a)(2)(ii). Neither of those subsections includes any subsections themselves. Thus, for the sample requirement, S_C is two. However, if our sample requirement was mapped to (a)(3) and (b), then the S_C would be four because there are two subsections under (a)(3) in addition to the initial mappings.

The number of exceptions (E) is calculated by summing the total number of exceptions found in the subsections to which a requirement maps. An exception is a specific condition specified in a legal text under which broader legal conditions do not hold [17, 15]. Often, the structure of an exception is similar to a case statement in a procedural programming language [17, 15]. In such an instance, each subsection is marked as an exception and counted separately for the purposes of this calculation. In our sample legal text, section (a)(1) contains a single exception. As a result, $E = 1$ for the sample requirement in Figure 4.2. However, if subsection (a)(1) from the sample legal text in Figure 4.1 had subsections detailing two conditions when the exception held, such as (a)(1)(i) and (a)(1)(ii), then E would be two.

When combining metrics in an algorithm to determine legal implementation readiness, they must be normalized and weighted according to the particular legal requirements triage needs at the time of the triage. A requirements engineer seeking to identify those requirements where complexities within the legal text are most likely addressed would heavily weigh the Complexity metrics. The N_W and N_S metrics can be normalized as percentages against the total number of words (W_T) and sentences (ST_T) in the legal text, respectively. The S_C and E metrics are normalized against the total number of subsections in the legal text (S_T) and the total number of exceptions in the legal text (E_T), respectively. As with the dependency metrics, the maximum possible value for each of the normalized Complexity metrics is one.

4.1.3 Maturity Metrics

Maturity metrics estimate the implementation readiness of a requirement. Estimating maturity is an important part of legal requirements triage because it distinguishes between requirements that represent a complex legal obligation as stated in the legal text and requirements that represent numerous, less complex legal obligations. The former are requirements that have been stated as well as possible given the complexities of the legal text; the latter are requirements that could be refined and still maintain legal compliance.

Consider the transaction code sets described in the HIPAA Transaction Rule (§162.900 through §162.1802). A requirement for transaction logging may be traced to significantly more sections in the Transaction Rule than a privacy requirement might map to in the Privacy Rule even if both requirements are specified optimally given the descriptions in the legal text. Although such a transaction logging requirement needs refinement from a software engineering standpoint, it accurately represents the requirements needed for legal compliance. Thus, it may be ready for additional requirements analysis or even implementation. Without Maturity metrics, this requirement may have consistently high triage scores and may be classified as needing refinement for many iterations.

We developed two maturity metrics to address concerns regarding requirements that cannot be refined further based solely on the legal text: the subsection depth (S_D, found in Equation 4.1) and the subsection fulfillment percentage (S_F, found in Equation 4.2).

$$S_D = SC_D - S_M \tag{4.1}$$

Subsection depth is a maturity estimate because deeper level sections of a hierarchical legal text tend to be more specific than higher level subsections. Thus, the deeper the requirement maps within the legal text, the more specific it is likely to be. The subsection depth (S_D) is calculated as the deepest-level subsection to which a requirement maps (SC_D) minus the number of subsections to which that requirement maps (S_M). We subtract the number of subsections to which a requirement maps because mapping to numerous subsections of a legal text makes a requirement less specific. Consider our sample requirement maps to two subsections of our sample legal text: (a)(1) and (a)(2)(ii). The deepest of these is (a)(2)(ii), which as a depth of three. Since $SC_D = 3$ and $S_M = 2$ for our sample requirement, S_D is one.

$$S_F = \frac{\left(S_M + S_O\right)}{SC_H} \tag{4.2}$$

Subsection fulfillment percentage is a maturity estimate because unknown or unstated legal requirements are less likely to be omitted as a higher percentage of legal subsections can be traced to some known, stated requirement. Subsection fulfillment (S_F) percentage represents the percentage of subsections to which some requirement maps in the highest-level sections to which the requirement under analysis maps. The highest-level section for a reference is the section with a depth of one. Consider the sample requirement in Figure 4.2. It has a mappings to subsections (a)(1) and (a)(2)(ii). Since (a) has a depth of one, it is the highest-level section for that reference. If our sample requirement had been mapped to both (a)(3) and (b), then S_F would be calculated based on both (a) and (b) since they each have a depth of one.

The S_F metric is calculated as follows: First, calculate the subsection count for the highest-

level subsections mapped by the requirement (SC_H). This represents the base number for the percentage. Note that this is not the same value as the S_C metric because it is calculated using the highest-level subsections rather than the more specific mappings. For our sample legal text, $SC_H = 9$ because our sample requirement has (a) as its highest-level subsection, and there are nine subsections under section (a).

Second, determine how many subsections are mapped to by other stated, known requirements in the highest-level sections mapped by the requirement under analysis (S_O). For our example, assume that we have other requirements that map to (a)(4), (b), and (c) in our sample legal text. This gives us a total of three other mapped subsections. However, the highest-level section mapped by the requirement is (a), so we ignore the requirements mapping to (b) and (c), which leaves us with a value of one.

Third, divide the total number of mapped sections ($S_M + S_O$) by the total number of subsections in the highest-level subsections to which the requirement maps (SC_H). This value is the subsection fulfillment percentage (S_F). For our sample requirement, the result is $\frac{2+1}{9} = \frac{1}{3}$.

Again, when combining metrics in an algorithm to determine legal implementation readiness, they must be normalized and weighted according to the particular legal requirements triage needs at the time of the triage. A requirements engineer seeking to identify those requirements stated as clearly as possible given the clarity of the legal text would heavily weigh the Maturity metrics. For the S_D metric, we simply divide by the deepest-level subsection found in any of the subsections mapped to the requirement. We do not need to normalize the S_F metric because it already has a maximum value of one.

4.2 Legal Requirements Triage Algorithm

Our example algorithm can be used to subdivide a requirements set into three subsets: (a) legal requirements that are implementation ready; (b) legal requirements that require further refinement; and (c) non-legal requirements. Applying our algorithm consists of four steps: (1) producing an XML-markup of the legal text; (2) entering requirements into our legal requirements management tool; (3) performing the legal requirements triage; and (4) performing a final pairwise prioritization. Our triage technique uses numeral assignment [36] and *k-means* clustering (also called Lloyd's algorithm) [44] to divide the legal requirements into two subsets: implementation ready and needing further refinement. The non-legal requirements comprise the third and final output requirements set. We employ example requirements from the iTrust Medical Records System to demonstrate the application of this algorithm [73, 74].

In the remainder of this Section we discuss the calculation of triage scores for each requirement in Subsection 4.2.1 and the use of *k-means* clustering to identify which requirements are implementable based on their triage scores in Subsection 4.2.2.

4.2.1 Calculation of Triage Scores

The purpose of our legal requirements triage algorithm is to calculate triage scores for each requirement and subdivide the requirements set so that developers can begin implementation quickly. In this section, we assign triage values to requirements based on metrics calculable from the mappings of the requirements to the legal text. We assign weightings to the metrics in our algorithm because the metrics produce relative values rather than absolute values. The weightings allow requirements engineers to interpret the scores more easily. We also normalize the metrics used in our calculation so that requirements engineers can more easily emphasize particular areas.

Consider our legal requirements metrics outlined in Table 4.1. Note that a lower triage score indicates a requirement that is more implementation ready and a higher triage score indicates a requirement needs further refinement. (This is why the maturity metrics are subtracted from the sum of the other two metrics.) Also note that we normalize the metrics and adjust the weightings to compensate for the number of metrics in each category.

$$T = D + C - M \tag{4.3}$$

$$D = \frac{D_W}{2} \cdot \left(\frac{S_M}{S_T} + \frac{C}{C_T} \right) \tag{4.4}$$

$$C = \frac{C_W}{4} \cdot \left(\frac{N_W}{W_T} + \frac{N_S}{ST_T} + \frac{S_C}{S_T} + \frac{E}{E_T} \right) \tag{4.5}$$

$$M = \frac{M_W}{2} \cdot \left(\frac{S_D}{SD_T} + S_F \right) \tag{4.6}$$

4.2.2 Identifying Implementation-Ready Requirements

After all the triage scores are calculated, we separate the legal requirements into three requirements sets: implementation ready, needing further refinement, and non-legal. First, we assign any requirement not mapped to a section of the legal text to the non-legal requirements set. Then, we perform the *k-means* clustering to group the remaining requirements into the implementation ready requirements set and the needing further refinement requirements set.

The *k-means* algorithm subdivides a set of values into k clusters iteratively based on values provided to the algorithm as an initial mean for each group. We have chosen not to perform a cluster analysis as a part of our algorithm because we are only interested in a single, clear situation: where there are two obviously distinct groups. Although a cluster analysis may reveal a more natural grouping with more clusters, unless a *2-means* analysis reveals a clearly distinct division into two groups, then our minimum standard for legal compliance cannot clearly be identified. Those results may mean that our requirements set does not lend itself to a textual interpretation of the requirements set against the legal text. This is a subjective risk

assessment built into this algorithm, which may have resulted in the limited results described in Chapter 5. Other researchers or engineers may choose to make different assumptions about the risk of legal non-compliance. However, for our purposes we set $k = 2$ to get two groups.

Since additional iterations improve the accuracy of group assignments in *k-means*, we use a n-iteration *2-means* algorithm, where n is the total number of legal requirements. In the first iteration, we select the highest triage score and the lowest triage score as our two initial means. Because these requirements are the maximum and minimum, we know that they will be in separate sets. Then, we randomly select a requirement from the remaining requirements in the set and assign it to the group with the closest mean. After assigning the requirement, we recalculate the mean for the group to which it was assigned. We continue assigning requirements to groups in this manner until all the requirements are assigned. This ends the first iteration. Starting with the second iteration, we begin by using the final means calculated in the previous iteration as the initial means for the two groups. Then, we randomly select a requirement and assign it to the group with the closest mean. Once again, we continue until all the requirements have been assigned.

Since the *2-means* algorithm will always produce two groups, even if all the requirements are actually implementation ready or all the requirements actually need further refinement, the final sets must be analyzed to determine if they are different enough to treat as distinct. To accomplish this, we calculate the standard deviation of the implementation ready requirements group. If the mean of the implementation ready requirements group is more than five standard deviations away from the mean of the group of requirements needing further refinement, then we accept the groups as distinct for the purposes of legal implementation readiness assessments. We use standard deviation as an acceptance criteria based our prior work where we found over five standard deviations of separation between the implementation ready requirements and those requiring further refinement [49]. However, since each development situation may have different time constraints, budgets, and resources, the final decisions to accept the groups as distinct and to accept one group as legally implementation ready may reasonably be made using other criteria.

4.2.3 Legal Requirements Triage Methodology

To support agile and iterative software development processes for systems that must comply with the law, software developers need to quickly identify requirements that are legally ready for implementation. To this end, we built a requirements management tool, called Red Wolf, that provides automated support for the legal requirements triage algorithm described in Section 4.2. Figure 4.3 is a screenshot of Red Wolf with six of the requirements from this case study.

Red Wolf Requirements Management

A product of The Privacy Place and North Carolina State University

Requirements | Users | Settings

Aaron: Edit Profile | Logout

Listing Requirements (showing 75 of 75)

Name	Description	User	Legal Subsection	
Requirement 1	iTrust shall allow an employee, using their authorized account, to create a new patient record by adding demographic information for the new patient.	Aaron	None.	Show Edit Destroy
Requirement 2	iTrust shall allow an employee, using their authorized account, to disable a patient record by marking that patient's record as disabled.	Aaron	None.	Show Edit Destroy
Requirement 3	iTrust shall allow an employee, using their authorized account, to destroy invalid patient records.	Aaron	None.	Show Edit Destroy
Requirement 4	iTrust shall email a patient when their account status is altered as per Requirements 1, 2 or 3 with a description of the alteration made.	Aaron	164.312(c)(2) and 164.312(e)(2)(i)	Show Edit Destroy
Requirement 5	iTrust shall email a personal representative when the account status of a patient they represent is altered as per Requirements 1, 2 or 3.	Aaron	164.312(c)(2) and 164.312(e)(2)(i)	Show Edit Destroy
Requirement 6	iTrust shall maintain a patient's records for a period of seven years after their record has been disabled.	Aaron	164.528(a)(1), 164.308(a)(7)(ii)(A), 164.528(a)(3), 164.528(b)(1), and 164.530(j)(2)	Show Edit Destroy

Figure 4.3: Red Wolf Requirements Management Screenshot

We now describe our four-step methodology for using Red Wolf for legal requirements triage. Figure 4.4 displays an overview of this methodology. Note that steps one and two can be done concurrently. In our case study, however, we produce an XML markup of the legal text first, and then enter the requirements into our legal requirements management tool second. Third, we choose weightings for the triage algorithm and conduct the triage. Fourth, we discuss options for pairwise prioritization of the final requirements groups.

Step One: Produce Legal XML

Requirements engineers must have some legal text to perform a legal requirements triage. In the United States, the Code of Federal Regulations (CFR) is the repository of legal regulations created and maintained by Executive Branch agencies of the federal government. These legal regulations are the detailed administrative laws to which individuals and businesses must directly comply. There are 50 titles in all representing broad areas such as agriculture, natural resources, commerce, transportation, and healthcare. The U.S. Government Printing Office

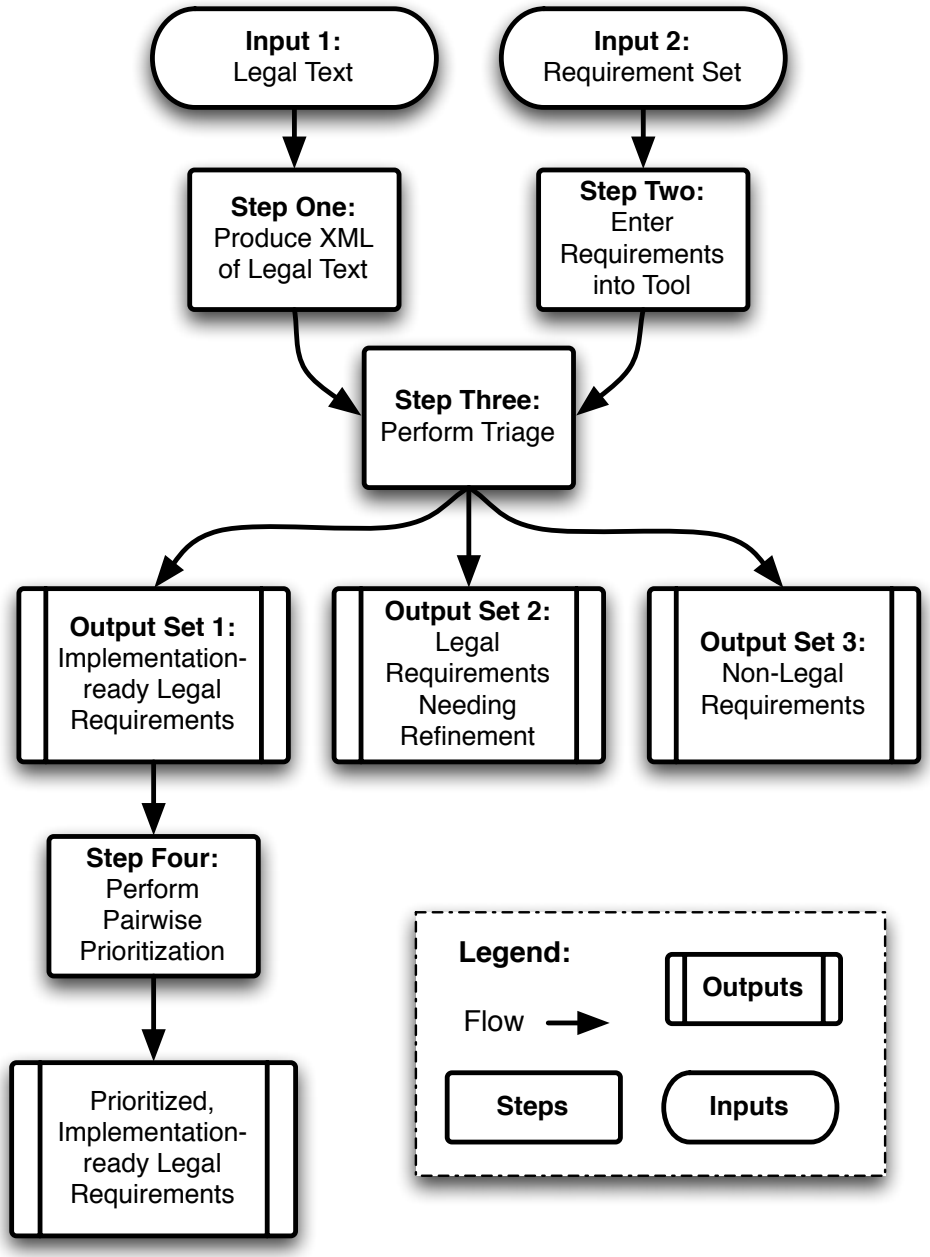

Figure 4.4: Legal Requirements Triage Methodology Overview

§ 164.312 Technical safeguards.

A covered entity must, in accordance with § 164.306:

(a)(1) *Standard: Access control.* Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in § 164.308(a)(4).

Figure 4.5: HIPAA §164.312(a)(1)

(GPO) hosts many regulations, including HIPAA, online. Figure 4.5 shows a sample section of HIPAA as it appears in on the GPO website[2].

To enable automated legal requirements triage, a requirements engineer must first produce an XML-formatted version of the legal text. We built tool support to partially automate the process of producing an XML-formatted version of HIPAA using the text files available for download on the GPO's website. This tool consists of a set of regular expressions generated by a manual translation of the legal text to our XML format. The requirements engineer must provide Red Wolf with the URL of the GPO page. When a regulation spans multiple text files, Red Wolf will merge them together into a single text file. Because the text files contain some HTML links (e.g. a link to the table of contents on every page), Red Wolf automatically removes these HTML tags, page numbers, and other bracketed notes. Unfortunately, Red Wolf does not yet complete the actual XML markup automatically. As a result, a requirements engineer must manually verify and, if necessary, complete the markup.

Other researchers have employed XML representations of legal texts [15, 12, 39]. Herein, we modified Breaux's XML format [15] in three ways. First, we gave unique names with semantic meaning to each hierarchy of the XML tags rather than using "div" for each of them. For example, subparts were labeled with the "subpart" tag, sections were labeled with the "section" tag and so on. Second, we decided to use the "ref" tag to denote a cross-reference to another law or another section within the HIPAA. Third, we denote exceptions using the "except" tag. Neither Breaux's nor Kerrigan and Law's XML formatting track cross-references or exceptions [15, 39]. Figure 4.6 depicts the same sample legal text found in Figure 4.5 using our XML markup format. Note that once a legal text has been marked up using our XML format requirements triage can be performed without manual intervention on every iteration.

[2] `http://www.access.gpo.gov/nara/cfr/waisidx_07/45cfr164_07.html`

Step Two: Enter Legal Requirements

Our legal requirements triage algorithm also requires a set of legal requirements that are mapped to (or have traceability links to) specific subsections of the legal text. In order to automatically perform the legal requirements triage, these requirements must be entered into Red Wolf. If a set of requirements that have been previously mapped to a legal text is not available, then a requirements engineer should use an existing technique to create this mapping [15, 17, 50, 52]. Legal requirements that would qualify as an input may be elicited through a variety of techniques. For example, the techniques for eliciting and specifying requirements outlined by Breaux et al. and Maxwell et al. are acceptable so long as traceability to the original legal text is maintained [15, 17, 52]. In addition, Massey et al. outline a procedure for mapping existing requirements to the relevant portions of a legal text [50]. Optionally, a requirements engineer may choose to use an automated traceability technique [12, 24] and manually complete the mapping to specific subsections of the legal text using one another technique [15, 17, 50, 52].

Our legal requirements management tool accepts natural language requirements with the following attributes: Name, Description, Legal Subsections, Origin, Context, and Rationale. These attributes were chosen because we used them in a prior study in which we elicited and validated legal requirements for the iTrust Medical Records System [50]. However, the only attributes required by Red Wolf are Name, Description, and Legal Subsections. The *Name* attribute is a short name that can identify the requirement. The *Description* attribute is a text block that represents the key functionality required. The *Legal Subsection* is a text string identifying the list of subsections to which the requirement maps in the law. Our legal requirements management tool parses this subsection to perform the requirements triage. Figure 4.7 displays the required attributes of a sample requirement from our case study.

The optional requirements attributes are Origin, Context, and Rationale. The *Origin* attribute is a string describing how the requirement was elicited. For example, "During consultation with a legal expert." The *Context* attribute further describes the situations or environments

```
<section id="164.312" title="Technical safeguards.">A covered entity must, in accordance
with <ref>Sec. 164.306</ref>:
  <section id="(a)">
    <section id="(1)" title="Standard: Access control.">Implement technical policies
    and procedures for electronic information systems that maintain electronic protected
    health information to allow access only to those persons or software programs that have
    been granted access rights as specified in <ref>Sec. 164.308(a)(4)</ref>.</section>
  Remaining text of Section 164.312(a).</section>
Remaining text of Section 164.312.</section>
```

Figure 4.6: Sample XML Markup for HIPAA §164.312(a)(1)

Name: Requirement 6

Description: iTrust shall maintain a patient's records for a period of seven years after their record has been disabled.

Legal Subsections: 164.528(a)(1), 164.308(a)(7)(ii)(A), 164.528(a)(3), 164.528(b)(1), and 164.530(j)(2)

Figure 4.7: Sample Case Study Requirement

when the requirement is used. For example, "This requirement is a part of the user registration process." The *Rationale* attribute describes the reasons the requirement is needed. For example, "The customer wants to make sure that log files are backed up on an hourly basis." These attributes can be useful when conducting a pairwise prioritization in Step Four.

Red Wolf automatically assigns and updates the following attributes to requirements as they are processed: ID, User, Created At Date, Updated At Date, and Priority. The *ID* attribute is a unique number that can also be used to identify the requirement. The *User* attribute is the username of the individual who added the requirement to the system. The *Created At Date* and *Updated At Date* attributes denote the date and time the requirement was created and last updated respectively. The *Priority* attribute is a text string identifying the priority group of the requirement. This attribute has four possible values: Unanalyzed, Non-Legal, Not Implementation Ready, and Implementation Ready. The Unanalyzed value denotes a requirement that has not yet been processed by the requirements triage system. The Non-Legal value is assigned to requirements that do not map to legal subsections and cannot be triaged using our algorithm. The Not Implementation Ready value denotes a legal requirement that has been triaged and found to be not yet ready for implementation. The Implementation Ready value denotes a legal requirement that has been triaged and found to be ready for direct implementation or further prioritization with a pairwise prioritization technique.

Step Three: Execute Requirements Triage

Once the XML-formatted legal text and the requirements are entered into Red Wolf, requirements engineers must set the weightings described in Section 4.2.1 to perform the requirements triage. Red Wolf has a settings page on which requirements engineers can establish both the

weightings for the categories of requirements and the version of the XML-formatted legal text to use for the triage. Once set, the requirements engineer simply clicks a button to perform the triage.

When Red Wolf completes the triage, it displays a summary screen. The summary screen shows which requirements the tool found to be in each of the three groups (implementation ready, needing refinement, and non-legal) along with the average triage score and standard deviation for each of the *k-means* clusters. Red Wolf also indicates if the mean value of implementation ready requirements set is five standard deviations away from the mean value of the set of requirements needing further refinement.

Step Four: Perform Prioritization

After the automated triage, requirements engineers must still prioritize the implementation ready requirements according to software engineering standards. Because our triage technique focuses on legal implementation readiness, software concerns must be taken into account separately. Numerous pairwise prioritization techniques, including AHP, could fulfill this step [27, 33, 36, 37, 38, 43, 61, 64, 7].

4.2.4 Triage Results

In this section, we discuss the results of an experiential case study in which we applied our requirements triage technique to 75 functional requirements for the iTrust Medical Records System, an open-source EHR system designed by students at North Carolina State University over 11 semester-long courses for use in the United States healthcare industry. The initial iTrust developers expressed their requirements for iTrust as Unified Modeling Language use cases in consultation with both a practicing physician and a professional from the North Carolina Healthcare Information and Communications Alliance. Because iTrust is designed for eventual real-world use, it must comply with HIPAA.

Massey et al. previously evaluated the iTrust requirements for legal compliance, and provided an initial mapping of iTrust requirements to the HIPAA regulations [50]. This evaluation produced a total of 73 requirements, of which 63 were functional and 10 were non-functional [50]. To these 63 functional requirements we add 12 additional functional requirements for iTrust identified by Maxwell and Antón [52]. These 75 functional requirements and the text of HIPAA, to which these requirements must comply, form the inputs for this case study.

We chose to use a Dependency weighting (D_W) of 30, a Complexity weighting (C_W) of 30, and a Maturity weighting (M_W) of 60 to ensure a larger magnitude of triage scores. In addition, the algorithm is structured that ratios of 1 : 2 for $D_W : M_W$ and $C_W : M_W$ will keep relatively

Table 4.2: Triage Results ($D = 30, C = 30, M = 60$)

Set	Requirements	Mean	Std. Dev.
Implementation Ready	46	5.73	0.76
Needing Refinement	12	24.4	2.7
Non-Legal	17	N/A	N/A

good triage scores near zero. Using these weightings, Red Wolf produced 46 implementation ready requirements as shown in Table 4.2. The number of pairwise comparisons for 46 requirements is 1,035. In contrast, a pairwise prioritization technique that produce a total ordering of 75 initial requirements would require 2,775 comparisons. Thus, our legal requirements triage algorithm reduced the number of pairwise comparisons needed to prioritize implementation ready legal requirements to 37.3% when compared to a complete prioritization.

The non-legal requirements set produced is not analyzed as a part of the triage and they may be implementation ready. Therefore, requirements engineers may reasonably desire to include the non-legal requirements in their post-triage, pairwise prioritization of requirements. In our case study, we would need to include the 17 non-legal requirements for such a pairwise prioritization. This results in 1,953 pairwise comparisons, which is about 70.4% of the number of pairwise comparisons needed for all 75 requirements.

4.3 Chapter Summary

This chapter introduces and defines our legal requirements metrics. These metrics were designed to function as a part of a legal requirements triage process by quickly determining from a set of requirements traced to subsections of a legal text to which they must comply which requirements were LIR and which were not. This chapter also discusses an example algorithm that uses legal requirements metrics as a part of a legal requirements triage process.

The legal requirements metrics discussed in this chapter were not designed to replace legal domain experts in a legal requirements analysis session. Instead, they were designed for a legal requirements triage process, which allows both engineers and legal domain experts to quickly assess the particular legal issues pertinent to an existing set of requirements. In the next chapter, we examine how software engineers assess software requirements for legal implementation readiness and compare their assessments with those created using the legal requirements metrics detailed in this chapter.

Chapter 5

Validation Studies

"Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius—and a lot of courage—to move in the opposite direction."

–Albert Einstein

In this chapter, we discuss three validation studies for the legal requirements metrics developed and defined in Chapter 4. Each study seeks is designed to learn more about the nature of assessing software requirements for legal compliance. Specifically, we are interested in the processes by which software engineers make determinations about whether a requirement meets or exceeds its legal obligations. A requirement is considered to be Legally Implementation Ready (LIR) if it meets or exceeds its legal obligations as expressed in relevant regulations. Each case study in this chapter employs a set of Electronic Health Records (EHR) system requirements, coupled with a section of the U.S. Health Insurance Portability and Accountability Act (HIPAA) to which the system must comply.

In our first study, the LIR Assessment study, we performed an experiment in which we asked graduate-level software engineering students to determine whether a set of software requirements for an EHR system met or exceeded their corresponding legal obligations as expressed in regulations created pursuant to the U.S. Health Insurance Portability and Accountability Act (HIPAA). We compare the determinations made by graduate students with the findings of three subject matter experts—individuals with both software engineering and HIPAA compliance expertise—who evaluated the same set of requirements for legal implementation readiness. In addition, we contrast these results with those generated by our legal requirements triage algorithm, which uses legal requirements metrics to assess the same set of requirements for legal implementation readiness as described in Section 4.2 of Chapter 4. We also created a statistical model using our legal requirements metrics to determine whether

or not they aid legal compliance decisions.

Our LIR Assessment case study (see Section 5.1) reveals that graduate-level computer science students exhibit little consensus about LIR requirements, and they poorly identified which requirements were LIR. Our first study also validates the need for subject matter experts or additional guidance for individuals making legal compliance decisions. Finally, our first study reveals that the metrics used in our legal requirements triage algorithm are useful in identifying LIR requirements; the strengths and limitations of the metrics and algorithm are presented in Subsection 5.1.6.

Our Replication case study (see Section 5.2) replicates and expands elements of the first study to address two concerns raised by our first study. First, we sought to confirm that graduate-level software engineers are not able to achieve consensus about whether software requirements are LIR. Second, we examine whether additional consensus cutoffs would improve the accuracy of LIR assessments conducted individually. Our findings suggest that our first study was correct: the participants were once again unable to achieve consensus on their assessment regarding whether the requirements were LIR. In addition, we were able to determine that voting cutoffs play a significant role in determining the accuracy of assessments.

In our Wideband Delphi case study (see Section 5.3), we used the Wideband Delphi estimation method to derive consensus among 13 graduate-level software engineers about whether the requirements examined in our previous two studies were LIR. We conducted this study to determine whether graduate-level software engineers are able to improve their assessments when they are forced to come to a consensus. Our findings suggest that our participants were able to slightly improve the accuracy of their LIR assessments, but their assessments were much more conservative than the experts, which would likely result in over-engineering of the underlying system.

5.1 LIR Assessment Case Study

The experiment design for our LIR Assessment case study employs the Goal/Question/Metric (GQM) approach [9, 10], which is based on the idea that measurement is useful for making intelligent decisions and improving those decisions over time, but that measurement must be focused, and based on goals and models. The GQM approach proposes that every measurement activity have one or more goals. Each goal must then have at least one question that helps achieve the goal. Finally, each question is answered using one or more measures or metrics. Our research goal as formulated using the GQM template is as follows:

Goal: Analyze empirical observations for the purpose of characterizing legal imple-

mentation readiness with respect to software requirements from the viewpoint of software engineers in the context of an EHR system that must comply with HIPAA regulations.

Given this research goal, we formulate the following six research questions:

Q1 Is there a consensus among subject matter experts on which requirements are LIR?

Q2 Is there a consensus among graduate students on which requirements are LIR?

Q3 Can graduate students accurately[1] assess which requirements are LIR?

Q4 Can we predict which requirements are LIR using attributes of those requirements?

Q5 How do the categories for the legal requirements metrics affect whether a given requirement is LIR? In other words, when the metrics are grouped in the categories defined by our legal requirements triage algorithm, do they remain valid measures of whether or not a requirement is LIR?

Q6 Can we use our legal requirements triage algorithm for automating the process of predicting whether a requirement is LIR?

In Subsection 5.1.1, we describe the materials for our research experiment. In Subsection 5.1.2, we discuss our participant population. In Subsection 5.1.3, we describe how we created our canonical set of correct responses by achieving consensus among experts. In Subsection 5.1.4, we describe how we used our legal requirements triage algorithm and the legal requirements metrics from which it is built. In Subsection 5.1.5, we discuss our analysis methodology and the statistical tools we used in our experiment. We introduce our measures for answering our six research questions in Subsection 5.1.6.

5.1.1 Materials

Each experiment group (described in Sections 5.1.2 and 5.1.3 below) received three inputs:

a. A sample legal text

b. A requirements specification that includes a glossary of terms

c. A traceability mapping of the individual requirements to the legal text.

[1]Objectively accurate assessment of LIR requirements is impossible due to the inherently subjective nature of interpreting laws and regulations. Thus, we compare against a canonical set of requirements created by experts as described in Subsection 5.1.3

The legal text chosen for this experiment is HIPAA § 164.312, which governs technical safeguards. We selected this section for three reasons: First, it is a part of HIPAA, with which we have extensive experience [17, 54, 47, 49, 52, 53]. Second, it is focused on technical measures for protecting healthcare information, which is something that software engineers are more likely to be familiar with than a less technical HIPAA section. If there is a gap between the results from the experts' canonical classification and that of the graduate students, then we can reasonably infer that this gap would only widen if we were to perform the experiment again on more ambiguous or less technically oriented HIPAA sections. Third, it is a short, self-contained section which allowed us to test it in its entirety rather than exerpting and testing a longer HIPAA section.

The requirements used for this experiment are from the iTrust Medical Record System requirements specification, an open source EHR system designed by students at North Carolina State University. This system was discussed previously in Chapter 3. The iTrust system shares many characteristics with other existing software systems that must comply with new laws and regulations. Such systems must be evaluated, updated, and improved in order to achieve compliance with each enactment or amendment of relevant laws and regulations.

For purposes of our experiment, we modified the iTrust requirements specification. Instead of including all 75 iTrust system requirements, we started with the 15 requirements with legal obligations traced within HIPAA § 164.312. Then we iteratively applied the methodology for evaluating requirements for legal compliance outlined in Chapter 3 to the other iTrust system requirements. After three iterations, we identified an additional 17 requirements for inclusion in our experiment. However, we chose not to cover all elements of HIPAA § 164.312, and we also included some requirements that only partially covered the elements of the legal text to ensure that some requirements were explicitly not LIR. Additionally, we modified our selected requirements to remove terminology and references to other aspects of iTrust not relevant to HIPAA § 164.312. Any remaining terminology was provided to the participants in the form of a glossary.

Our goal in creating this document was not to create a set of requirements that were already perfectly aligned with the law; instead, we sought to examine a variety of legal compliance situations ranging from relatively easy to relatively challenging decisions. We selected one of our 32 requirements to be used as a part of the tutorial for providing basic training to participants about how software engineers may reason about legal compliance. As a result, we had 31 remaining requirements to use in our experiment.

The materials also included a traceability mapping of the 31 selected requirements to the specific subsection(s) of HIPAA § 164.312 to which they applied. We constructed the traceability links iteratively using the techniques outlined in Chapter 3. In addition, all of the materials outlined in this Subsection are provided in full in Appendix A for the interested

reader.

5.1.2 Software Engineering Participants

The participants in our experiment are computer science graduate students who have taken or are taking the graduate-level software engineering course at North Carolina State University. Through their coursework, we know that they are familiar with the basic practices and responsibilities of a software engineer working as a part of a larger team. For this case study, which was conducted during the Spring semester of 2011, we had 32 participants.

Before participating in the experiment, all participants received lessons in requirements engineering and a brief tutorial on legal compliance in software systems, which can be found in Appendix A at Section A.1. This tutorial introduced some basic concepts in legal compliance for software requirements. It consisted of an explanation to the importance of legal compliance, as well as an explanation of the traceability mapping of requirements to the legal text and how that mapping might be useful in evaluating legal compliance. The tutorial included the example legal compliance scenario outlined in Figure 5.1.

After explaining this example to the participants, we verbally described another potentially conflicting requirement that is a variation on Requirement B. We asked the participants to consider whether Requirement B would become LIR if the phrase "so long as the user ID remains unique" was added. After allowing the participants to consider this for a few moments, we showed that it would be possible for a user to adopt a previously discarded user ID as their own. In this situation, a user could change their ID to something totally unique, but their old ID would then become available for another user to use as their ID. This could result in access logs that have a single user ID that represents two separate users, which would be a clear violation of the legal obligations stated.

5.1.3 Subject Matter Experts

Canonical sets of LIR requirements can be used to check other requirements sets for legal compliance [48]. To generate a canonical set of LIR requirements, we recruited three subject matter experts: Paul Otto, Dr. Annie Antón, and the author of this dissertation. All three experts have varying years of academic experience with HIPAA legal compliance in software engineering; additionally, one expert (Paul Otto) has a law degree.

We employed the Wideband Delphi method [14] to determine the consensus subject matter expert opinion to identify the LIR requirements and those requirements needing further refinement. First, we gave each expert the materials described in Subsection 5.1.1. Next, we asked the experts to individually identify both the LIR requirements and those needing further refinement, recording their rationale for each decision. This individual analysis was used to

Consider Requirement A:

Requirement A: iTrust shall generate a unique user ID and default password upon account creation by a system administrator. [Traces to §164.312(a)(1) and §164.312(a)(2)(i)]

For reference, here are the two subsections of the legal text to which Requirement A is traced:

(a) (1) Standard: Access control. Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in § 164.308(a)(4).

(2) Implementation specifications: (i) Unique user identification (Required). Assign a unique name and/or number for identifying and tracking user identity.

Requirement A meets or exceeds its legal obligations outlined in §164.312 because no element of the regulation related to the requirement describes different or additional obligations than those described in the requirement.

In contrast, consider Requirement B:

Requirement B: iTrust shall allow an authenticated user to change their user ID and password. [Traces to §164.312(a)(1) and §164.312(a)(2)(i)]

Note that Requirement B traces to the same two subsections of legal text as Requirement A. Requirement B does not meet or exceed its legal obligations outlined in §164.312 because it describes a situation where it is possible for users to end up with the same identifying name or number, which violates §164.312(a)(2)(i).

Figure 5.1: Example Legal Compliance Scenario from Case Study Tutorial

address our first research question as described in Section 5.1.6. Third, we held a coordination meeting in which the experts discussed areas of disagreement and worked to arrive at a consensus opinion for each requirement. Their final, consensus opinions serve as the canonical LIR requirements set against which we compare responses from the graduate-level computer science participants in both the first and second case studies and the legal requirements triage algorithm.

5.1.4 Legal Requirements Triage Algorithm

In addition to our graduate-level software engineering participants, we examined the results from our legal requirements triage algorithm as a part of our first case study. Our legal requirements triage algorithm uses legal requirements metrics to classify requirements as either LIR or needing further refinement [49, 54]. The legal requirements metrics are composed of simple, consistent attributes of a legal text and a set of requirements that must comply with that legal text, such as the number of sections or subsections within a particular regulation. These metrics and the algorithm used to generate a relative score for each requirement are defined in detail in Chapter 4.

We implemented our legal requirements triage algorithm in Red Wolf [49, 54], a requirements management tool we developed for the purposes of automating the classification of requirements as either LIR or needing further refinement. Because our algorithm requires the legal text, a set of requirements, and a traceability mapping between the requirements and the legal text, it had the same inputs as the materials given to both the graduate-level software engineering students and the group of experts and practitioners.

5.1.5 Analysis Methodology

Weka[2] is a tool used for modeling and data mining. We employed Weka to produce a logistic regression model [34] to predict whether or not each requirement is LIR. Weka performs a regression fit on the data using the consensus expert results from the Wideband Delphi study about which requirements are LIR and produces coefficients for the logistic regression model. When we substitute values for a given requirement's dependency, complexity, and maturity into the model, the model suggests the likelihood that the requirement is LIR. Weka uses this likelihood to produce a threshold at which a requirement is considered to be LIR. Weka uses this threshold to produce a result that we compare to the canonical set of LIR requirements created by the subject matter experts, which provides values that we use to calculate specificity, sensitivity, and a Kappa statistic for the model. We used the R Project[3], a statistical modeling

[2] `http://www.cs.waikato.ac.nz/ml/weka`
[3] `http://www.r-project.org/`

package, to calculate Fleiss Kappa statistics for the data in our study.

All of our logistic regression prediction models were built using 10-fold cross validation, which is a method for simulating a prediction scenario using smaller amounts of data. In 10-fold cross validation, the original sample is partitioned into 10 subsamples [56]. The model is then trained on 9 of the subsamples, and tested on the remaining subsample. This process is repeated 10 times for each division of subsamples. The results can then be averaged to reach a single estimation of the model's performance.

We formed a consensus using Wideband Delphi [14] for the three subject matter experts. In Wideband Delphi, consensus is formed in rounds with discussion at the end of each round. First, each participant provides answers independently and records their rationale for the answers they have chosen. Next, participants share their answers and reconsider their positions. Then, participants meet to achieve a final consensus. Although we completed the Wideband Delphi technique to produce our canonical set of LIR requirements, we also chose to analyze the results of our independently recorded first analysis to see what level of agreement existed among the subject matter experts at the beginning of the process.

5.1.6 LIR Assessment Case Study Results

First, we describe reactions and questions from our participants. Next, we discuss some of the points of disagreement found during our consensus meeting for the subject matter experts. Then, we analyze the data according to our analysis methodology.

Participant Experiences

We conducted our experiment with 32 graduate students. The majority (21) of these participants completed the experiment in the same room at the same time. The remaining participants completed the experiment individually or in small groups over the course of the next week. Participants were trained using the same training materials, which took about five minutes to introduce and explain at the beginning of the experiment. Each participant was then given 45 minutes to analyze 31 requirements and determine whether each requirement was LIR or not. Training was conducted in a group setting when possible, but all participants worked on their own during the 45-minute analysis portion of the experiment.

We answered clarifying questions during each session, and for each question asked we repeated the question and the answer for every participant. Because the first session was the largest group to participate, we assembled all of the questions that could not be answered by referring to other material within the requirements specification or legal text provided to the participants; we then provided those questions and corresponding answers to subsequent groups as part of their training material. For example, a question about the definition of

an invalid employee account was not repeated because it is listed as a term in the glossary provided. However, a question about whether or not an email would be considered part of an access control log (answer: they are not) was included in subsequent training material.

We did not answer substantive questions about legal compliance or interpretation of the software requirements. We instructed the participants that these were the sorts of questions the study was designed for them to consider when making their own assessment of the legal implementation readiness of the requirements. Although such a response provided the participants with some information, namely that the question asked was indeed substantive, we felt our only alternative would have been to forbid any questions about the study materials. Such an approach would not have allowed us to respond to basic clarifying questions about the materials and procedure. Refusing to answer substantive questions about legal compliance or interpretation of the software requirements, and thus indicating that the question was indeed substantive, was an acceptable tradeoff for us to ensure participants fully understood their task.

Subject Matter Expert Discussion

As described in Section 5.1.3, our subject matter experts entered the Wideband Delphi consensus meeting already having achieved a moderate level of consensus based on their individual responses. However, there were still 12 out of 31 requirements for which the subject matter experts did not achieve universal agreement after the first round of Wideband Delphi.

In some cases, two subject matter experts immediately accepted the rational used by one subject matter expert to denote a non-LIR requirement. Consider Requirement 18, as described in Figure 5.2.

Note that HIPAA § 164.312(b) reads as follows:

> **Standard: Audit controls.** Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

In other cases, the subject matter experts took some time to debate whether or not a

> **R-18:** Whenever a new employee account is created, iTrust shall create a record of the employee account created and the system administrator who created it in the access control log. [Traces to §164.312(b)]

Figure 5.2: iTrust Requirement 18

R-26: Each time a printable emergency report for a medical record is generated, iTrust shall email a notification to the patient to whom the record pertains and to all of that patient's personal representatives. [Traces to §164.312(b)]

Figure 5.3: iTrust Requirement 26

requirement should be considered LIR. In iTrust, any employee is allowed to generate a printable summary of a patient's medical record for the purposes of handling emergency medical scenarios. Part of this functionality is described by Requirement 26, shown in Figure 5.3.

This requirement traces to the same section of HIPAA as our previous example, but the discussion of whether or not it is a LIR requirement is more nuanced. The experts debated several considerations: First, they determined that email notification does not qualify as a record in an access control log. Since R-25 describes the creation of such a record in the exact same printable emergency report scenario, R-26 could not be found in need of this refinement. Second, the experts debated whether or not an email notification qualified as a "procedural mechanism that records or examines activity" as prescribed by HIPAA § 164.312(b). On one hand, email notification may allow a patient some oversight through the ability to examine the activity of generating a printable emergency report. On the other hand, email notification does not ensure that a patient will examine the activity. This second position is a broader interpretation of HIPAA § 164.312(b), and the experts agreed that assuming the broader standard was the safer course of action to ensure legal compliance. Third, the subject matter experts determined that R-26 should not be traced to HIPAA § 164.312(b) because notification is a separate action from either 'recording' or 'examining activity.' Ultimately, the experts reached consensus that R-26 is LIR because it does not describe any action that violates any legal obligation.

Data Analysis

To analyze our results, we built a statistical model of the subjects' results, the experts' results, and the algorithm's results. These predictive models allow us to compare the results from each group. Since any predictive model for classifying requirements as LIR or not LIR will either do so either correctly or incorrectly, we can compare the rates of correct responses versus the rates of incorrect responses. For a given test of each model, some responses will be true positives, where the model correctly classifies a requirement as LIR, and other responses will be true negatives, where the model correctly classifies the requirement as not LIR. When the model is wrong, some responses will be false positives, where the model classifies the requirement as

being LIR but the requirement was not LIR, and other responses will be false negatives, where the model failed to identify an LIR requirement.

The performance of a given model to classify a requirement as being one of two binary options is often evaluated using two measurements: sensitivity and specificity [59]. Sensitivity measures the ability of a given model to predict positives. In the case of LIR, this is a weighted measurement of how many LIR requirements the model classifies as not LIR. From a software engineering standpoint, this situation describes a scenario in which requirements that already meet or exceed their legal obligations are considered for further refinement. Sensitivity is defined in Equation 5.1, where tp is the number of true positives and fn is the number of false negatives.

$$Sensitivity = \frac{tp}{tp + fn} \tag{5.1}$$

Specificity measures the ability of a given model to predict negatives. In the case of LIR, this is a weighted measurement of how many non-LIR requirements the model classifies as LIR. From a legal compliance standpoint, such a classification presents a dangerous situation because it may result in requirements that have not met or exceeded their legal obligations being implemented. Specificity is defined in Equation 5.2, where tn is the number of true negatives and fp is the number of false positives.

$$Specificity = \frac{tn}{tn + fp} \tag{5.2}$$

The Fleiss Kappa statistic [35], κ, measures level of agreement between raters on a given set of subjects. We employ the Fleiss Kappa statistic to calculate the level of agreement between subjects' determinations about whether a requirement is LIR or not. The Fleiss Kappa statistic ranges from –1 to 1. The value 1 reflects perfect agreement between raters, the value –1 indicates perfect disagreement, and the value 0 indicates the amount of agreement that would be expected by random chance. Every value between 0 and 1 reflects some level of agreement, with larger values indicating more agreement.

We now discuss the results of our experiment for each of our research questions identified in Section 5.1.

Q1. Is there a consensus among subject matter experts on which requirements are LIR?

Measure: Fleiss Kappa statistic for three subject matter experts' results requirements.

The Fleiss Kappa statistic for our three experts was $\kappa = 0.517$ ($p < 0.0001$). This result indicates that with a high level of statistical significance, our experts moderately agreed on their first assessment of the 31 requirements before completing the Wideband Delphi session

to reach consensus.

Answer: We can say that there is a moderate consensus among experts regarding LIR.

Q2. Is there a consensus among graduate students on which requirements are LIR?

Measure: Fleiss Kappa statistic for 32 graduate students on 31 requirements.

The Fleiss Kappa statistic for our 32 students was $\kappa = 0.0792$ ($p < 0.0001$). This result indicates that with a high level of statistical significance, the students in our case study had only a slight agreement on their assessment of the 31 requirements. Because Fleiss Kappa accounts for random chance, this level of agreement is only slightly better than what would be expected from a random set of responses. Due to the high level of significance in both Kappa scores for Q1 and Q2, we can say that there was a higher level of consensus among experts than students.

Figure 5.4 displays the responses for the students in the LIR Assessment study on the top bar graph, labeled "Spring 2011 Study". (The bottom bar graph, labeled "Fall 2011 Study," displays the data from our replication study, which is discussed in Section 5.2.) The vertical axis represents the number of students. Students that indicated a requirements is LIR (i.e. a 'Yes' response) with a blue bar or not LIR (i.e. a 'No' response) with a red bar. The horizontal axis represents the requirement for which a decision was made. Note that every requirement received some 'Yes' responses and some 'No' responses. Also, the majority of requirements have more 'Yes' responses than 'No' responses.

Answer: We can say that there is little consensus among graduate students regarding LIR.

Q3. Can graduate students correctly assess which requirements are LIR?

Measures: Sensitivity, specificity, and percent agreement between students' and experts' results for 31 requirements.

To answer Q3, we formed a consensus among the 32 students' votes on whether each of the 31 requirements was LIR or not to create our predictive model (RMSE = 0.52; AIC = 46.62). We used a simple majority to form consensus between the students: if more than 50% of the students said that a given requirement was LIR, the consensus was that the requirement was LIR. Otherwise, the consensus was that the requirement was not LIR. We used these consensus values to determine whether students were more likely to identify a requirement as LIR when it was not LIR (i.e. a false positive or a Type 1 error) than they were to identify a requirement as not LIR when it was LIR (i.e. a false negative or a Type 2 error).

Using this technique, the students' consensus had sensitivity of 0.875 and specificity of 0.20. A sensitivity of 0.875 indicates that students had a low false negative rate; when they said

a requirement needed further refinement, they were more likely to be correct. A specificity of 0.20 indicates that the students had a high false positive rate; when they said a requirement was LIR, they were more likely to be incorrect. This is a serious concern from a legal compliance standpoint; implementing a requirement that has legal compliance concerns may result in expensive efforts to refactor the resulting system or worse, a violation of the law. Overall, the students agreed with the experts for only 54.8% of the requirements. These values mean that the students were better at predicting a requirement as LIR and likely to miss a requirement that is not LIR.

Answer: We can say that graduate students cannot accurately assess the LIR status of a requirement and are more likely to miss requirements that are not LIR.

Q4. Can we predict which requirements are LIR using attributes of those requirements?

Measures: Sensitivity, specificity, and Fleiss Kappa statistic for a statistical model built based on the legal requirements triage algorithm's results for 31 requirements.

We used our legal requirements metrics to produce a logistic regression model (RMSE = 0.54; AIC = 39.17) that predicts whether or not requirement is LIR. Using this model to predict the experts' opinions, we achieved sensitivity of 0.625 and specificity of 0.80, with $\kappa = 0.35$ ($p < 0.0001$). These values mean that the model was more likely to miss a requirement that was LIR, sending a requirement to experts for evaluation that required little to no more evaluation, than to say a requirement was LIR that was not. The model also had a higher specificity than the students, meaning it was more likely to catch requirements that were not LIR than the students. The Fleiss Kappa score for this comparison also indicates a fair agreement between the model and the experts, with the model agreeing with the experts more than the students do.

Answer: We can say that our legal requirements metrics can be used to make a model that is useful in predicting whether a requirement is LIR status.

Q5. How do the categories for the legal requirements metrics affect whether a given requirement is LIR? In other words, when the metrics are grouped in the categories defined by our legal requirements triage algorithm, do they remain valid measures of whether or not a requirement is LIR?

Measures: Complexity, Maturity, and Dependency for 31 requirements.

As described in Section 5.1.5, Weka fits a logistic regression function to the dataset to create the model (RMSE = 0.54; AIC = 46.07) we used to predict the LIR status of each requirement. The coefficients of this resultant logistic regression function tell us how each metric affects the

Table 5.1: Model Coefficient Signs

Term	**Coefficient Sign**
Dependency	Negative
Complexity	Negative
Maturity	Positive

probability that a requirement is LIR for this dataset.

If the coefficient for the metric's term in the model is negative, higher values of the metric mean it is less likely that metric is LIR. If the coefficient for the metric's term in the model is positive, higher values of the metric mean it is more likely that the metric is LIR. Table 5.1 presents the signs for the coefficients in our model.

Answer: These results indicate that Dependency and Complexity make the metric less likely to be LIR, and Maturity makes the metric more likely to be LIR.

Q6. Can we use our legal requirements triage algorithm for automating the process of predicting whether a requirement is LIR?

Measures: Sensitivity, specificity, and Fleiss Kappa statistic for algorithm and experts on 31 requirements.

To answer Q6, we executed the algorithm on the 31 requirements and obtained its classifications for whether each requirement was LIR or not. We used these values to measure the ability of the algorithm to accurately predict (RMSE = 0.52; AIC = 46.91) whether the requirement was LIR, using the experts' consensus as an oracle. Using this technique, the algorithm had sensitivity of 0.5 and specificity of 0.466, with $\kappa = -0.03$ ($p < 0.0001$). These values indicate that the algorithm was slightly better at predicting requirements that were LIR than it was at predicting requirements that were not LIR. The Fleiss Kappa value indicates that the model often disagreed with the expert opinion. However, the algorithm was better at predicting requirements that were not LIR than the students, although not as good at predicting either LIR requirements or not LIR requirements as the subject matter experts.

Answer: We can say that the algorithm is not useful for predicting LIR in its current form; however, the algorithm would be less likely to miss a requirement that is not LIR than the students.

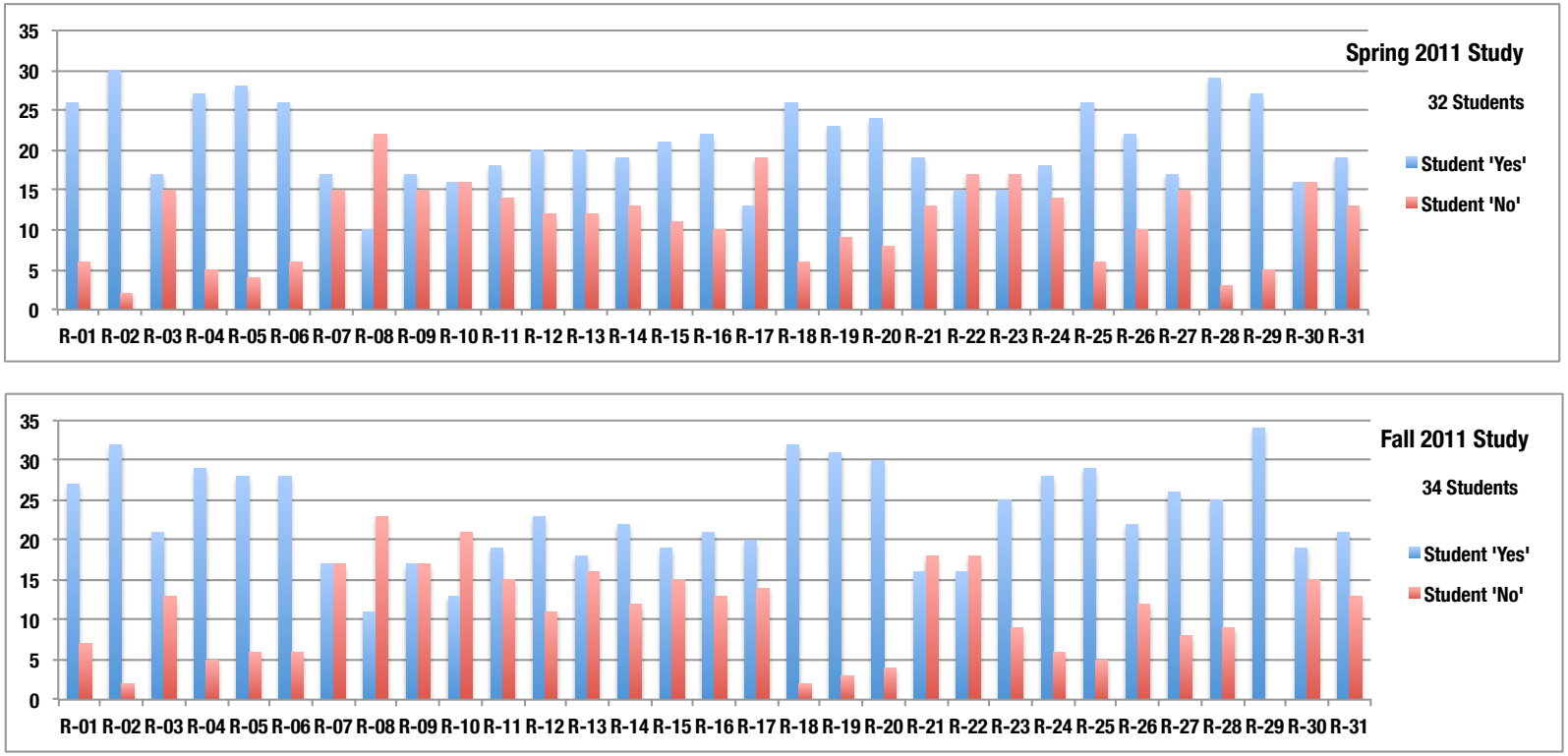

Figure 5.4: Student Responses to the Legal Implementation Readiness Studies of Spring 2011 and Fall 2011

5.1.7 LIR Assessment Case Study Threats to Validity

Construct validity refers to the appropriateness and accuracy of the measures and metrics used for the concepts studied [75]. The primary measure we use is the canonical set of LIR requirements generated by the subject matter experts using a Wideband Delphi procedure. The use of a canonical set of LIR requirements to test other requirements for legal compliance is an accepted practice [48]. In addition, Wideband Delphi is an accepted practice for achieving consensus in software engineering [14]. However, the three subject matter experts worked together previously on the concept of LIR requirements. As a result, they are not entirely independent individuals despite conducting the first phase of our Wideband Delphi consensus procedure independently from one another. In addition, our subject matter experts have not developed a commercial EHR system.

Another construct validity concern is the choice of statistical tools and modeling techniques. Our choice of sensitivity and specificity was based on the fact that a false negative (identifying a requirement as needing refinement when it is actually LIR) has a low penalty whereas a false positive (identifying a requirement as LIR when it really needs further refinement to meet its legal obligations) has a high penalty. Precision and recall [59] are similar statistical tools that may be better estimators, but these statistics treat false positives and false negatives as equally problematic.

A modeling technique other than logistic regression might perform better at predicting whether or not requirements are LIR. We considered many alternative modeling techniques and used Weka to compare the sensitivity and specificity for each technique. Based on these scores, Weka reveals that logistic regression modeling consistently outperform the other choices available in the Weka modeling toolkit. A discussion comparing logistic regression to other prediction modeling techniques is outside the scope of this paper.

The answers we provided to participants' questions during the experiment are another source of potential construct validity concerns. For example, one participant asked about the difference between Required and Addressable as used in the legal text. We explained that Required indicates a legal obligation to implement all elements described in that subsection as closely as possible, whereas Addressable indicates that the general concept must be present but does leave some implementation details up to the specific software engineer. Our explanation for Addressable, however, was an inadvertent misrepresentation of the term's precise legal meaning, provided in HIPAA § 164.306(d)(3). Addressable subsections allow entities to assess whether the subsection's requirements are "reasonable and appropriate" for their systems; if not, then entities may instead implement equivalent measures that are in fact "reasonable and appropriate" for their system, or explain why no measures are required. Our misrepresentation was not inconsistent with the legal text, and our explanation implied that all Addressable

subsections had been predetermined to be "reasonable and appropriate" for implementation in iTrust.

External validity refers to the ability to generalize findings and results to other domains [75]. By selecting a highly technical subsection of HIPAA that should have played to the technical background of our participants, we believe that we have established a valid baseline from which we can generalize to other sections of HIPAA or other pieces of legislation. In other words, because our participants exhibited little consensus when examining a piece of legislation with an emphasis on technology standards, we believe that they would exhibit even less consensus on a less technical and more legally oriented piece of legislation. In addition, we plan to conduct additional studies on broader sections of HIPAA.

Another external validity concern is that graduate students may perform differently if they were in a genuine legal compliance scenario. For purposes of this experiment, we had to limit the time and resources that our participants were allowed to use while reviewing the requirements for legal compliance.

Reliability refers to the ability of other researchers to repeat an experiment [75]. We assiduously documented our process and procedures while conducting our experiment. In addition, we can provide exact copies of all materials used in our experiment for other researchers interested in repeating it. As a result, we do not believe reliability is a concern for this study.

5.1.8 Discussion

The need to better understand how we can support software engineers in developing legally compliant software systems is clear. It is increasingly important for software engineers to manage laws and regulations during development. Legal compliance may ultimately become the single most important non-functional requirement for a large number of software systems.

This research supports the goal of understanding legal compliance in software systems by providing an empirical baseline for the accuracy of legal compliance decisions made by software engineers. As we have shown in this paper, graduate-level computer science students with a background in software engineering are ill-prepared to make legal compliance decisions with any confidence. These students exhibit little consensus regarding whether or not requirements are LIR, and they disagreed with our subject matter experts on most decisions. To our knowledge, no other research exists on this topic. If our participant population of computer science graduate students does not differ substantively from the average entry-level software engineer, then this is a crucial finding for organizations that must develop software that complies with laws and regulations.

Our research further illustrates the value of involving subject matter experts in evaluating

whether or not requirements are LIR. The subject matter experts bested the experiment's participants even before conducting the consensus meeting. We note, however, that 12 out of 31 requirements did not enjoy universal agreement entering the consensus meeting. The experts' initial differences illustrate the specific difficulty of determining whether or not requirements are LIR and the general challenge of dealing with ambiguity-laden legal texts. We suspect that even subject matter experts would benefit from structured, metric-based support in making their assessments of whether or not a given requirement is LIR.

Additionally, we have shown that legal requirements metrics are a promising area of research for supporting software engineers as they face legal compliance decisions. Metrics are used in numerous domains to quickly and accurately examine aspects of natural language and source code. Despite the fact that the legal requirements triage algorithm performed poorly, we were able to show that prediction models based on the same metrics the algorithm was using performed quite well at predicting whether a requirement is LIR. We plan to use the results of this research to improve the algorithm in our future work. For example, the coefficients developed by the prediction model for the Dependency, Complexity, and Maturity terms could replace the weighting factors we discussed when creating the legal requirements triage algorithm [54]. Even if this approach fails to significantly improve the results, we could simply build a prediction model as a part of the legal requirements triage process.

5.2 Replication Case study

Two concerns from our first case study prompted a follow-up study. First, we wanted to confirm the finding that graduate-level software engineers are not able to achieve consensus about whether software requirements are LIR. Although it might have been anticipated that they would not be able to accurately assess whether the requirements were LIR, it was surprising to find that they did not achieve agreement as a group. In fact, their responses were only a marginal improvement over complete randomness.

Our second concern was that our methodology for generating consensus could have been improved. In our first study, we generated a consensus assessment for each requirement by considering each student's assessment as a vote either in favor or against a particular requirement's LIR status. We chose 50% as a cutoff: any requirement with greater than 50% of the students 'voting' it as LIR was considered to be LIR by consensus. However, another percentage cutoff may have improved our results.

In this case study, we again employ the GQM approach. Our goal remains the same as our first case study:

Analyze empirical observations for the purpose of characterizing legal implemen-

tation readiness with respect to software requirements from the viewpoint of software engineers in the context of an EHR system that must comply with HIPAA regulations.

However, we focus exclusively on two or our original six research questions outlined in our first case study:

Q2 Is there a consensus among graduate students on which requirements are LIR?

Q3 Can graduate students accurately[4] assess which requirements are LIR?

The materials for this case study were identical to those used in the first case study and described in Subsection 5.1.1. We used the same data collected from our subject matter experts as described in Subsection 5.1.3 as a canonical set of correctly assessed requirements.

Our graduate-level software engineering participant population, however, changed entirely from our first case study. This study was conducted during the Fall semester of 2011, and we had 34 participants. Our participants were, once again computer science graduate students at North Carolina State University. However, they were all[5] currently taking the graduate-level software engineering course. Furthermore, all of them had received one lecture on requirements engineering and another on legal compliance in software systems. We provided the exact same tutorial to the participants of this case study as in the first.

We now discuss the results of our experiment for both of our research questions.

Q2. Is there a consensus among graduate students on which requirements are LIR?

Measure: Fleiss Kappa statistic for 34 graduate students on 31 requirements.

The Fleiss Kappa statistic for our 34 students was $\kappa = 0.114$ ($p < 0.0001$). This result indicates that with a high level of statistical significance, the students in our case study had only a slight agreement on their assessment of the 31 requirements. This result is marginally higher than that of our first case study ($\kappa = 0.0792$), but it remains only slightly better than the result expected from a random set of responses. Thus, this result confirms the result from our first case study.

Figure 5.4 displays the responses for the students in the LIR Assessment study on the bottom bar graph, labeled "Fall 2011 Study." The top bar graph displays the data from our previous case study. We have placed them both in proximity so that they may be easily

[4]Objectively accurate assessment of LIR requirements is impossible due to the inherently subjective nature of interpreting laws and regulations. Thus, we compare against a canonical set of requirements created by experts as described in Subsection 5.1.3

[5]Some of our participants in the first case study had previously completed the course, whereas some of them were currently enrolled in it.

compared. The vertical axis represents the number of students. Students that indicated a requirements is LIR (i.e. a 'Yes' response) with a blue bar or not LIR (i.e. a 'No' response) with a red bar. The horizontal axis represents the requirement for which a decision was made. Note the similarities to the LIR Assessment study, which is displayed on the top bar graph. The are some notable differences (e.g. Requirements 17, 21, 23, and 24), but they are generally quite similar.

If the two groups are considered as a single group of 66 participants, which we can do because the two groups were provided the same materials and training, then our Fleiss Kappa statistic is $\kappa = 0.0958$)$p < 0.0001$). Once again, this result indicates slight agreement and is marginally better than random.

Answer: In every case we have tested for this research question, we found a higher level of consensus among experts than graduate-level software engineers.

Q3. Can graduate students correctly assess which requirements are LIR?

Measures: Sensitivity, specificity, and percent agreement between students' and experts' results for 31 requirements.

To answer Q3, we formed a consensus among the 34 students' votes on whether each of the 31 requirements was LIR or not to build our predictive model (RMSE = 0.49; AIC = 43.06). In our first case study, we used a simple majority to form consensus between the students: if more than 50% of the students said that a given requirement was LIR, the consensus was that the requirement was LIR. Otherwise, the consensus was that the requirement was not LIR. We used these consensus values to measure the ability of the students to accurately predict whether the requirement was LIR, using the experts' consensus as an oracle. Using this technique, the students' consensus had sensitivity of 0.9375 and specificity of 0.333. A sensitivity of 0.9375 indicates that the students had a low false negative rate; when they said a requirement needed further refinement, they were more likely to be correct. A specificity of 0.333 indicates that the students had a high false positive rate; when they said a requirement was LIR, they were more likely to be incorrect. Overall, the students agreed with the experts for only 64.5% of the requirements. Inaccurately making decisions for 35.5% of all system requirements would likely result in expensive system refactoring once the errors are discovered. If the errors are not discovered, then they may result in legal repercussions. These values confirm the findings from our LIR Assessment case study; the students were better at predicting a requirement as LIR and likely to miss a requirement that is not LIR.

Answer: We can say that graduate students cannot accurately assess the LIR status of a requirement and are more likely to miss requirements that are not LIR.

Our findings suggest that our LIR Assessment case study was correct: the participants were once again unable to achieve consensus on their assessment regarding whether the requirements were LIR. In addition, we were able to determine that voting cutoffs play a significant role in determining the accuracy of assessments.

5.3 Wideband Delphi Case Study

We conducted a third case study using the Wideband Delphi method to ensure a consensus opinion from graduate-level software engineers regarding whether the requirements from our two previous case studies were LIR. Neither our LIR Assessment case study nor our Replication case study derived a clearly superior consensus opinion using vote-based methods as to whether the requirements were LIR. We found no cutoff percentage for vote-based consensus generation that would indisputably represent consensus among graduate-level software engineers about whether requirements are LIR. The Wideband Delphi estimation method can be used to derive consensus among smaller groups ranging from 3 to about a dozen individuals [14]. Therefore, we employed it to derive consensus among 14 graduate-level software engineers about whether the requirements examined in our previous two studies were LIR. Our findings suggest that our participants were able to slightly improve the accuracy of their LIR assessments, but their assessments were much more conservative than the experts, which would likely result in over-engineering of the underlying system.

We once again employ the Goal/Question/Metric (GQM) approach [9, 10] to design our case study. For this case study, our goal changes slightly:

> *Goal:* Analyze empirical observations for the purpose of characterizing legal implementation readiness with respect to software requirements from the viewpoint of **a small group of software engineers working together** in the context of an EHR system that must comply with HIPAA regulations.

Note the changes from the goal used in both of our prior case studies are highlighted in bold. This goal prompts the following research questions:

Q7 Can graduate students working together using a Wideband Delphi method accurately assess which requirements are LIR?

Q8 What is the extent of the discussion on requirements during the application of the Wideband Delphi method?

In Subsection 5.3.1, we present the methodology and describe the material inputs used. In Subsection 5.3.2, we describe our participant population. In Subsection 5.3.3, we present

the results of our case study. Finally, in Subsection 5.3.4, we discuss the implications of this research.

5.3.1 Wideband Delphi Case Study Methodology and Materials

Because this case study was designed to answer questions surfaced in our previous two case studies, we retained the sample legal text, the requirements specification that includes a glossary of terms, and the traceability mapping of the individual requirements to the legal text described earlier in Subsection 5.1.1. These materials are provided in their entirety in Appendix A.

Our methodology for conducting this case study differed significantly from our previous two case studies in two specific ways as we now discuss. We conducted this case study on three separate days over the course of two weeks. On the first day, we introduced the case study and received completed informed consent forms for those participants interested in taking part in the study. We then provided participants with all of the materials for the case study and walked them through the same tutorial displayed in Figure 5.1. This tutorial introduced some basic concepts in legal compliance for software requirements and explained the importance of legal compliance in requirements engineering. The complete tutorial can be found in Appendix A at Section A.1.

On the first day we also provided an overview of our application of the Wideband Delphi method. Wideband Delphi can be used to derive consensus among a group of participants on a set of subjects. Subjects are the set of things about which the participants seek to achieve consensus. For example, subjects may be questions that must be answered, tasks for which effort must be estimated, or requirements that must be disambiguated. In our case study, our subjects were requirements that must be determined to be either LIR or not.

In Wideband Delphi, each participant is required to attend the initial consensus meeting having already made an initial determination for all the subjects about which consensus is to be determined. This background ensures that each participant has considered the subject at hand and formed an opinion about it. In our case study, meeting this aspect of the Wideband Delphi method meant that participants would have to take home the case study materials and determine which requirements were LIR. They were instructed to work alone and not contact other participants, and they were given 48 hours to complete the task.

The second day of the case study began two days after the first one. On this second day, we conducted our first consensus meeting. The meeting began by asking whether the participants' experience in making their initial determinations had prompted any questions. In contrast to our first two case studies in which participants made their determinations in our presence, asking questions as they arose, the participants in this study made their

determinations individually. However, during the second day consensus meeting, we did answer questions that clarify terminology or concepts. As in the previous two studies, we did not answer substantive questions about compliance or content. We discuss this initial question and answer session in Subsection 5.1.6.

Participants were instructed that our consensus meetings would be time-limited to 75 minutes.[6] Our initial plan was to conduct this as a single block of time, but we were unable to do so given the extent of the questions we received in the initial question and answer session. Therefore, our consensus meetings were divided into two sessions as described on Page 5.1.6.

During each consensus meeting, we moderated the discussion, allowing us to ensure that no one participant took over the meeting by virtue of their official capacity as the moderator. Our role as moderator entailed the following activities:

a. Beginning the discussion of a new requirement with a show-of-hands vote based on both the participants' initial determination made prior to the meeting and their thoughts on the discussion of previous requirements.

b. Answering clarifying questions raised about terminology or concepts, but not compliance or content.

c. After the initial show-of-hands vote, asking for a participant who felt the requirement was not LIR to explain their rationale. We selected a participant who felt the requirement was not LIR because any valid rationale that demonstrates a compliance concern is likely to persuade the other participants that the requirement was not LIR.

d. Allowing the discussion to continue in a free-form fashion until it seemed to either achieve consensus or stall. At that point, we called for another show-of-hands vote. If this vote was unanimously in favor of either considering the requirement under discussion as LIR or not-LIR, then we moved on to the next requirement. If this vote was not unanimous, we asked the participants if they wanted to continue discussion or move on to the next requirement. Only requirements that had received a unanimous vote in favor of considering it to be LIR were officially considered LIR at the end of the study.

e. Recording the time taken to discuss each requirement.

f. Recording the votes taken for each requirement.

g. Recording our observations on the level of disagreement for each requirement using a Likert scale as describe in Subsection 5.3.3.

[6] As described in Subsection 5.3.2, all of our participants were students in a graduate-level Requirements Engineering course at North Carolina State University. We chose this time limit because it was the duration of a single class period.

h. Recording any additional observations made during the consensus meetings, such as the number of requirements for which a participant explicitly referred to the sample legal text as a part of their rationale.

We did not explicitly pressure participants to wrap up their discussion based on the length of time the discussion was taking. However, we did tell participants at the beginning of each session how much class time was reserved for the activity.

5.3.2 Wideband Delphi Case Study Participants

Fourteen computer science graduate students currently enrolled in a graduate-level Requirements Engineering course at North Carolina State University participated in this study. A graduate-level software engineering course is a pre-requisite for the Requirements Engineering course. Through their coursework, we know that they are familiar with the basic practices and responsibilities of a software engineer working as a part of a larger team. In addition, the study was conducted after a week of lectures on legal compliance concerns in requirements engineering.

It is worth noting that these were the most sophisticated, most experienced participants of our three case studies. This was the only case study for which all of our participants were enrolled in a requirements engineering course. In addition, several of them had professional jobs as software engineers.

5.3.3 Wideband Delphi Case Study Results

The first result from our Wideband Delphi case study is the initial starting point for each participant, which represents their individual determination for each requirement prior to any group discussion. This starting point does not represent the initial vote for each requirement. Since we conducted votes by a simultaneous show-of-hands, participants could influence one another even for the first vote. Furthermore, discussion from the earlier requirements may influence the voting on later requirements. Figure 5.5 displays the initial starting point for the discussions in the Wideband Delphi case study. The vertical axis represents the number of students. Students that indicated a requirements is LIR (i.e. a 'Yes' response) with a blue bar or not LIR (i.e. a 'No' response) with a red bar. The horizontal axis represents the requirement for which a decision was made. Note that participants have indicated more requirements are not LIR than the previous case studies shown in Figure 5.4, particularly for Requirements 6 through 16. Also, note only 13 students are represented in this Figure because one student did not record their individual determinations for each requirement prior to the case study.

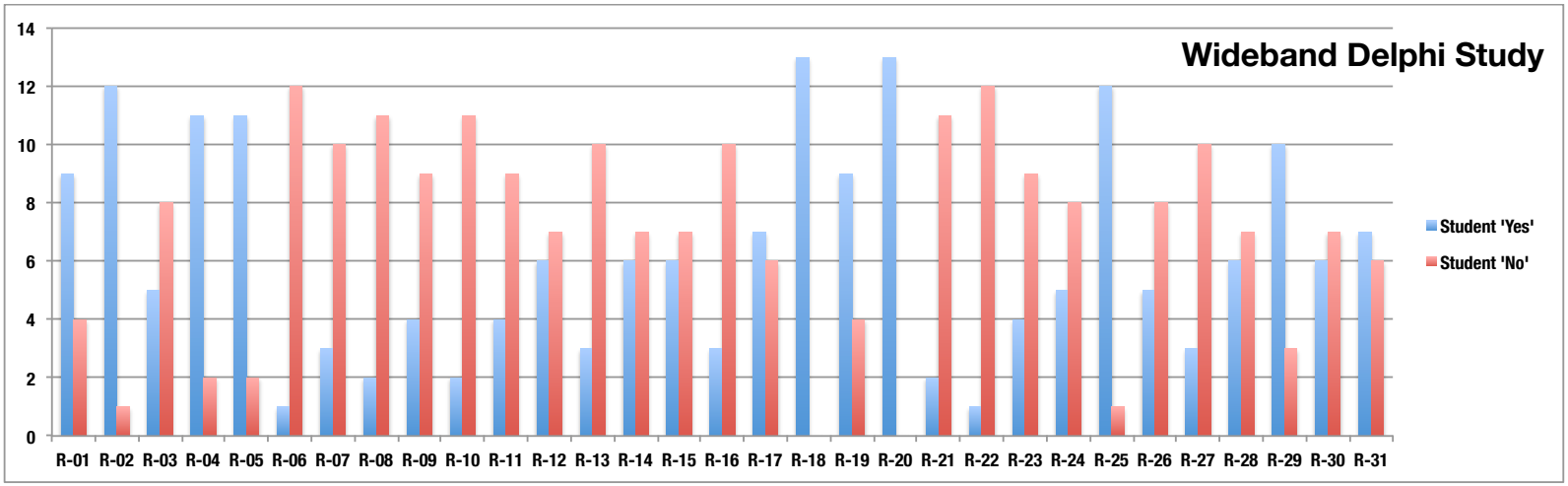

Figure 5.5: Assessment of Requirements by Participants Prior to Wideband Delphi Method

Recall that we once again employ the Goal/Question/Metric (GQM) approach [9, 10] for the Wideband Delphi case study with the following goal:

> *Goal:* Analyze empirical observations for the purpose of characterizing legal implementation readiness with respect to software requirements from the viewpoint of a small group of software engineers working together in the context of an EHR system that must comply with HIPAA regulations.

For this case study, we seek to examine the accuracy of the assessments and the extent of the consensus within the group. We now discuss results for the accuracy of the assessments made by consensus, which was represented in this study using the following question:

Q7. Can graduate students working together using a Wideband Delphi method accurately assess which requirements are LIR?

Measures: Sensitivity, specificity, and percent agreement between graduate students' and experts' results for 31 requirements.

Let us first consider the level of agreement between the graduate students' consensus assessment using the experts's consensus assessment from the LIR Assessment case study as an oracle. The consensus graduate student opinion only agreed with the expert opinion for 54.8% of the requirements. These results resolve the discrepancy identified in our first two case studies. Recall that the participants in our LIR Assessment case study agreed with the expert opinion for 54.8% of the requirements as well. In our Replication case study, the participants agreed with the expert opinion for 64.5% of the requirements. Both of these values were calculated based on the 50% voting methods used in both of our first two case studies. However, we can say that when using a methodology designed to achieve consensus, the consensus achieved by the graduate-level software engineers remained roughly the same, which provides more evidence that they are ill-prepared to make legal compliance assessments of software requirements.

We then constructed a predictive model (RMSE = 0.52; AIC 45.48). The sensitivity and specificity for the consensus generated using the Wideband Delphi method were 0.313 and 0.800, respectively. Recall that sensitivity measures the proportion of LIR requirements that were accurately identified as LIR, whereas specificity measures the proportion of non-LIR requirements that were accurately identified as non-LIR. We are particularly interested in specificity because identification of a non-LIR requirement prior to design and implementation is crucial to a legal compliance effort.

The results generated for specificity by the graduate-level software engineers using the Wideband Delphi method were a significant improvement over those from both our first

(0.20) and our second case studies (0.333). However, the results for sensitivity significantly worsened. In this study, the sensitivity measured 0.313, whereas in the first study sensitivity was 0.875 and in the second study it was 0.938. In short, the students became overly cautions in their assessment of requirements. This is an improvement in the sense that it means fewer legal compliance risks, but it also means increased development costs as engineers must now re-evaluate and potentially refine requirements that were already LIR. An ideal evaluation of requirements would have both high sensitivity and high specificity.

Q8. What is the extent of the discussion on requirements during the application of the Wideband Delphi method?

Measures: Length of discussion in minutes and observation of the level of disagreement on a Likert scale from 1 (very little disagreement) to 5 (very high disagreement).

As discussed in Subsection 5.3.1, this case study was conducted with a planned 75 minute window for discussion. To ensure that each requirement was discussed, the participants agreed to end discussion without achieving unanimous consensus seven times. The participants also agreed that those requirements where unanimous consensus could not be reached would be considered non-LIR. Had the participants used the 50% voting method to achieve consensus for those requirements, only one of them (R-27) would have been considered LIR; the others (R-11, R-14, R-15, R-28, R-30, and R-31) would have all been considered non-LIR.

The length of discussion for each requirement varied from unanimous agreement at the outset to roughly 8 minutes of discussion. For example, Requirement 11, displayed in Figure 5.6, was discussed for 6 minute and 18 seconds. The participants were split between two beliefs about the account structure described by the iTrust system. One group believed that employee accounts were distinct from patient accounts; the other group believed that employees with accounts must use the same account when they are considered patients in the system. Ultimately, the participants decided that they would not be able to come to a unanimous consensus for this requirement, which meant that it is considered not LIR. Figure 5.7 displays the duration of the discussion for each of the requirements.

In addition to recording the duration of discussion for each requirement, we observed and

R-11: When an employee record is updated, iTrust shall send an email to an employee's email address according to the stored preferences regarding email for that employee. [Traces to §164.312(b)]

Figure 5.6: iTrust Requirement 11

recorded the level of disagreement for each requirement using a Likert scale. For a discussion that was unanimous or nearly unanimous, we recorded a 1, indicating very little disagreement. If a participant raised a point for the sake of argument but didn't press the point when challenged, we recorded a 2, indicating little disagreement. If participants discussed two or more alternative interpretations and weighed them seriously before making a decision, we recorded a 3, indicating moderate disagreement. If participants discussed two or more alternative interpretations with a standard bearer from each alternative actively campaigning for their choice, we recorded a 4, indicating a high level of disagreement. If participants discussed two or more alternatives and either appeared to be unable to come to consensus or actually were unable to come to a consensus as a result of the discussion, we recorded a 5, indicating a high level of disagreement.

We also noted the content of the discussion for those requirements with a non-trivial amount of discussion (i.e. those requirements that were not unanimously or nearly unanimously decided). For example, we noted the specific words or phrases that were interpreted differently by the various participants. We also noted whether the participants referenced the glossary, the traceability matrix, or the legal text during these disputes.

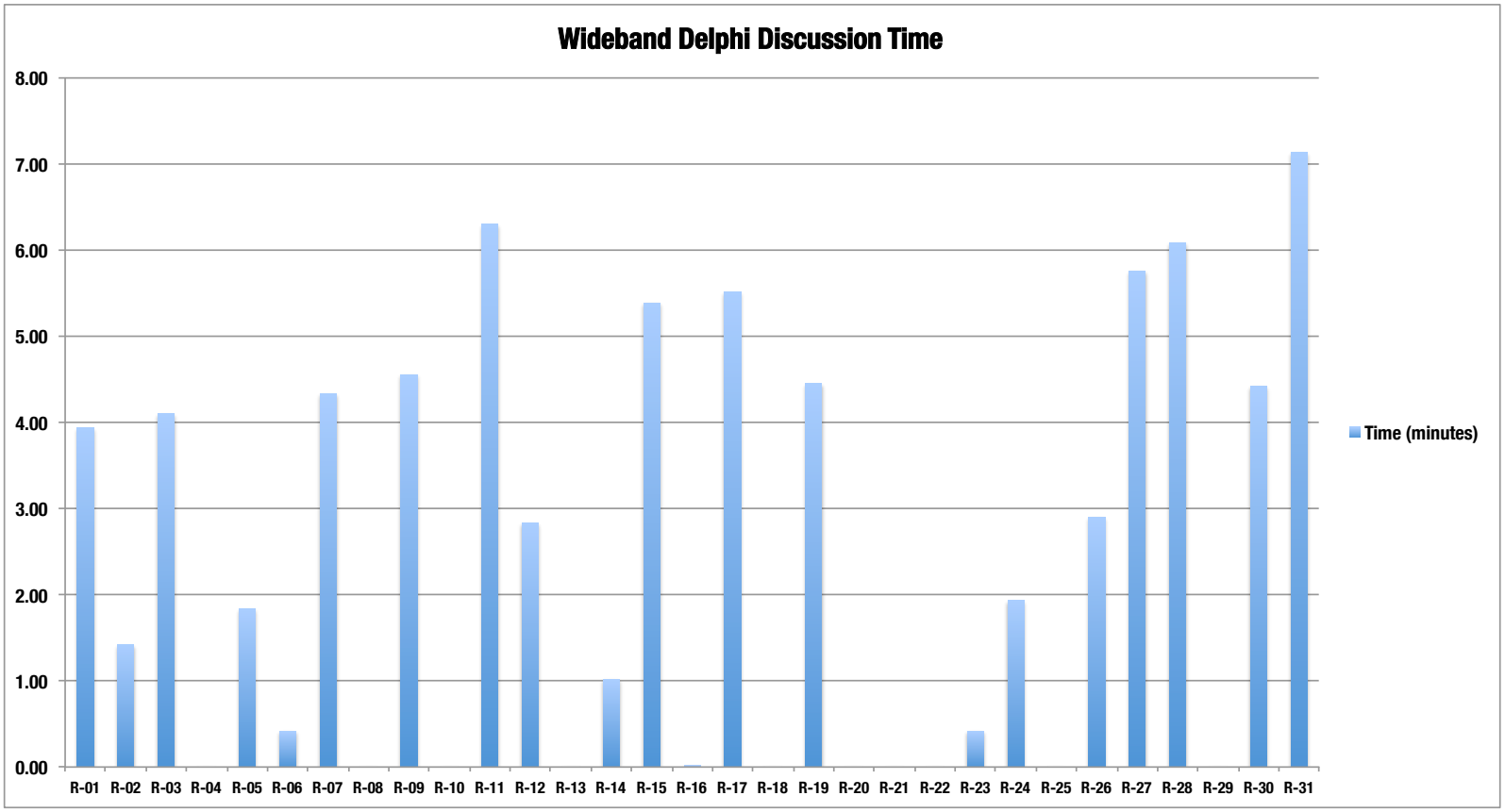

Figure 5.7: Duration of Discussion for Requirements in the Wideband Delphi Case Study

5.3.4 Wideband Delphi Case Study Discussion

In this section, we discuss the results of and observations from our Wideband Delphi case study. Our discussion begins with the second day of our case study, which was after the participants had made their individual determinations for each requirement.

Question and Answer Session

The second day of our case study began with a brief question and answer period. Recall that after providing our participants with our tutorial on legal implementation readiness, they had two days prior to our first consensus session to make their individual determinations about the requirements we would be discussing in our consensus session. They came back with two basic types of questions about the study, as we now discuss.

The first type of question were clarifying questions about the concepts (e.g. legal implementation readiness or refinement of requirements) from the study. We responded to these as directly and succinctly as possible. For example, one participant wanted to know whether the requirements that were determined by consensus to be LIR would be implemented immediately after the decision was made. We explained that additional design and engineering concerns may require further refinement of the requirement, but that it would not be refined further to address legal concerns prior to being implemented.

Several students wanted to know whether the iterative cycles in the development process used for these requirements were days, weeks, or months long. We explained that LIR decisions could be made in short or long development cycles, and that the more important concern was that any requirement determined to be LIR would not be further examined as a part of the legal requirements compliance process.

The second type of question the participants asked related to the content of the study. We responded to these questions by reinforcing that they were ultimately the sort of things we wanted the participants themselves to consider as they determined whether the requirements in the study were LIR. For example, some students asked what weight they should give to the cross-reference in the sample legal text. Figure 5.8 displays the cross-reference they were concerned about.

We explained that participants could determine for themselves whether or not they believed this cross reference was important from a legal compliance standpoint. We also told the participants that it may or may not contain information relevant to the various requirements that traced to § 164.312(a)(1), and each participant could decide whether they wanted to concern themselves with this sort of risk.

Participants also asked several questions about the glossary terms. For example, one participant want to know if there was any overlap between the iTrust roles defined in the

glossary provided. We explained that the anything not explicitly excluded by the role could be a valid concern to anyone in a legal compliance decision-making process.

The third type of question participants asked was procedural. For example, the students wanted to know more about the role of the moderator. We explained that my purpose was to facilitate the consensus sessions, including prompting participants to vote, observing and recording aspects the discussion, and ensuring we proceeded from one requirement to another if the vote resulted in consensus or if it was clear consensus could not be achieved in a timely fashion.

First Consensus Session

Once participant questions had been adequately addressed, we began the first discussion session. This session took 55 minutes and covered the first 26 requirements. Our stated goal was to take no more than 75 minutes to discuss the requirements. Although time limitations are not typically a part of the Wideband Delphi process, they are a real world constraint. Figure 5.7 details how long participants discussed each requirement.

The discussion was conducted one requirement at a time starting with the first and ending with the 31st. For each requirement, the discussion started with an initial vote, conducted by a show-of-hands. For the first several requirements, these initial votes were identical to the participants' individual determinations made in the previous 48 hours. However, as the votes progressed, some participants altered their decisions for their initial public vote based on the discussion of previous requirements.

For many of the early requirements, the participants focused the discussion on terminology. In particular, the first requirement, below, spawned an extended discussion about the uniqueness of user IDs:

> *R-1:* iTrust shall support user authentication by requiring a user to input their user ID and password. [Traces to §164.312(a)(1), §164.312(a)(2)(i), and §164.312(d)]

```
(a) (1) Standard: Access control. Implement technical
policies and procedures for electronic information systems
that maintain electronic protected health information to
allow access only to those persons or software programs
that have been granted access rights as specified in
§ 164.308(a)(4).
```

Figure 5.8: HIPAA § 164.312(a)(1)

Some participants asked questions about account hierarchies based on the definitions provided in the glossary (see Appendix A). These participants were concerned that employees who were also patients in the healthcare facility would be able to violate access control rules if they had two separate accounts. The participants spent some time discussing this before determining that many of the authentication requirements were LIR based on the definitions provided in the glossary. Although it was not the purpose of this study, it is interesting to note that the actor hierarchies discussed in Chapter 3 may have alleviated some of this discussion had they been available.

Second Consensus Session

We were unable to achieve consensus on every requirement before our scheduled class session ended. Therefore, we completed the consensus building in a second session. The second consensus session took about 20 minutes, covered the final five requirements, and consisted of 12 participants. In addition, two of the thirteen participants from the first session were unable to attend the second. One participant who was unable to attend the first session was present for the second.

We began the second session with a brief overview of the previous session. When asked, the participants unanimously decided that their determinations for each of the previous 26 requirements remained correct and did not require revision. We then proceeded to examine the final five requirements, which proved to be some of the most contentious requirements in the study. The participants in the study agreed that they would be unable to achieve a unanimous consensus for four of these final five requirements. For example, the discussion for Requirement 27, displayed in Figure 5.9, focused on whether or not a cryptographic hash would provide any level of security. Some participants felt that it was intended to do so; others felt that the hash was meant simply as a means of identification. The participants felt that it was clear these two groups would not agree on the final decision.

At the end of the session, we asked the participants if they felt their determinations for any of the requirements had changed. The participants unanimously agreed that their earlier determinations remained accurate even after subsequent discussion.

> **R-27:** Each time a printable emergency report for a medical record is generated, iTrust shall include a cryptographic hash of the authenticated employee user ID and the patient user ID at the end of the report. [Traces to §164.312(b)]

Figure 5.9: iTrust Requirement 27

5.4 Chapter Summary

In this chapter, we discuss three validation studies for the legal requirements metrics:

The LIR Assessment Case Study is an initial examination of how graduate-level software engineers assess software requirements for legal implementation readiness. The findings from this study indicate that they are ill-prepared to make these determinations. This study also provides empirical evidence that our legal requirements metrics defined in Chapter 4 are useful for determining whether requirements are LIR.

The Replication Case Study confirms our findings from the LIR Assessment case study; graduate-level software engineers are ill-prepared to make LIR assessments of software requirements. This study also demonstrated that vote-based methods for determining consensus among graduate-level software engineers are not the most effective method for achieving consensus about LIR assessments.

The Wideband Delphi Case Study addresses concerns from the previous two findings, neither of which derived a clearly superior consensus opinion using vote-based methods regarding whether the requirements were LIR. We used the Wideband Delphi method, which is designed for achieving consensus, to determine if graduate-level software engineers were able to make more accurate LIR assessments of requirements by consensus. Our findings suggest a slight improvement in the accuracy of their assessments. However, both our quantitative findings and our qualitative findings suggest that graduate-level software engineers using the Wideband Delphi method to achieve consensus about software requirements are overly cautious in their assessments.

Each of these studies is designed to examine the nature of assessing software requirements for legal compliance. In the next chapter, we discuss the contributions of this dissertation, potential limitations and threats to the validity of our case studies, and potential future work in the area of legal compliance in software systems.

Chapter 6

Conclusion

"We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time."

–T.S. Elliot

Software systems are increasingly important to modern society. Marc Andreessen, a prominent venture capitalist and the creator of one of the earliest web browsers, recently wrote an article titled "Why Software is Eating the World" for the Wall Street Journal [2]. In it, he provides several examples of how software systems are taking over entire industries traditionally thought to be unrelated to software. Netflix is taking over video rentals. Amazon is taking over book stores. Music, film, photography, navigation, communications, and other industries are almost completely dependent upon software for their functionality. He believes healthcare is next:

> Health care and education, in my view, are next up for fundamental software-based transformation. My venture capital firm is backing aggressive start-ups in both of these gigantic and critical industries. We believe both of these industries, which historically have been highly resistant to entrepreneurial change, are primed for tipping by great new software-centric entrepreneurs.
>
> Even national defense is increasingly software-based. The modern combat soldier is embedded in a web of software that provides intelligence, communications, logistics and weapons guidance. Software-powered drones launch airstrikes without putting human pilots at risk. Intelligence agencies do large-scale data mining with software to uncover and track potential terrorist plots.

If software is to become fundamental to these undeniably critical aspects of our society,

then software engineers must have techniques and tools that can evaluate those systems for compliance with relevant laws and regulations. The basic ethics espoused by the Hippocratic Oath or the Code of Hammurabi remain a critical aspect of modern society. The increasingly important role of software raises a pressing legal, ethical, and engineering question: How can we ensure software is built to support legal compliance?

This dissertation examines a narrow element of legal compliance analysis in software systems: the identification of requirements that have met or exceed their legal obligations, which we call Legally Implementation Ready (LIR) requirements. We have examined the identification of LIR requirements within the healthcare domain and focusing on the U.S. Health Insurance Portability and Accountability Act (HIPAA). This dissertation has several contributions, including:

a. An empirically validated methodology for tracing software requirements to specific subsections of the law. This methodology can be used to perform an evaluation of legal compliance of software requirements.

b. A set of eight empirically evaluated legal requirements metrics that can be used to estimate the dependency, complexity, and maturity of a set of software requirements that must comply laws and regulations. These metrics provide engineers a means to identify different interpretations of legal compliance, a means to identify concerns for which a legal domain expert should be consulted, and enable discussions about which interpretation is preferable in a given context.

c. A prototype tool that supports identifying LIR requirements using the legal requirements metrics defined by this dissertation.

The empirical case studies used to validate these contributions resulted in several important lessons learned, including:

Traceability supports consensus building. It is possible for graduate-level software engineers to achieve consensus even if all an organization does to improve their legal compliance is trace their requirements to the applicable subsections of the legal text.

Graduate-level software engineers are ill-prepared to assess software requirements for legal compliance. Legal domain experts are necessary for a complete legal requirements analysis.

When using a Wideband Delphi method designed to achieve consensus among graduate-level software engineers, the consensus achieved is overly conservative. Building a legally compliant software system involves balancing the need to ensure compliance with other software engineering

constraints. If software engineers greatly exceed their legal obligations, they may produce software that is over budget or compromise the quality of the resulting system.

My empirical studies suggest that the average graduate-level software engineer is ill-prepared to identify legally compliant software requirements with any confidence and that domain experts are an absolute necessity. When working together as a team graduate-level software engineers make extremely conservative legal implementation readiness decisions. Furthermore, we observe that the legal requirements metrics discussed in this dissertation can be used to improve legal implementation readiness decisions. These findings, along with legal and ethical concerns, make the study of legal compliance in software engineering a critical area for continued research.

6.1 Threats to Validity

To evaluate case study research, we must consider possible threats to the validity of individual studies. In this section, we examine four possible threats to the validity of the case studies presented in this dissertation.

Construct validity refers to the appropriateness and accuracy of the measures and metrics used for the concepts studied [75].

External validity refers to the ability to generalize the findings of a case study to other domains [75].

Internal validity refers to the ability to establish causal relationships based on recorded observations from the study [75].

Reliability refers to the ability of other researchers to repeat a case study.

The case studies in this dissertation are descriptive in nature. For example, Chapter 4 only attempt to describe the relationship that exists between the structure of a legal text and the software requirements that can be traced to them. We have not attempted to establish a causal relationship, and, therefore, we do not need to consider internal validity as a threat.

Using multiple sources of data mitigates *construct validity* by improving the appropriateness and accuracy of results obtained. The case studies in this dissertation rely on several sources for requirements for the iTrust Medical Records system: (1) the original source documentation created by the designers of the system; (2) the requirements identified by Maxwell and Antón [53]; (3) the requirements constructed by as a result of the case study described in Chapter 3; and (4) those requirements designed to fill out the validation studies described in Chapter 5.

In addition, multiple researchers reviewed the protocols and procedures for each case study in this dissertation multiple times. We have strictly adhered to the methodologies presented for each case study.

To address potential threats to *external validity*, we employed as many real-world regulations as possible. The Health Insurance Portability and Accountability Act of 1996 (HIPAA) is actual legislation that has been enforced [11, 23]. The U.S. Department of Health and Human Services (HHS), which is charged with creating, maintaining, and enforcing regulations pursuant to HIPAA, has required systematic changes in institutional privacy practices as a result of nearly 15,000 resolved HIPAA investigations.[1] HIPAA violations can result in serious monetary penalties. HHS recently fined one healthcare provider $4.3 million dollars for violations of the HIPAA Privacy Rule.[2]

HIPAA represents a single legal domain, and although we have no scientific evidence to demonstrate that it is quantitatively similar to other laws and regulations, we have consulted with legal experts and determined that HIPAA is significantly similar in form to other executive branch administrative regulations. For example, many other regulations are structured in sections and subsections, which are crucial to the application of our legal requirements metrics. Although no healthcare provider is currently using iTrust in industry, healthcare professionals have consulted with the developers of iTrust from the beginning as described in Chapter 3. In addition, other researchers are using iTrust as a subject of study for requirements engineering research [24].

Internal validity refers to the causal inferences made based on experimental data [75]. Computer science graduate students may identify different requirements as LIR than software engineers working in industry. We did not ask participants to report the details of their previous professional experience, which may have affected inferences in this research. We plan to conduct additional studies using professional software engineers and legal domain experts to further validate the inferences made in this research.

To ensure that other researchers are able to repeat our case study, which would address potential threats to *reliability*, we have provided all of the materials used in the studies described in Chapter 5 in Appendix A. When our first validation case study resulted in clearly demonstrating that graduate-level software engineers are ill-prepared to determine whether requirements are legally implementation ready, we repeated the study on a completely new population as described in Section 5.2 to ensure the result was repeatable. We developed a prototype tool, called Red Wolf, that supports and automates the calculation of our legal requirements metrics. We are currently in the process of preparing Red Wolf for release as an open source requirements management tool that supports legal requirements analysis.

[1] `http://www.hhs.gov/ocr/privacy/hipaa/enforcement/highlights/index.html`
[2] `http://www.hhs.gov/news/press/2011pres/02/20110222a.html`

6.2 Future Work

Evaluating legal compliance in existing systems or determining which requirements have met or exceeded their legal obligations are persistent problems that present numerous avenues for future research. In this dissertation, we described the development of a set of legal requirements metrics that can be used to evaluate and improve the legal compliance of a software system. These metrics are a first effort towards empirically measuring the legal compliance efforts of a software system.

Although we provided a methodology for evaluating legal compliance and developed legal requirements metrics that can guide a legal compliance process as a part of our methodology, we have by no means provided a comprehensive overview of the extensive legal compliance issues faced by an EHR system [8, 16, 17, 18, 60]. For example, we have not addressed the prioritization of legal obligations and exceptions or the management of evolving laws and regulations over time. We have also not addressed the acquisition or specification of legal requirements, which as only recently become the subject of extensive research [15].

We have also shown that graduate-level software engineers can use these traceability links to achieve consensus about which requirements are LIR, but we have not examined the effects of traceability links beyond requirements artifacts. If a design document were created, then this traceability could be extended from the requirements phase to the design phase and eventually to the test phase of software development. A design document would also enable the refactoring of the source code to ensure legal compliance. Future work could examine the effects of following the traceability link from the legal text to the design, to implementation, to test, and to deployment.

In addition to improving traceability, future work may include refining the requirements through examination of domain expertise. The regulatory landscape is complex and better techniques for representing and maintaining domain knowledge may significantly improve our ability to reason about legal compliance in software systems. For example, legal compliance experts use risk analysis to determine which aspects of an organization present the most critical legal concerns. Incorporating this domain knowledge may appreciably improve the accuracy and usefulness of all software artifacts related to any software system.

In this dissertation, we examine natural language requirements, but numerous other methods of representing requirements are actively used in industry. Goals, scenarios, and other models of requirements are traditional areas of requirements engineering have a long research history associated with disambiguating and clarifying the requirements for a system [1, 3, 4, 6, 69, 70, 71]. These methods of representing requirements may also prove valuable for legal compliance analysis and are worthy of examination.

Our legal requirements metrics and the process of performing a legal requirements triage

may both be extended and improved through future work. The eight legal requirements metrics presented in this dissertation do not in any way imply that other metrics useful for legal compliance analysis may not exist. In addition, the predictive model used to perform legal requirements triage may be improved by incorporating additional legal domain knowledge.

REFERENCES

[1] K. Allenby and T. Kelly. Deriving safety requirements using scenarios. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 228–235, 2001.

[2] Marc Andreessen. Why Software is Eating the World. Wall Street Journal, `http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html`, Aug. 2011.

[3] A.I. Antón. Goal-based Requirements Analysis. *Proceedings of the Second International Conference on Requirements Engineering*, pages 136–144, 15-18 Apr 1996.

[4] A.I. Antón, R.A. Carter, A. Dagnino, J.H. Dempster, and D.F. Siege. Deriving goals from a use-case based requirements specification. *Requirements Engineering*, 6(1):63–73, 2001.

[5] Annie I. Antón, Julia B. Earp, and Ryan A. Carter. Precluding incongruous behavior by aligning software requirements with security and privacy policies. *Information and Software Technology*, 45(14):967–977, 2003.

[6] Annie I. Antón and Colin Potts. A representational framework for scenarios of system use. *Requirements Engineering*, 3(3):219–241, 03 1998.

[7] P. Avesani, C. Bazzanella, A. Perini, and A. Susi. Facing Scalability Issues in Requirements Prioritization With Machine Learning Techniques. *13th IEEE International Conference on Requirements Engineering*, pages 297–305, Aug.-2 Sept. 2005.

[8] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and Contextual Integrity: Framework and Applications. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

[9] Victor R. Basili. Software modeling and measurement: the goal/question/metric paradigm. Technical report, College Park, MD, USA, 1992.

[10] Victor R. Basili. *Applying the Goal/Question/Metric Paradigm in the Experience Factory*, pages 21–44. International Thomson Computer Press, 1995.

[11] Kevin Beaver and Rebecca Herold. *The Practical Guide to HIPAA Privacy and Security Compliance*. Auerbach Publications, 2004.

[12] B. Berenbach, D. Grusemann, and J. Cleland-Huang. The Application of Just In Time Tracing to Regulatory Codes and Standards. *Eighth Conference on Systems Engineering Research*, 2010.

[13] Brian A. Berenbach. Comparison of uml and text based requirements engineering. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 247–252, New York, NY, USA, 2004. ACM.

[14] Barry W. Boehm. *Software Engineering Economics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[15] Travis D. Breaux. *Legal Requirements Acquisition for the Specification of Legally Compliant Information Systems*. PhD thesis, North Carolina State University, 2009.

[16] Travis D. Breaux and Annie I. Antón. Mining rule semantics to understand legislative compliance. In *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*, pages 51–54, New York, NY, USA, 2005. ACM Press.

[17] Travis D. Breaux and Annie I. Antón. Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering*, 34(1):5–20, Jan. 2008.

[18] Travis D. Breaux, Matthew W. Vail, and Annie I. Antón. Towards Regulatory Compliance: Extracting Rights and Obligations to Align Requirements with Regulations. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 49–58, Washington, DC, USA, September 2006. IEEE Society Press.

[19] Frederick P. Brooks. *The Mythical Man-Month*. Addison-Wesley, anniversary edition, 1995.

[20] Ronald Benton Brown and Sharon Jacobs Brown. *Statutory Interpretation: The Search for Legislative Intent*. Natl Inst of Education, December 2002.

[21] Raymond P.L. Buse and Westley R. Weimer. A metric for Software Readability. In *Proceedings of the 2008 international symposium on Software testing and analysis*, ISSTA '08, pages 121–130, New York, NY, USA, 2008. ACM.

[22] George W. Bush. A new generation of american innovation. Executive Office of the President of the United States, April 2004.

[23] Young B Choi, Kathleen E Capitan, Joshua S Krause, and Meredith M Streeper. Challenges associated with privacy in health care industry: implementation of hipaa and the security rules. *J Med Syst*, 30(1):57–64, 2006 Feb.

[24] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A Machine Learning Approach for Tracing Regulatory Requirements Codes to Product Specific Requirements. *32nd International Conference on Software Engineering*, May 2-8 2010.

[25] A.M. Davis. The Art of Requirements Triage. *Computer*, 36(3):42–49, Mar 2003.

[26] Catherine M DesRoches, Eric G Campbell, Sowmya R Rao, Karen Donelan, Timothy G Ferris, Ashish Jha, Rainu Kaushal, Douglas E Levy, Sara Rosenbaum, Alexandra E Shields, and David Blumenthal. Electronic health records in ambulatory care–a national survey of physicians. *N Engl J Med*, 359(1):50–60, 2008 Jul 3.

[27] Chuan Duan, Paula Laurent, Jane Cleland-Huang, and Charles Kwiatkowski. Towards Automated Requirements Prioritization and Triage. *Requirements Engineering*, 14(2):73–89, 06 2009.

[28] Einer Elhauge. *Statutory Default Rules: How to Interpret Unclear Legislation*. Harvard University Press, 2008.

[29] William N. Eskridge. *Dynamic Statutory Interpretation*. Harvard University Press, December 1994.

[30] R. F. Flesch. A New Readability Yardstick. *Journal of Applied Psychology*, 32:221–233, 1948.

[31] Philip P. Frickey, Elizabeth Garrett, and William N.Eskridge. *Legislation and Statutory Interpretation*. Foundation Press, January 2006.

[32] R. Gunning. *The Technique of Clear Writing*. McGraw-Hill International Book Co, 1952.

[33] Andrea Herrmann and Maya Daneva. Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research. *16th IEEE International Requirements Engineering*, pages 125–134, Sept. 2008.

[34] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, 2nd. edition, 2000.

[35] J.L. Fleiss and J. Cohen. The Equivalence of Weighted Kappa and the Intraclass Correlation Coefficient as Measures of Reliability. *Educational and Psychologial Measurement*, 33:613–619, 1973.

[36] J. Karlsson. Software Requirements Prioritizing. *Proceedings of the Second International Conference on Requirements Engineering*, pages 110–116, Apr 1996.

[37] J. Karlsson and K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, 14(5):67–74, Sep/Oct 1997.

[38] Joachim Karlsson, Claes Wohlin, and Björn Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14-15):939 – 947, 1998.

[39] Shawn Kerrigan and Kincho H. Law. Logic-Based Regulation Compliance-Assistance. In *Proceedings of the 9th International Conference on Artificial Intelligence and Law*, pages 126–135, New York, NY, USA, 2003. ACM.

[40] J.P. Kincaid and E.A. Smith. Derivation and Validation of the Automated Readability Index for use with Technical Materials. *Human Factors*, 12:457–464, 1970.

[41] B. Kitchenham, S.L. Pfleeger, and N. Fenton. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929 –944, dec 1995.

[42] Gloria T. Lau, Shawn Kerrigan, Kincho H. Law, and Gio Wiederhold. An e-government information architecture for regulation analysis and compliance assistance. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, pages 461–470, New York, NY, USA, 2004. ACM.

[43] P. Laurent, J. Cleland-Huang, and Chuan Duan. Towards Automated Requirements Triage. In *15th IEEE International Requirements Engineering Conference*, pages 131–140, Oct. 2007.

[44] S.P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[45] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.

[46] Fabio Massacci, Marco Prest, and Nicola Zannone. Using a security requirements engineering methodology in practice: The compliance with the Italian data protection legislation. *Computer Standards & Interfaces*, 27(5):445–455, 2005.

[47] Aaron Massey, Paul Otto, Lauren Hayward, and Annie Antón. Evaluating Existing Security and Privacy Requirements for Legal Compliance. *Requirements Engineering*, 15:119–137, 2010. 10.1007/s00766-009-0089-5.

[48] Aaron K. Massey and Annie I. Antón. Triage for Legal Requirements. Technical report, North Carolina State University, 2010.

[49] Aaron K. Massey, Paul N. Otto, and Annie I. Antón. Legal Requirements Prioritization. *Proc. of the 2nd Intl. IEEE Workshop on Requirements Engineering and the Law*, 2009.

[50] Aaron K. Massey, Paul N. Otto, Lauren J. Hayward, and Annie I. Antón. Evaluating Existing Security and Privacy Requirements for Legal Compliance. *Requirements Engineering*, 2009.

[51] J.C. Maxwell, A.I. Anton, and P. Swire. A legal cross-references taxonomy for identifying conflicting software requirements. In *Requirements Engineering Conference (RE), 2011 19th IEEE International*, pages 197 –206, 29 2011-sept. 2 2011.

[52] Jeremy C. Maxwell and Annie I. Antón. Developing Production Rule Models to Aid in Acquiring Requirements from Legal Texts. *17th IEEE International Requirements Engineering Conference*, pages 101 –110, 31 2009-Sept. 4 2009.

[53] Jeremy C. Maxwell and Annie I. Antón. Validating Existing Requirements for Compliance with Law Using a Production Rule Model. *Proc. of the 2nd Intl. IEEE Workshop on Requirements Engineering and the Law*, pages 1–6, 2009.

[54] Jeremy C. Maxwell and Annie I. Antón. Discovering Conflicting Software Requirements by Analyzing Legal Cross-References. *(In Submission) IEEE International Requirements Engineering Conference*, 2010.

[55] Michael J. May, Carl A. Gunter, and Insup Lee. Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies. *Proceedings of the Computer Security Foundations Workshop*, pages 85 – 97, 2006.

[56] G.J. McLachlan, K.A. Do, and C. Ambrose. *Analyzing Microarray Gene Expression Data*. John Wiley and Sons, 2004.

[57] G.H. McLaughlin. SMOG grading – a new readability. *Journal of Reading*, May 1969.

[58] A.P. Meneely, B.H. Smith, and L. Williams. Validating Software Metrics: A Spectrum of Philosophies. *Transactions on Software Engineering Methodologies*, 2011.

[59] D.L. Olson and D. Delen. *Advanced Data Mining Techniques*. Springer, 2008.

[60] Paul N. Otto and Annie I. Antón. Addressing Legal Requirements in Requirements Engineering. *15th IEEE International Requirements Engineering Conference*, pages 5–14, 15-19 Oct. 2007.

[61] A. Perini, A. Susi, F. Ricca, and C. Bazzanella. An Empirical Study to Compare the Accuracy of AHP and CBRanking Techniques for Requirements Prioritization. *Fifth International Workshop on Comparative Evaluation in Requirements Engineering*, pages 23–35, Oct. 2007.

[62] Henry Petroski. *To Engineer Is Human*. Vintage Books, 1982.

[63] C. Potts, K. Takahashi, and A.I. Antón. Inquiry-based requirements analysis. *Software, IEEE*, 11(2):21–32, March 1994.

[64] K. Ryan and J. Karlsson. Prioritizing Software Requirements In An Industrial Setting. In *Proceedings of the 19th International Conference on Software Engineering*, pages 564–565, May 1997.

[65] Thomas L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[66] Luo Si and Jamie Callan. A Statistical Model for Scientific Readability. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pages 574–576, New York, NY, USA, 2001. ACM.

[67] E. Simmons. Requirements Triage: What Can We Learn From a "Medical" Approach? *IEEE Software*, 21(4):86–88, July-Aug. 2004.

[68] Emily Singer. A big stimulus boost for electronic health records. MIT Technology Review, February 20 2009.

[69] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262, 2001.

[70] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in system development: current practice. *Software, IEEE*, 15(2):34–45, Mar/Apr 1998.

[71] J. Whittle and J. Schumann. Generating statechart designs from scenarios. *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 314–323, 2000.

[72] L. Williams and Y. Shin. Wip: Exploring security and privacy concepts through the development and testing of the itrust medical records system. *Frontiers in Education*, pages S1F30–31, 2006.

[73] L. Williams, T. Xie, and A. Meneely. The iTrust Medical Records System, September 2008.

[74] L. Williams, T. Xie, and A. Meneely. The iTrust Medical Records System, September 2008.

[75] Robert K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications, 3rd edition, 2003.

[76] Ernst & Young. 9th Annual Global Information Security Survey. 2006.

[77] Ernst & Young. 10th Annual Global Information Security Survey. 2007.

[78] Ernst & Young. 13th Annual Global Information Security Survey. 2010.

APPENDIX

Appendix A

Materials for User Study

In this appendix, we provide the materials used in several of our case studies for reference or replication. Chatper 5 details how we used these materials. All section references (§) refer to HIPAA.

A.1 iTrust Medical Records System: Requirements for Technical Safeguards

Physicians and healthcare practitioners use Electronic Health Records (EHR) systems to obtain, manage, and share patient information. With paper-based healthcare records, access to, and verification of, a patient's medical history is often difficult. However, the transition from a paper-based record keeping system to an EHR introduces new concerns for the security and privacy of patient information.

In 1996, Congress passed the Health Insurance Portability and Accountability Act (HIPAA), which safeguards the security and privacy of both paper-based and electronic patient medical records. Penalties for non-criminal violations of HIPAA are severe and could result in fines of up to $25,000 per violation per year. Criminal violations of HIPAA could result in fines of up to $250,000 and up to 10 years in jail.

This document describes the software requirements for technical safeguards section of HIPAA in the iTrust Medical Records System, an open source EHR system designed and implemented by the students and faculty at North Carolina State University. Each requirement includes a description of the functionality to be implemented in software and a traceability link to show which subsection of HIPAA requires this functionality. A traceability matrix appears at the end of the document for reference.

Your task is to determine which of the iTrust requirements included in this document meet

or exceed the legal obligations outlined in HIPAA §164.312: Technical Safeguards provided as a part of this packet. Any requirements that do not meet or exceed these legal obligations will be refined and prepared for the next iteration in the software development lifecycle.

For example, consider Requirement A:

> **Requirement A:** iTrust shall generate a unique user ID and default password upon account creation by a system administrator. [Traces to §164.312(a)(1) and §164.312(a)(2)(i)]

Requirement A meets or exceeds its legal obligations outlined in §164.312 because no element of the regulation related to the requirement describes different or additional obligations than those described in the requirement. In contrast, consider Requirement B:

> **Requirement B:** iTrust shall allow an authenticated user to change their user ID and password. [Traces to §164.312(a)(1) and §164.312(a)(2)(i)]

Requirement B does not meet or exceed its legal obligations outlined in §164.312 because it describes a situation where it is possible for users to end up with the same identifying name or number, which violates §164.312(a)(2)(i). You may refer only to this document and to the legal text provided. Please indicate which requirements are legally ready for implementation in software and which ones need further refinement on the table of requirements provided in this packet.

A.2 iTrust Requirements

These are the requirements used in the user study. For more information on iTrust and on how these requirements were specified, please see Chapters 3, 4, and 5.

A.2.1 Authentication

R-1: iTrust shall support user authentication by requiring a user to input their user ID and password. [Traces to §164.312(a)(1), §164.312(a)(2)(i), and §164.312(d)]

R-2: When an authenticated user has remained idle for the duration of the timeout limit, iTrust shall automatically deauthenticate that user and redirect to the login screen. [Traces

to §164.312(a)(2)(iii) and §164.312(d)]

R-3: iTrust shall default to a timeout limit of five minutes and shall, at any time after installation, allow an authenticated system administrator to customize the timeout limit. [Traces to §164.312(a)(1), §164.312(a)(2)(iii), and §164.312(d)]

R-4: Each time a user is authenticated, deauthenticated, or denied authentication, iTrust shall create a record of the activity in the access control log. [Traces to §164.312(a)(1), §164.312(b), and §164.312(d)]

R-5: For all deauthentication records in the access control log, iTrust shall denote whether a user was deauthenticated at their own request or as a result of a timeout. [Traces to §164.312(a)(1), §164.312(b), and §164.312(d)]

R-6: iTrust shall ensure that access to protected health information is granted only to authorized individuals. [Traces to §164.312(a)(1) and §164.312(d)]

A.2.2 Account Information

R-7: When a user's account information is altered, iTrust shall email the users affected with a description of the alterations made. [Traces to §164.312(b)]

R-8: When patient account information is altered, iTrust shall email the personal representatives of the affected patients with a description of the alterations made. [Traces to §164.312(b)]

R-9: When a patient medical record is updated, iTrust shall email the patient affected according to the stored preferences regarding email for that patient. [Traces to §164.312(c)(1) and §164.312(c)(2)]

R-10: When a patient medical record is updated, iTrust shall email the personal representatives for that patient according to the stored preferences regarding email for those personal representatives. [Traces to §164.312(c)(1) and §164.312(c)(2)]

R-11: When an employee record is updated, iTrust shall send an email to an employee's email address according to the stored preferences regarding email for that employee. [Traces to §164.312(b)]

R-12: iTrust shall allow an authenticated patient to specify a preference regarding whether they would like to receive emails detailing changes to their medical records. [Traces to §164.312(c)(1) and §164.312(c)(2)]

R-13: iTrust shall allow an authenticated personal representative to specify a preference regarding whether they would like to receive emails detailing changes to the medical records of those they represent. [Traces to §164.312(c)(1) and §164.312(c)(2)]

R-14: iTrust shall allow an authenticated employee to specify a preference regarding whether they would like to receive emails detailing changes to their employee records. [Traces to §164.312(b)]

A.2.3 Account Creation and Deletion

R-15: iTrust shall allow an authenticated system administrator to create a new employee account using demographic information for the new employee. [Traces to §164.312(a)(1)]

R-16: iTrust shall allow an authenticated system administrator to delete invalid employee account. [Traces to §164.312(a)(1)]

R-17: iTrust shall allow an authenticated employee to create a new patient account using demographic information for the new patient. [Traces to §164.312(a)(1)]

R-18: Whenever a new employee account is created, iTrust shall create a record of the employee account created and the system administrator who created it in the access control log. [Traces to §164.312(b)]

R-19: Whenever an invalid employee account is deleted, iTrust shall create a record of the invalid employee account and the system administrator who deleted it in the access control log. [Traces to §164.312(b)]

R-20: Whenever a new patient account is created, iTrust shall create a record of the patient account created and the employee who created it in the access control log. [Traces to §164.321(b)]

R-21: iTrust shall store all deleted employee records for a period of 30 days before destroying them in an unrecoverable fashion. [Traces §164.312(a)(1) and §164.312(c)(1)]

R-22: iTrust shall store all deleted patient records for a period of 30 days before destroying them in an unrecoverable fashion. [Traces §164.312(a)(1) and §164.312(c)(1)]

R-23: iTrust shall allow an authenticated administrator to view or restore any record found in the deleted records store within the 30 day period.[Traces §164.312(a)(1) and §164.312(c)(1)]

A.2.4 Emergency Situations

R-24: iTrust shall allow any authenticated employee to generate a printable emergency report for a patient's medical record. [Traces to §164.312(a)(2)(ii)]

R-25: Each time a printable emergency report for a medical record is generated, iTrust shall create a record in the emergency access log detailing the time it was generated, the authenticated employee who generated it, and the patient to which the record pertains. [Traces to §164.312(a)(2)(ii)]

R-26: Each time a printable emergency report for a medical record is generated, iTrust shall email a notification to the patient to whom the record pertains and to all of that patient's personal representatives. [Traces to §164.312(b)]

R-27: Each time a printable emergency report for a medical record is generated, iTrust shall include a cryptographic hash of the authenticated employee user ID and the patient user ID at the end of the report. [Traces to §164.312(b)]

A.2.5 Encryption and Backups

R-28: When electronically transmitting medical records to other healthcare institutions, iTrust shall encrypt patient account information. [Traces to §164.312(e)(1)]

R-29: When storing medical records in its database, iTrust shall encrypt them using 256- bit AES encryption. [Traces to §164.312(e)(2)(ii)]

R-30: iTrust shall allow an authenticated system administrator to backup all encrypted patient records to a secondary database. [Traces to §164.312(e)(2)(ii)]

R-31: iTrust shall allow an authenticated system administrator to schedule regular, automatic backups of all encrypted patient records to a secondary database. [Traces to §164.312(e)(2)(ii)]

A.3 iTrust Glossary

Access Control Log: A record of transactions used to audit access to and control of protected health information.

Employee Account: Any information that can be linked to a specific employee user ID. Invalid Employee Account: An employee account that is no longer authorized to access any aspect of iTrust.

Medical Record: A set of protected health information that can be linked to a specific individual through their user ID.

Password: A secret alphanumeric character string that is used to authenticate a user account.

Patient Account: Any information that can be linked to a specific patient user ID, including protected health information.

Personal Representative: A patient account that is authorized to make medical decisions for another patient.

Protected Health Information: Any information about health status, provision of health care, or payment for health care that can be linked to a specific individual.

System Administrator: An employee account that is authorized to perform system-level maintenance and administration but not authorized to view medical records or other protected health information.

User ID: A unique identifier that represents an individual user account.

A.4 Traceability Matrix

The following table displays all traceability links described in the requirements specification. All HIPAA sections are within §164.312. For example, (a)(2)(ii) refers to HIPAA §164.312(a)(2)(ii). An X indicates that a traceability link exists. A blank space indicates that no traceability link exists.

Table A.1: User Study Traceability Matrix

Rqmt	(a)(1)	(a)(2)(i)	(a)(2)(ii)	(a)(2)(iii)	(a)(2)(iv)	(b)	(c)(1)	(c)(2)	(d)	(e)(1)	(e)(2)(i)	(e)(2)(ii)
R-1	X	X							X			
R-2				X					X			
R-3	X			X					X			
R-4	X					X			X			
R-5	X					X			X			
R-6	X								X			
R-7						X						
R-8						X						
R-9							X	X				
R-10							X	X				
R-11						X						
R-12							X	X				
R-13							X	X				
R-14						X						
R-15	X											
R-16	X											
R-17	X											
R-18						X						
R-19						X						
R-20						X						
R-21	X						X					
R-22	X						X					
R-23	X						X					
R-24			X									
R-25			X									
R-26						X						
R-27						X						
R-28										X		
R-29												X
R-30												X
R-31												X

A.5 HIPAA § 164.312 Technical Safeguards

A covered entity must, in accordance with § 164.306:

(a) (1) Standard: Access control. Implement technical policies and procedures for electronic information systems that maintain electronic protected health information to allow access only to those persons or software programs that have been granted access rights as specified in § 164.308(a)(4).

(2) Implementation specifications:

(i) Unique user identification (Required). Assign a unique name and/or number for identifying and tracking user identity.

(ii) Emergency access procedure (Required). Establish (and implement as needed) procedures for obtaining necessary electronic protected health information during an emergency.

(iii) Automatic logoff (Addressable). Implement electronic procedures that terminate an electronic session after a predetermined time of inactivity.

(iv) Encryption and decryption (Addressable). Implement a mechanism to encrypt and decrypt electronic protected health information.

(b) Standard: Audit controls. Implement hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use electronic protected health information.

(c) (1) Standard: Integrity. Implement policies and procedures to protect electronic protected health information from improper alteration or destruction.

(2) Implementation specification: Mechanism to authenticate electronic protected health information (Addressable). Implement electronic mechanisms to corroborate that electronic protected health information has not been altered or destroyed in an unauthorized manner.

(d) Standard: Person or entity authentication. Implement procedures to verify that a person or entity seeking access to electronic protected health information is the one claimed.

(e) (1) Standard: Transmission security. Implement technical security measures to guard against unauthorized access to electronic protected health information that is being transmitted over an electronic communications network.

(2) Implementation specifications:

(i) Integrity controls (Addressable). Implement security measures to ensure that electronically transmitted electronic protected health information is not improperly modified without detection until disposed of.

(ii) Encryption (Addressable). Implement a mechanism to encrypt electronic protected health information whenever deemed appropriate.