iTrust Electronic Health Care System: A Case Study

Andrew Meneely, Ben Smith, Laurie Williams

Department of Computer Science
North Carolina State University
890 Oval Drive, Engineering Building 2, Room 3272
Campus Box 8206
Raleigh, NC 27695-8206 USA
{apmeneel, bhsmith3, lawilli3}@ncsu.edu

1 Introduction

Electronic health record (EHR) systems present a formidable "trustworthiness" challenge because people's health records, which are transmitted and protected by these systems, are just as valuable to a myriad of attackers as they are to health care practitioners. Major initiatives in EHR adoption and increased sharing of health information raise significant challenges for protecting the privacy of patients' health information.

The United States is pursuing the vision of the National Health Information Network (NHIN) in which the electronic health records of the American people are passed between sometimes-competing health care providers. The American Recovery and Reinvestment Act of 2009 (ARRA) [1] provides $34 billion of incentives to health care providers to deploy a government-approved EHR. The ARRA will, by 2014, impose penalties on those who do not. As a result, the use of EHR systems is likely to proliferate in the US in the next four years.

Dr. Laurie Williams created iTrust in 2005 as a course project for undergraduates in North Carolina State University's Software Engineering course. iTrust is intended as a patient-centric application for maintaining an EHR. An ideal health care system combines medical information from multiple sources to provide a summary or detail view of the history of a particular patient in a way that is useful to the health care practitioner.

iTrust is not intended to fulfill the requirements set forth to be approved by the government, nor is it intended for use by practitioners in the field of medicine. The

primary goal for the project is to provide software engineering students with a project with real-world relevance and enough depth and psychological complexity as to mimic industrial systems that students may encounter while working in the software industry. Additionally, iTrust provides an educational testbed for understanding the importance of security and privacy requirements. iTrust is particularly focused with maintaining the privacy standards set forth in the HIPAA Security and Privacy Rules [2].

The notion that a software developer's role is often to maintain, test, and refine software rather than creating it "from scratch" is a unique learning objective for students at North Carolina State. For the past five years, each semester students in the undergraduate software engineering course enhance the project deemed to be the best from the prior semester. Refactoring of iTrust by graduate students often occurs during the summer. As such, students must learn the code base of more than 10,000 lines of Java Server Page code to make required enhancements.

This chapter highlights the key pieces of iTrust's project artifacts that pertain to traceability and describes the project in detail. The version of iTrust we are describing in this chapter is v10.0, which was released in the August 8th, 2010, and built from requirements specification v18. The source code for this project, as well as all the artifacts we describe in this chapter are available from iTrust's homepage[1]. The iTrust project consists of the following artifacts:

- Source code, including:
  - Production source code (Java, Java Server Pages)
  - Automated test code
- Testing documents, including:
  - Black box test plan
  - Acceptance test plan
  - Test data
- Requirements, including sections describing:
  - System Roles
  - Use cases
  - Non-functional requirements and constraints
  - Data field formats
  - Use case tracing from requirements to JSP pages
- Traceability Matrix

The rest of this chapter is organized as follows. Section 2 focuses on iTrust as a project and how the team develops and maintains it. Section 3 describes an overview of the iTrust functionality. Section 4 describes the architecture and organ-

[1] http://realsearchgroup.com/iTrust

ziation of the iTrust system. Section 5 describes the traceability provided by the project's maintainers, and Section 6 summarizes the chapter.

2 iTrust Project

iTrust is an active team project for undergraduate students in North Carolina State University's Software Engineering course. Dr. Laurie Williams conceived the project in the Fall of 2005 and the project has been released to undergraduate and graduate students at North Carolina State for the following five years (10 semesters). As a part of their homework assignments, students in the undergraduate Software Engineering course as well as the graduate Software Testing course are required to perform maintenance and feature additions to iTrust.

In between semesters, the project administrators (typically graduate students) perform a "housekeeping" procedure. The graduate students spend approximately one to two weeks on housekeeping, and the procedure entails one or more of the following:

- **Updating the automated test plan**, which consists of improving the coverage and accuracy of JUnit and system-level integration testing.
- **Fixing or clarifying the documentation of the iTrust code**, which consists of Javadoc that explains the functionality and use of each Java class.
- **Discussions on the future of the project**, including possible architectural design changes, new decisions on technologies to use for testing, and other high-level decisions that would be infeasible during a semester.
- **Minor features**, which often involve removing or adding functionality that students have complained about but not changed, or functionality that would be required to prepare the system for assignments in the upcoming semester.
- **Cosmetic changes**, primarily involving editing the style sheets and Java Server Pages to improve the user interface of the system.
- **Refactoring**, which has often been major, involving a complete redesign of the system, or sometimes minor, such as implementing and redesigning a component of the system to be more amenable to future changes and development.

Table 1 presents measurements on the source lines of code and number of Java classes or JSP files that make up the iTrust code base.

Table 1: iTrust project size

Component	Number of Files	LOC#
Production Classes	226	14,570
Java Server Pages	135	12,942
Unit Tests	244	11,936
HTTP Tests	50	4,146

The iTrust requirements v18 contains 40 functional requirements, six non-functional requirements, and eight constraints. iTrust v11 was released for download from SourceForge on August 8th, 2010. Since students were the primary developers for iTrust, there has been no public feedback on the project, although the install base is rather large. Since this release date, iTrust v11 has been downloaded from SourceForge 394 times.

3 iTrust Functionality

We designed iTrust to be a patient-centric application for maintaining an electronic health record. An ideal health record combines medical information from multiple sources to provide a summary or detail view of the history of a particular patient in a way that is useful to the health care practitioner. iTrust is particularly focused with maintaining the privacy standards set forth in the HIPAA Security and Privacy Rules [2]. In addition to maintaining the patient's personal information and health history, iTrust maintains a comprehensive transaction log. The transaction log, which can be used for repudiation and to track the actual operational profile, contains 53 different high-level transaction types that include viewing patients' information, sending reminders, and adding a prescription. The patient can view a list of which health care professionals have viewed his or her medical information upon login. Also, iTrust has a focus on providing health care providers with dynamically determined information regarding a patient's chronic disease risk factors including diabetes and heart disease. Finally, iTrust allows a health care professional to view trend information about patients' causes of death. Often iTrust requirements are obtained from the US Department of Health and Human Services (HHS) use cases [http://www.hhs.gov/healthit/usecases/]; those that are obtained from HHS reference the use cases. The remaining requirements are developed in a creative process by the teaching staff, with the intent of covering the software engineering curriculum.

3.1 System Roles

iTrust contains eight roles in its role-based access control system. The role of a user determines their viewing and editing capabilities.

- **Patient**: When an American infant is born or a foreigner requests medical care, each is assigned a medical identification number and password. Then, this person's electronic records are accessible via the iTrust Medical Records system.
- **Administrator**: The administrator assigns medical identification numbers and passwords to LHCPs. [Note: for simplicity of the project, an administrator is added by directly entering the administrator into the database by an administrator that has access to the database.]
- **Licensed Health Care Professional (LHCP)**: A licensed health care professional that is allowed by a particular patient to view all approved medical records. In general, a patient does not know this non-designated health care professional, such as an emergency room doctor, and the set of approved records may be smaller than that granted to a designated licensed health care professional.
- **Designated Licensed Health Care Professional (DLHCP)**: A licensed health care professional that is allowed by a particular patient to view all approved medical records. Any LHCP can be a DLHCP to some patients (with whom he/she has an established relationship) and an LHCP to others (whom he/she has never/rarely seen before).
- **Emergency Responder (ER)**: Police, Fire, Emergency Medical Technicians (EMTs), and other medically trained emergency responders who provide care while at, or in transport from, the site of an emergency (referred to as “on site care providers” by Department of Health and Human Services Emergency Responder Electronic Health Record Use Case [3]).
- **Unlicensed Authorized Personnel (UAP)**: A health care worker such as a medical secretary, laboratory technician, case manager, care coordinator, or other authorized clerical-type personnel. An unlicensed personnel can enter and edit demographic information, diagnosis, office visit notes and other medical information, and can view records.
- **Personal Representative**: A person legally authorized to make health care decisions on an individual's behalf or to act for a deceased individual. When a person logs into iTrust, if he or she is a personal representative, they view their own records or those of the person/people they are representing. (For example, a mother is a personal health representative for her children and could choose herself and any one of her children upon logging into iTrust.)
- **Public Health Agent**: A person legally authorized view and respond to aggregated reports of adverse events.

- **Software Tester**: An information technology worker who tests the iTrust Medical Records system. Of particular interest to the software tester is the operational profile information which informs him/her of the frequency of use of the features of the system.

3.2 Patient-Centered Functionality

One of the unique characteristics of iTrust is its patient-centered functionality where patients can log into the system to view their own records and perform a variety of tasks.

The primary way of tracking care for a given patient is through office visits. An office visit represents a specific consultation with an LHCP on a specific date in a specific location. Various standardized health care codes are linked to office visits, including diagnoses, immunizations, procedures, prescriptions, and general demographics such as height and weight. The LHCP logs the information for a given office visit, and the patient can view the records for of his or her previous office visits. Patients can also take a satisfaction survey on the LHCP, which is aggregated for other patients in search for an LHCP.

In addition to office visit tracking, patients have access to several forms of auditability. iTrust takes data provenance very seriously, so all access and changes to patient records are permanently logged. Patients are presented with an activity feed upon logging in to iTrust, and can configure email alerts when their records have been accessed or changed.

Lastly, iTrust focuses on providing informative feedback to both patients and LHCPs. Patients are shown potential risk factors on their record, such as for diabetes or heart disease. High risk patients who have not had a recent office visit are also alerted. LHCPs can also request biosurveillance to detect potential epidemics. The epidemic detection feature uses statistical modeling to determine an abnormal number of diagnoses for a given location. Additionally, LHCPs can view cause-of-death trends for a given location.

The requirements document in iTrust is a use-case based specification as shown in Figure 1.

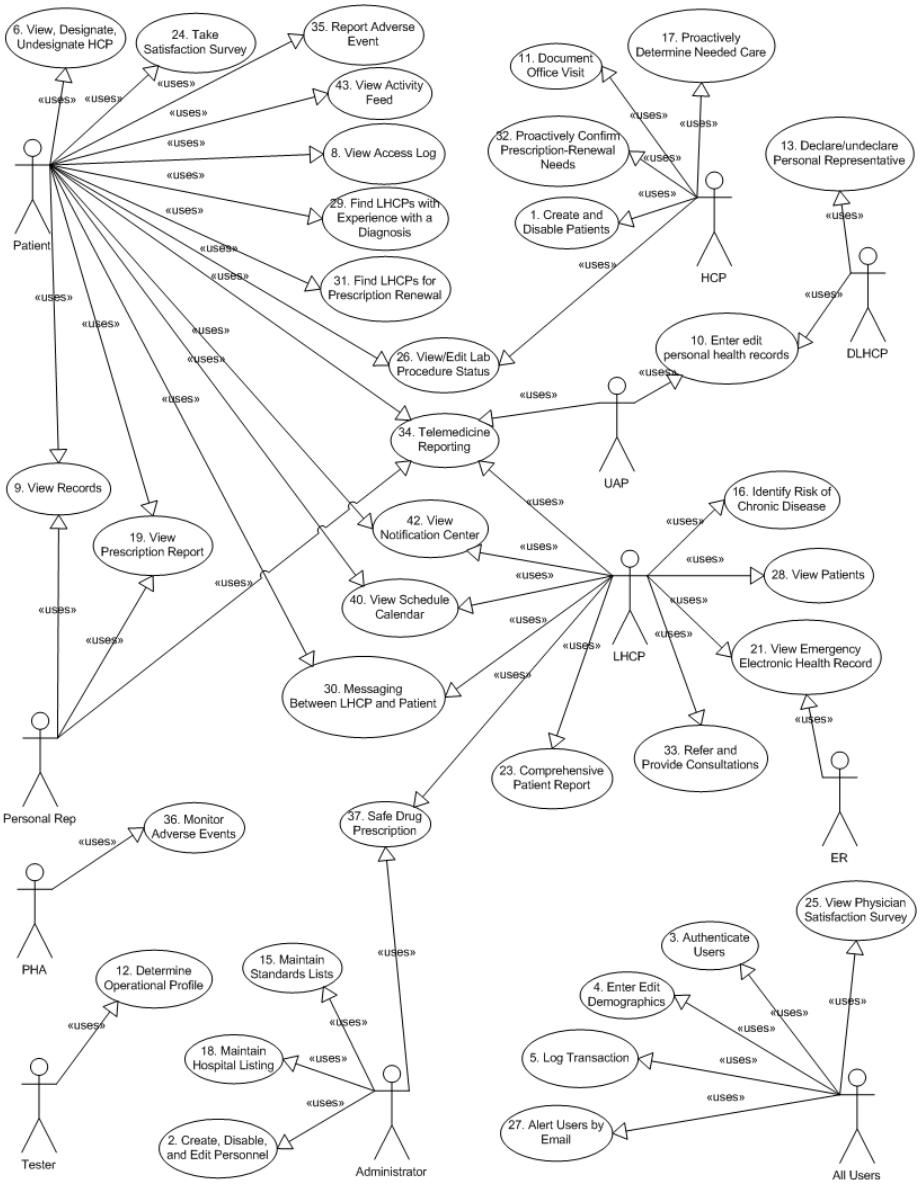


Figure 1. High-level Overview of the iTrust Use Cases

The requirements specification breaks down into the following sections:

- System Roles (described in Section 3.1)
- Use cases
- Non-functional requirements & constraints

- Data field formats

Each use case represents a small piece of functionality that students implemented in a two week iteration. The project administrators wrote the use cases in terms of the roles to imply the access controls surrounding the feature. Each use case has a precondition describing what conditions need to be met prior to accessing the feature (e.g. authentication). The main flow of the use case provides a high-level overview of the feature from the perspective of what the user does. The main flow of the use case references different sub-flows of the use case that provide added detail on the different events of the feature (e.g. the flow of events for when a patient is deceased). Lastly, each use case contains an alternative flow that describes the behavior of the feature outside of typical functionality (e.g. when the user enters wrong data). The requirements document also contains a reference from each sub-flow to the web page implementing that functionality. For an example of a use case, see the "Traceability in iTrust" section.

After the use cases, the rest of the document comprises of non-functional requirements and constraints. The non-functional requirements describe limitations that all features must adhere to. For example, all features must adhere to HIPAA standards. The constraints section covers the development process, such as the programming language and coding standards. iTrust was written in Java 1.5, and was designed to work with Tomcat v5.5.27 and MySQL 5.0.

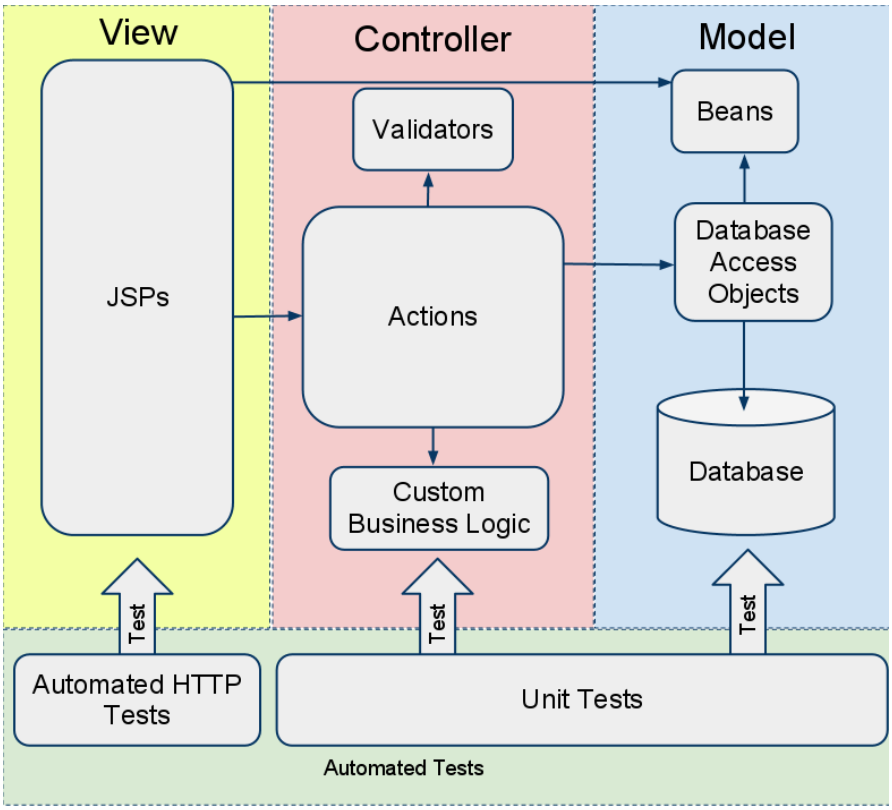
The data field formats section covers all of the inputs to the iTrust system and how the field can be validated. For example, the data fields section defines which characters are allowed in a patient's name. Many data fields are defined according to common health care standards. iTrust uses the following standard medical codes:

- ICD9CM for diagnoses
- CPT for procedures
- NDC for drug prescriptions

The iTrust requirements document is stored in a wiki format online. Storing the document in a wiki allows the requirements to be edited in a central location by authorized project maintainers. Each revision of the requirements document is retained so that the entire history of the document is preserved. Using the "diff" feature of the wiki also provides students with the ability to view what has recently changed in the requirements document without having to find changes manually.

4 iTrust Architecture

The iTrust source code is designed around the Model-View-Controller design pattern [4]. The goal of this organization is to separate the logic associated with the user interface (i.e. the "view") from the logic of the persistent storage (i.e. the "model"), while organizing most of the complex business logic in one place (i.e. the "controller"). In iTrust, the view is implemented in JavaServer Pages (JSPs), the controller is implemented in Java, and the model is implemented in SQL and Java. An overview of the iTrust architecture can be found in Figure 2.



4.1 Source Code Organization

View/JSPs. The primary purpose of the JSPs is to provide a web-based user interface. Each JSP contains Java code, HTML, and potentially some Javascript. Each

JSP has a one-to-one mapping to an action class. The JSP instantiate the Action class

Controller. The overall purpose of the controller in iTrust is to provide a bridge between the user experience and the persistent storage of the database. Most of the complex logic behind validating data, and processing database query results are implemented in the controller.

The primary classes in the controller are *action classes.* Representing specific functionality in iTrust, the purpose of an action class is to delegate responsibility to the appropriate classes. Action classes serve as thin mediators between the user interface and the database and business logic. The responsibilities of action classes include:

- Delegating any input validation to a Validator.
- Logging transactions for auditability
- Delegating any custom business logic, such as risk factor calculations
- Delegating database interaction
- Handling exceptions in a secure manner

In addition to action classes, the controller contains *validators.* The sole purpose of validators is to validate any input brought into the system. Since security is a high priority in iTrust, the validators operate on using both whitelist and blacklist techniques for checking input. Additionally, the validators are designed to aggregate all errors in input so that the user is given a full report of all the problems with the input.

Lastly, the controller contains several classes with *custom business logic*. The custom business logic classes are a set miscellaneous Java classes designed for specific use cases. For example, Use Case 14 (UC14) is a feature for determining if a patient is at risk for several risk factors. Many of the queries involved in UC14 are specific to certain risk factors (e.g. having a viral infections during childhood), so the UC14 requires its own business logic.

Model. The model involves the all of the logic related to persistent storage in iTrust. *Beans* are placeholders for data related to an iTrust entity (e.g. Patient). Beans have minimal functionality other than storing data. Other supporting classes load beans from database result sets, validate beans based on input, or any other custom logic needed.

The *relational database* is the sole storage mechanism for iTrust. The database stores all persistent information, including patient records, immunizations, office visits, and transaction logs. The database schema is defined by a set of custom scripts found in the source code tree. The database for iTrust does not contain any

foreign keys, as the students who use iTrust do not usually have a background in relational databases and would not be able to debug foreign key constraint violations.

To interact with the database, iTrust employs *database access objects* (DAOs). DAOs are Java objects that interact with the iTrust relational database. Action classes will typically use DAOs to store and query the database. DAOs provide a set of common queries required by the action classes so that database query logic is contained to the DAO layer. Every DAO assumes that the incoming data is valid and any exception is handled by the Action classes. Connections to the DAOs are handled by the DAOFactory, which is a singleton class that utilizes a database connection pool for better performance and reliability. By convention, each database entity maps to a single Database Access Object and a single Bean.

4.2 Testing Artifacts

iTrust contains both automated and manual testing artifacts. All testing artifacts are constantly maintained throughout the development process.

Black Box Test Plan. As a part of their assignments in the graduate and undergraduate software engineering courses, students are required to maintain and develop manual, black box test cases for the functionality of iTrust. The black box test plan is intended to be executed by a software tester using a web browser with no background in the project or how it can be used. The black box test plan is intended to cover each use case and sub-flow, including the exceptional or alternative flow cases.

A subset of the black box test plan is the acceptance test plan. The acceptance test plan is a set of black box, manual test cases that can be executed with passing results by the iTrust customer. When a new use case is developed for a course assignment, the instructors of the software engineering course develop an acceptance test case that corresponds to the use case. The acceptance test plan acts as a tool for grading how well students performed the assignment as well as providing a clarification of certain details of the specification that may be lacking from the requirements specification. Students are then responsible for adding additional black box tests for each use case flow.

Automated Unit Tests. The goal of the unit tests is to test individual iTrust functionality at the Java class level. Students are expected to test both regular functionality and boundary cases for virtually every unit in the iTrust system. When students are assigned faults to fix, they are required to write an automated unit test to ensure that the fault remains fixed. As the iTrust code is being developed, stu-

dents are required to maintain 80% line coverage of all Java classes. Between semesters, the automated unit test plan is improved and maintained such that 80% coverage is maintained on all relevant classes if the students had not done so. Students are encouraged, but not required to use a test-driven approach to writing unit tests. iTrust uses JUnit for our automated unit tests, and EclEmma for code coverage in an Eclipse environment.

iTrust also contains a number of supporting classes to aid the automated testing process. A test database is set up clean before each unit test on database functionality (i.e. DAO classes), and the test data is a standard data set across all student projects.

There are some packages and classes of the iTrust Java classes for which unit testing does not make sense or is not applicable. The following types of classes are excluded from the 80% coverage requirement:

- **The Server Package**, which contains Java classes that interface with the Apache Tomcat API to provide session time out functionality and other web-server specific features.
- **Test Utilities**, which provide developer-friendly methods for inserting the correct test data into the database.
- **Tag Classes**, which provide custom JSP tags for data fields such as the US state the patient lives in.

Automated HTTP Tests. The automated HTTP tests simulate a user using iTrust in an web browser. Using HTTPUnit[2], the automated HTTP tests execute on a fully-deployed iTrust system by crafting HTTP requests and checking the responses.

As opposed to the automated unit tests, the automated HTTP tests are intended for regression testing. Students are required to implement HTTP tests based on the acceptance test plan. Thus, each acceptance test case is represent by at least one HTTP test. Students also automate security penetration testing using HTTP tests.

5 Traceability in iTrust

The iTrust project administrators maintain multiple traceability matrices amongst the artifacts. The main three artifacts that are involved in tracing are:

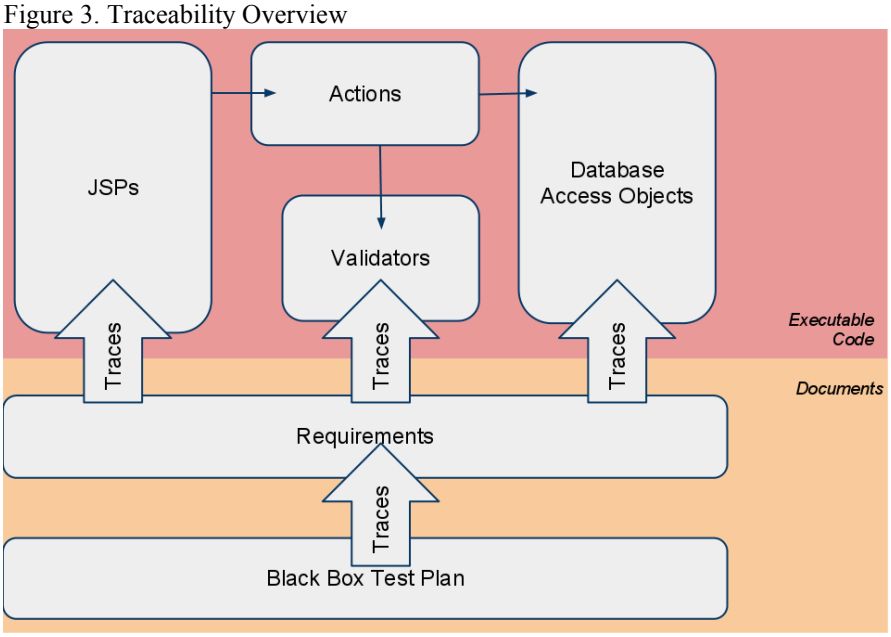
- Black box test plan

[2] http://httpunit.sourceforge.net/

- Requirements document
- System archetypes (e.g. JSPs, Actions, Validators, DAOs)

Figure 3 shows an overview of how the test plan, requirements, and system archetypes are traced to each other. The requirements document contains sub-sections for each use case (UC) that trace to the implementing JSP. Students can use this traceability analysis to find the place in the code that implements a given requirement for comprehending the code as well as improved testing. Additionally, the whole traceability matrix is available for students on the wiki for posterity.

Figure 3. Traceability Overview



To construct the tracing, a software engineering graduate student conducted a manual traceability analysis on iTrust. The procedure was as follows:

1. Examine the first (or next) use case sub-flow in the iTrust requirements document. Record the unique identifier of the use case. For example, UC1S3.
2. Manually perform the action described in the use case. Record the relative URL in the browser window along with the use case and sub-flow. For example /hcp-uap/addPatient.jsp. The observed URLs correspond to JSP files (e.g. addPatient.jsp) contained within the iTrust code base.
3. If the use case cannot be performed, or does not involve any JSPs, enter “No links” for the use case and sub-flow.

4. Inspect the JSP code for the recorded URL. If more than one JSP is involved in executing the described action, for instance when more than one URL is observed in the browser window while executing the action, record each JSP separately on its own line with a trace to the use case and sub-flow in question.
5. Record DAOs, Action classes and Validators separately with their own trace to the sub-flow and use case in question.
6. Return to Step 1.

For an extended example of this traceability analysis, consider iTrust Use Case 1 sub-flow 1.

UC1. Create and disable patients use case

Preconditions:
The iTrust HCP has authenticated himself or herself in the iTrust Medical Records system.

Main Flow:
An HCP creates patients and disables patients. The create/disable patients and HCP transaction is logged.

Sub-flows:
- [S1] The HCP enters a patient as a new user of iTrust Medical Records system. Only the name and email are is provided. An email with The patient's assigned MID and a secret key (the initial password) is personally provided to the user, with which the user can reset his/her password. The HCP can edit the patient with all initial values (except patient MID) defaulting to null and/or 0 as appropriate. Patient MID should be the number assigned when the patient is added to the system and cannot be edited. The HCP does not have the ability to enter/edit/view the patient's security question/password.
- [S2] The HCP provides the MID of a patient for whom he/she wants to disable. The HCP provides a deceased date. An optional diagnosis code is entered as the cause of death.

Table 2. Traceability Results for Use Case 1 Sub-flow 1

Use Case Subflow	Source Code
UC1S1	/auth/hcp-uap/addPatient.jsp
UC1S1	AddPatientAction().addPatient()
UC1S1	PatientDAO.addEmptyPatient()
UC1S1	AuthDAO.addUser()
UC1S1	PatientDAO.editPatient()
UC1S1	TransactionDAO.logTransaction()
UC1S2	No link

iTrust has a separately maintained list of manual black box test cases that students and administrators maintain. The black box test plan contained traceability to the requirements specification before the traceability analysis described in this chapter was complete. Students created and developed black box tests for the project as a part of their course requirements and included the use case and sub-flow their test case was based upon when creating the test.

This traceability analysis procedure was scoped for the purposes of this case study, and is limited in the following ways:

1. The traceability was conducted manually. We did not look at possible automated approaches since we conducted the analysis exclusively for this case study.
2. The matrix was not checked and confirmed by any other students. Another researcher or developer performing the analysis may arrive at different results.

From the 40 functional requirements in the iTrust requirements specification v18, we elicited 199 use case sub flows that could potentially trace to portions of the code. These 199 sub flows contained 609 separate links to 310 Java methods or JSP files. Of the 199 use case sub flows, 38 did not trace to any code within the iTrust project.

Although we traced the full list of 40 functional requirements, we excluded the set of six non-functional requirements in v18 of the iTrust requirements specification. The functional requirements typically traced to one or two components of each layer of the iTrust architecture. The traceability of the non-functional requirements in iTrust was less straightforward, however. Some of the non-functional requirements trace to every member of certain archetypes in iTrust (e.g. form validation), and others have no direct target (e.g. enabling multiple simultaneous users to be logged in).

6 Summary

iTrust is a patient-centered electronic health record web application used as an educational project in graduate and undergraduate software engineering courses at North Carolina State University. The software development project contains a use case-based requirements document, a black box test plan, automated tests, and source code. The project administrators maintain a manual traceability matrix from the black box test plan to the requirements document, and from the require-

ments document to the source code. iTrust is an open source software project, and all of its artifacts are publicly-available online.

7 References

[1] American Recovery and Reinvestment Act of 2009, U.S.C. 111-5, 2009.

[2] Health Insurance Portability and Accountability Act Privacy Rule. http://www.hhs.gov/ocr/privacy/hipaa/administrative/privacyrule/index.html

[3] US Department of Health and Human Services ER Use Case http://www.dhhs.gov/healthit/usecases/documents/EmergencyRespEHRUseCase.pdf

[4] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.