

Triage for Legal Requirements

Aaron K. Massey and Annie I. Antón
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
{akmassey, aianton}@ncsu.edu

October 11, 2010

Abstract

The high cost of non-compliance with laws and regulations that govern software systems makes legal requirements prioritization crucial. In addition, software design, expense, and time constraints all influence how requirements are prioritized. Prioritizing requirements derived from laws and regulations can be untenable using traditional pairwise requirements prioritization techniques because the number of pairwise comparisons grows quadratically with the number of requirements. In this paper, we introduce legal requirements triage—a technique used to subdivide a requirements set into three subsets: (a) implementation ready legal requirements; (b) legal requirements that require further refinement; and (c) non-legal requirements. Legal requirements triage supplements a traditional pairwise requirements prioritization by focusing analysts on implementation ready legal requirements to reduce the number of pairwise comparisons. Herein, we discuss a case study in which we applied these techniques to prioritize 75 functional requirements for the iTrust Medical Records System, an open-source electronic health records system that must comply with the U.S. Health Insurance Portability and Accountability Act (HIPAA). Our study shows that we were able to reduce significantly the number of pairwise comparisons.

1 Introduction

In the United States, the Health Insurance Portability and Accountability Act (HIPAA)¹ regulates Electronic Health Records (EHR) systems. HIPAA is designed to protect patient medical information. HIPAA includes serious penalties for non-compliance and raises the importance of solid requirements engineering and software design for legally compliant software. For non-criminal infractions, violators could be fined up to \$25,000 per year per violation.

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

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 2009, Congress amended HIPAA with the HITECH Act, which was passed as a part of the American Recovery and Reinvestment Act², President Obama's first major legislative initiative upon taking office. These changes include 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.

Tight legal compliance timeframes and substantial fines and penalties for non-compliance make lean or agile software development techniques appropriate. At the 2009 International Requirements Engineering Conference, Dave West of Forrester Research argued in his keynote that requirements engineering researchers should focus on improving or adopting agile methods and techniques because they are becoming more mainstream. Agile methods are more implementation-oriented, whereas requirements engineering methods are more documentation-oriented [25]. In addition, agile methods focus on shorter iteration cycles and more direct customer collaboration [27]. Although primarily considered an element of agile development, shorter iteration cycles can be applied to other iterative development methodologies and leads to earlier implementation and deployment [27]. Performing triage for legal requirements supports both agile and iterative software development processes for systems that must comply with the law.

To begin implementation, developers need a set of implementation ready requirements. Requirements triage is the process of determining which requirements should be implemented given the available time and resources [8]. In the context of legal requirements, implementation order should be influenced by legal domain knowledge [24]. Legal requirements triage allows requirements engineers to focus on requirements with legal implications early in the engineering process, avoid expensive and unnecessary refactoring, and demonstrate legal due diligence by incorporating laws and regulations efficiently throughout the engineering effort.

In this paper, we propose a legal requirements triage technique—a technique 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. Our technique 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 [11] and *k-means* clustering (also called Lloyd's algorithm) [16] to divide the legal requirements into two subsets: implementation ready and needing further refinement. The non-legal requirements comprise the

²Pub. L. No. 111-5, 123 Stat. 115 (2009)

third and final output requirements set. We employ the iTrust Medical Records System, an open-source EHR system—a system that must comply with the HIPAA—as the subject of our case study [32].

The remainder of this paper is organized as follows. Section 2 describes related work and background information for this research. Section 3 outlines our legal requirements triage algorithm. Section 4 describes our methodology for applying our requirements triage technique. Section 5 describes our experiences in applying this methodology in a case study. Section 6 describes the threats to validity of our case study. Section 7 summarizes our work and discusses potential future work in this area.

2 Related Work

We focus on two primary areas of related work: (A) legal requirements and (B) requirements triage and prioritization.

2.1 Legal Requirements

Bringing software into legal compliance is a challenging software engineering concern [24]. 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 [24]. Currently, engineers rely on abbreviated summaries of legal texts that attempt to make dense legal terminology more accessible for requirements engineers to achieve legal compliance [24].

In addition, laws and regulations may continue to be amended by administrative agencies or interpreted in judicial proceedings for years after their initial passage [24]. For HIPAA, the HHS has discretion on how frequently to updates regarding healthcare information technology can occur. Legal norms suggest that the HIPAA regulations could be revised as often as every year [2]. 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. Pairwise requirements prioritization techniques are at a disadvantage when used with legal requirements because the prioritization must be completely recalculated when a change in law occurs and the calculation effort grows quadratically with the number of requirements.

Some researchers focus on building logical models of regulation directly from actual legal texts [6, 23, 18, 21]. 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 [21].

Researchers are exploring automated processes for generating traceability links for regulatory compliance [3, 7]. Cleland-Huang et al. use a machine learning approach to automatically generate traceability links for regulatory compliance [7]. 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 [3]. They intentionally trace requirements to higher level segments of the legal text rather than to individual subsections [3]. To complete the process, a requirements engineer must be involved manually [3]. 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 paper, even though they do not currently produce such a mapping.

Breaux developed an XML markup of legal texts that formally describes the structure of the legal text [4]. Breaux applied this XML markup to HIPAA for the purposes of supporting requirements acquisition [4]. In this paper, 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.'

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 [24]. 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 [4, 5, 6]. This hierarchy can identify which requirement takes precedence in cases of legal exceptions [5]. 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 paper, we employ legal exceptions as a metric for performing requirements triage rather than legal precedence.

2.2 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 [8]. 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 [8, 30, 15]. 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 [30, 15]. 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 [9, 15]. Their system, called Pirogov, uses an automated clustering algorithm that groups requirements [9, 15]. A requirements engineer then prioritizes these groups and creates weightings for the categories of requirements [9, 15]. A prioritization algorithm then creates a final prioritization [9, 15]. In this paper, 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 implementation ready legal 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 [13]. 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 [11]. Pairwise comparison techniques involve ranking the relative priorities of each pair of requirements in the system [11]. 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 [13]. Avesani et al. identify Analytic Hierarchy Process (AHP) as the reference method for pairwise requirements prioritization in case study-based requirements engineering research [1], and numerous researchers adapted AHP for requirements prioritization [10]. AHP provides heuristics for prioritizing elements from multiple domains based on multiple criteria[29]. In addition, AHP has mathematical support for ranking elements using priority vectors[29]. In this paper, we discuss the use of pairwise prioritization in Section 4.4.

The numeral assignment prioritization technique is a ranking system in which requirements engineers assign numeral values to represent the priority of software requirements [11]. One such approach consists of assigning a discrete value to a requirement on a scale from 1 (optional) to 5 (mandatory) [11]. Karlsson showed that numeral assignment prioritization techniques take more time and produce less reliable prioritizations than pairwise comparison techniques [11]. 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[19]. In this paper, we expand on our prior work in two ways: (1) We introduce four new metrics and maintain the four previously developed metrics for assigning numeral values to the requirements; and (2) We provide tool support to automate the process of calculating these values.

3 Legal Requirements Triage Algorithm

In this section we describe a legal requirements triage algorithm that takes as an input a legal text and a set of requirements previously mapped to specific sections of a legal text as described in prior work [20, 19]. The algorithm produces as outputs three sets of requirements: (a) implementation ready legal requirements; (b) legal requirements that need further refinement; and (c) non-legal requirements.

Legal texts are hierarchical documents. Without knowing the meaning of a legal text, we can still infer some meaning from its structure. Consider the structured text in Figure 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 this algorithm, 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.

In the remainder of this section we discuss the calculation of triage scores for each requirement in subsection 3.1 and the use of *k-means* clustering to identify which requirements are implementable based on their triage scores in subsection 3.2.

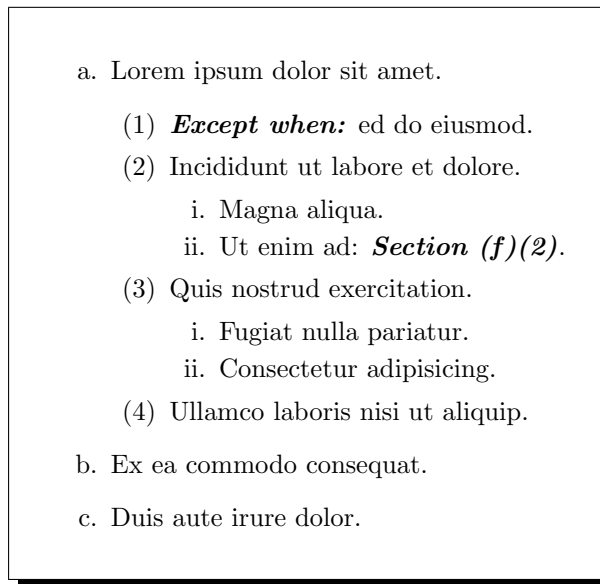
- 
- a. Lorem ipsum dolor sit amet.
 - (1) ***Except when:*** ed do eiusmod.
 - (2) Incidunt ut labore et dolore.
 - i. Magna aliqua.
 - ii. Ut enim ad: ***Section (f)(2)***.
 - (3) Quis nostrud exercitation.
 - i. Fugiat nulla pariatur.
 - ii. Consectetur adipiscing.
 - (4) Ullamco laboris nisi ut aliquip.
 - b. Ex ea commodo consequat.
 - c. Duis aute irure dolor.

Figure 1: Sample Hierarchical Legal Text

3.1 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.

Our algorithm provides three categories of legal requirements triage metrics: *Dependency*, *Complexity*, and *Maturity*. Dependency metrics estimate the extent to which a requirement may be dependent on other, unknown requirements. Complexity metrics estimate the engineering effort required to implement a particular requirement. Maturity metrics estimate a requirement's readiness for implementation. These metrics categories form the final triage score outlined in Equation 1 below. Definitions for the metrics themselves can be found in Table 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.

<p>Name: Sample Requirement</p> <p>Description: Occaecat cupidatat non.</p> <p>Legal Subsections: (a)(1) and (a)(2)(ii)</p>
--

Figure 2: Sample Requirement

Table 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.

$$T = D + C - M \quad (1)$$

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

$$C = \frac{C_W}{4} \cdot \left(\frac{N_W}{W_T} + \frac{N_S}{S_T} + \frac{S_C}{S_T} + \frac{E}{E_T} \right) \quad (3)$$

$$M = \frac{M_W}{2} \cdot \left(\frac{S_D}{S_{D_T}} + S_F \right) \quad (4)$$

Dependency metrics estimate the extent to which a requirement may depend on other, unknown requirements. Equation 2 displays the complete calculation of the Dependency metrics. 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 a mapping of requirements to specific sections of a legal text is provided as an input to our technique. To calculate S_M count each mapping of a requirement to a subsection once. Consider the sample requirement from Figure 2, which maps to sections (a)(1) and (a)(2)(ii) from Figure 1; its S_M score is two. 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, $C = 1$ for the requirement from our previous example. It is important to note that neither S_M or C indicate actual, known dependencies; instead, these metrics indicate potential dependencies.

Recall that metrics are normalized by the algorithm and weighted according to the weightings provided by the requirements engineer performing the triage. 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. Recall that the Dependency metrics are weighted to enable easier interpretation of the triage scores. Since there are two Dependency metrics, the Dependency weighting (D_W) is divided by two and applied to both dependency metrics. Thus, the maximum possible dependency score for any provided value of D_W is D_W .

Complexity metrics estimate the engineering effort required to implement a particular requirement. Equation 3 displays the complete calculation of the Complexity metrics. In the Complexity category, the triage algorithm evaluates requirements based on four 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 [17]. For the sample requirement in Figure 2, $N_W = 22$ and $N_S = 5$. 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. For the sample

requirement, S_C is two. 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 [5, 4]. Often the structure of an exception is similar to a case statement in a procedural programming language [5, 4]. 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 an exception. As a result, $E = 1$ for our sample requirement.

As with the Dependency metrics, the Complexity metrics are then normalized and weighted according to the weightings provided by the requirements engineer performing the triage. The N_W and N_S metrics are 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. Because there are four Complexity metrics, the Complexity weighting (C_W) is divided by four and then multiplied by each normalized metric as shown in Equation 3. Thus, the maximum possible complexity score for any provided value of C_W is C_W .

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 requirement as stated in the legal text and requirements that represent higher level legal statements. 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 represent the requirements needed for legal compliance. Thus, it may be ready for further prioritization and implementation. Without maturity metrics, this requirement would have consistently high triage scores and may be classified as needing refinement for many iterations.

$$S_D = SC_D - S_M \quad (5)$$

$$S_F = \frac{(S_M + S_O)}{S_{CH}} \quad (6)$$

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 5) and the subsection fulfillment percentage (S_F , found in Equation 6).

Subsection depth is a maturity estimate because deeper level sections of a hierarchical legal text tend to be more specific than higher level subsections.

The deeper the requirement maps within the legal text, the more specific it is. 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). For example, 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 has a depth of three. Since $SC_D = 3$ and $S_M = 2$ for our sample requirement, S_D is one.

Subsection fulfillment percentage is a maturity estimate because legal requirements are less likely to be omitted as a higher percentage of legal subsections can be traced to some requirement. Subsection fulfillment (S_F) percentage represents the percentage of unmapped subsections in the highest-level sections to which a requirement maps. The highest-level section for a reference is the section with a depth of one. Consider the mapping (a)(2)(ii). Since (a) has a depth of one, it is the highest-level section for that reference. 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 other requirements in the highest-level sections mapped by the requirement (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 $\frac{2+1}{9} = \frac{1}{3}$.

As with both the Dependency and Complexity metrics, we normalize the Maturity metrics and apply the Maturity weighting (M_W) to them. 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. Since there are two Maturity metrics, we divide M_W by two and then multiply it against the two Maturity metrics as shown in Equation 4. Thus, the maximum possible maturity score for any provided value of M_W is M_W .

3.2 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. To get two groups, we set $k = 2$. 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 *k-means* algorithm will always produce two groups, even if all the requirements are actually implementation ready or needing 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. 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 [19]. However, since each development situation may have different time constraints, budgets, and resources, the final decision to accept the groups as distinct may reasonably be made using other criteria.

4 Methodology

In order 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 3. Figure 3 shows a screenshot of Red Wolf with six of the requirements from this case study.

We now describe our four-step methodology for using Red Wolf for legal requirements triage. Figure 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 (discussed in 4.1), and then enter the requirements into our legal requirements management tool second (discussed in 4.2). Third, we choose weightings for the triage algorithm and

Name	Description	User	Legal Subsection	
Requirement 1	Trust 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	Trust 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	Trust shall allow an employee, using their authorized account, to destroy invalid patient records.	Aaron	None.	Show Edit Destroy
Requirement 4	Trust 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	Trust 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	Trust 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 3: Red Wolf Requirements Management Screenshot

conduct the triage. Fourth, we discuss options for pairwise prioritization of the final requirements groups.

4.1 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 (GPO) hosts many regulations, including HIPAA, online. Figure 5 shows a sample section of HIPAA as it appears in on the GPO website³.

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

³http://www.access.gpo.gov/nara/cfr/waisidx_07/45cfr164_07.html

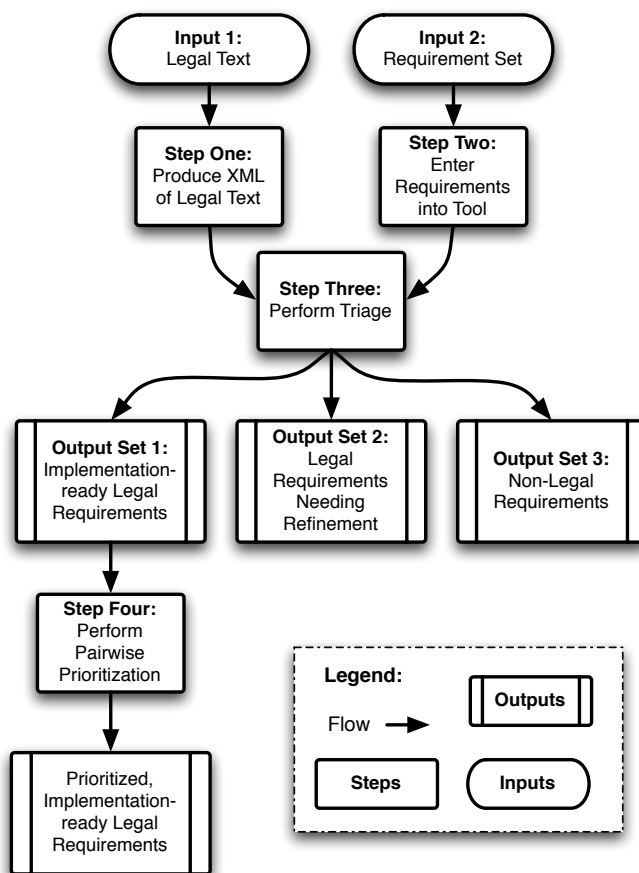


Figure 4: Methodology Overview

§ 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 5: HIPAA §164.312(a)(1)

actual XML markup automatically. As a result, a requirements engineer must manually verify and, if necessary, complete the markup.

```
<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 6: Sample XML Markup for HIPAA §164.312(a)(1)

Other researchers have employed XML representations of legal texts [4, 3, 14]. Herein, we modified Breaux’s XML format [4] 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 [4, 14]. Figure 6 depicts the same sample legal text found in Figure 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.

4.2 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 [4, 5, 20, 21]. 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 [4, 5, 21]. In addition, Massey et al. outline a procedure for mapping existing requirements to the relevant portions of a legal text [20]. Optionally, a requirements engineer may choose to use an automated traceability technique [3, 7] and manually complete the mapping to specific subsections of the legal text using one another technique [4, 5, 20, 21].

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 [20]. However, the only attributes required by Red Wolf are Name, Description, and Legal Subsections. The *Name* attribute

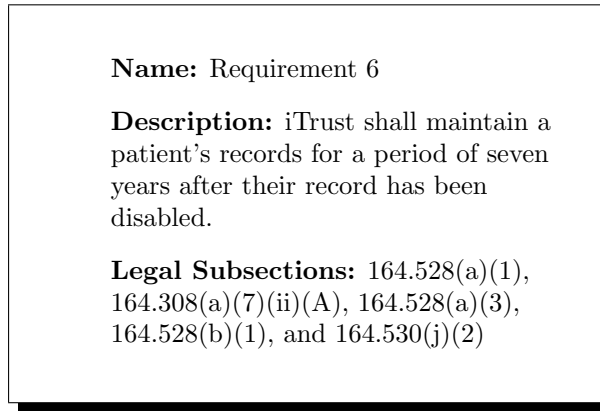


Figure 7: Sample Case Study Requirement

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 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 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.

4.3 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 3.1 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.

4.4 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 [9, 10, 11, 12, 13, 15, 26, 28, 1].

5 Case Study 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 [20]. This evaluation produced a total of 73 requirements, of which

Table 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

63 were functional and 10 were non-functional [20]. To these 63 functional requirements we add 12 additional functional requirements for iTrust identified by Maxwell and Antón [21]. 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 good triage scores near zero. Using these weightings, Red Wolf produced 46 implementation ready requirements as shown in Table 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.

6 Threats to Validity

We developed our legal requirements triage technique by conducting an exploratory case study in which we analyzed the iTrust requirements. Internal validity is not a concern for exploratory case studies [33]. Thus, we only discuss threats from construct validity, external validity, and reliability.

Construct validity refers to the appropriateness and accuracy of the measures and metrics used for the concepts studied [33]. This case study relies on two sources of requirements for the iTrust Medical Records System: those produced by Massey et al.[20] and those produced by Maxwell and Antón[22]. Using multiple sources of requirements improves the appropriateness and accuracy of our results. In this study, we also addressed construct validity by strictly

adhering to the methodology in Section 4 and by submitting draft reports of this work for review to our colleagues at The Privacy Place and North Carolina State University.

External validity refers to the ability to generalize the findings of a case study to other domains [33]. Although no healthcare provider is currently using iTrust in industry, healthcare professionals have consulted with the developers of iTrust from the beginning [20]. In addition, other researchers are using iTrust as a subject of study for requirements engineering research [7]. Although HIPAA represents a single legal domain, we have consulted with legal experts and determined that HIPAA is significantly similar in form to other executive branch administrative regulations.

Reliability refers to the ability of other researchers to repeat a case study [33]. After developing our triage algorithm, we developed the Red Wolf tool that supports and automates much of the process. We are currently in the process of licensing Red Wolf as an open source requirements management tool. This would allow others to repeat our case study.

7 Summary and Future Work

Quickly and accurately triaging and prioritizing legal requirements is increasingly important for software design of systems that must comply with the law. In this paper, we present a tool supported legal requirements triage technique. Our technique takes as inputs a legal text and a set of software requirements with traceability links to specific subsections of the legal text to which it must comply. It produces as output three sets of requirements that can be used for further prioritization: (a) implementation ready legal requirements; (b) legal requirements that require further refinement; and (c) non-legal requirements. We also developed Red Wolf, a tool that automates our legal requirements triage algorithm. Using Red Wolf in an experiential case study, we triaged 75 requirements for iTrust, an EHR system that must comply with HIPAA. Our case study showed that our legal requirements triage technique reduced the number of pairwise comparisons needed for a complete prioritization of implementation ready legal requirements to 37.3% of the number needed for a complete prioritization.

Our technique is designed to be used in an iterative software development processes, including various agile development methodologies. Such processes require quick identification of implementable requirements for both design and implementation. Currently, the only manual step is producing a custom XML markup of the legal text, which only has to be done once, prior to the first iteration. In addition, this step can be semi-automated through tool support as described in Section 4.1.

In this paper, we assume a mapping of requirements to specific subsections of a legal text as part of our input requirements set. However, in a real-world environment, these links must be generated. Techniques exist to do this manually, but it may be possible to automate this step. For example, techniques for automatically identifying traceability links for regulatory environments may be

integrated with the triage algorithm we present in this work [3, 7]. Currently, these techniques do not map a requirement directly to specific sections of a legal text.

Producing an XML-formatted legal text is the other manual component for real-world engineering applications of this technique. However, as we produced the XML-formatted legal text, we recorded a set of regular expressions that can match the changes needed to partially automate this process in the future. In addition, Data.gov⁴, an open government initiative, recently released XML-formatted versions of numerous U.S. legal texts, including the Code of Federal Regulations. We are currently modifying Red Wolf to support this official XML-format and eliminate the manual markup element described in Section 4.1.

Finally, because our technique is designed to improve management of evolving legal requirements we plan to conduct additional studies involving amendments to the HIPAA regulations, including the recent final ruling from HHS on the HITECH Act[31].

Acknowledgment

This work was supported by NSF ITR Grant #522931 and NSF Cyber Trust Grant #0430166. The authors would like to thank J. Maxwell, J. Young, and the members of The Privacy Place for their feedback on early drafts of this paper.

References

- [1] 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.
- [2] K. Beaver and R. Herold. *The Practical Guide to HIPAA Privacy and Security Compliance*. Auerbach Publications, 2004.
- [3] 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.
- [4] T. D. Breaux. *Legal Requirements Acquisition for the Specification of Legally Compliant Information Systems*. PhD thesis, North Carolina State University, 2009.
- [5] T. D. Breaux and A. I. Antón. Analyzing regulatory rules for privacy and security requirements. *IEEE Transactions on Software Engineering*, 34(1):5–20, Jan. 2008.

⁴<http://www.data.gov/>

- [6] T. D. Breaux, M. W. Vail, and A. 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.
- [7] 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.
- [8] A. Davis. The Art of Requirements Triage. *Computer*, 36(3):42–49, Mar 2003.
- [9] C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski. Towards Automated Requirements Prioritization and Triage. *Requirements Engineering*, 14(2):73–89, 06 2009.
- [10] A. Herrmann and M. Daneva. Requirements Prioritization Based on Benefit and Cost Prediction: An Agenda for Future Research. *16th IEEE International Requirements Engineering*, pages 125–134, Sept. 2008.
- [11] J. Karlsson. Software Requirements Prioritizing. *Proceedings of the Second International Conference on Requirements Engineering*, pages 110–116, Apr 1996.
- [12] J. Karlsson and K. Ryan. A Cost-Value Approach for Prioritizing Requirements. *IEEE Software*, 14(5):67–74, Sep/Oct 1997.
- [13] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements. *Information and Software Technology*, 39(14-15):939 – 947, 1998.
- [14] S. Kerrigan and K. 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.
- [15] P. Laurent, J. Cleland-Huang, and C. Duan. Towards Automated Requirements Triage. In *15th IEEE International Requirements Engineering Conference*, pages 131–140, Oct. 2007.
- [16] S. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.
- [17] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, MA, 1999.
- [18] F. Massacci, M. Prest, and N. 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.

- [19] A. K. Massey, P. N. Otto, and A. I. Antón. Legal Requirements Prioritization. *Proc. of the 2nd Intl. IEEE Workshop on Requirements Engineering and the Law*, 2009.
- [20] A. K. Massey, P. N. Otto, L. J. Hayward, and A. I. Antón. Evaluating Existing Security and Privacy Requirements for Legal Compliance. *Requirements Engineering*, 2009.
- [21] J. C. Maxwell and A. I. Antón. Developing Production Rule Models to Aid in Acquiring Requirements from Legal Texts. *17th IEEE International Requirements Engineering Conference*, pages 101 –110, Aug. 31-Sept. 4 2009 2009.
- [22] J. C. Maxwell and A. 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.
- [23] M. J. May, C. A. Gunter, and I. Lee. Privacy APIs: Access Control Techniques to Analyze and Verify Legal Privacy Policies. *Proceedings of the Computer Security Foundations Workshop*, pages 85 – 97, 2006.
- [24] P. N. Otto and A. I. Antón. Addressing Legal Requirements in Requirements Engineering. *15th IEEE International Requirements Engineering Conference*, pages 5–14, 15-19 Oct. 2007.
- [25] F. Paetsch, A. Eberlein, and F. Maurer. Requirements Engineering and Agile Software Development. *Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 308 – 313, June 2003.
- [26] 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.
- [27] Z. Racheva, M. Daneva, and L. Buglione. Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products. *Second International Workshop on Software Product Management*, pages 49 –58, Sept. 2008.
- [28] 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.
- [29] T. L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.
- [30] E. Simmons. Requirements Triage: What Can We Learn From a “Medical” Approach? *IEEE Software*, 21(4):86–88, July-Aug. 2004.

- [31] U.S. Department of Health and Human Services. Medicare and medicaid programs; electronic health record incentive program: Final rule. *U.S. Federal Register*, 2010.
- [32] L. Williams, T. Xie, and A. Meneely. The iTrust Medical Records System, <http://agile.csc.ncsu.edu/iTrust/wiki/dokuwiki.php>. September 2008.
- [33] R. K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications, 3rd edition, 2003.