Proposing Regulatory-Driven Automated Test Suites

Patrick Morrison, Casper Holmgreen, Aaron Massey, Laurie Williams
Department of Computer Science
North Carolina State University
Raleigh, NC
{pjmorris, cmholmgr, akmassey, lawilli3}@ncsu.edu

***Abstract*—In regulated domains such as finance and health care, failure to comply with regulation can lead to financial, civil and criminal penalties. While systems vary from organization to organization, the same regulations apply for all systems. As a result, efficiencies could be gained if the commonalities between systems could be captured in public, shared, test suites for regulations. We propose the use of Behavior-Driven-Development (BDD) technology to create these test suites. With BDD, desired system behavior with respect to regulatory requirements can be captured as constrained natural language 'scenarios'. The scenarios can then be automated through system-specific test drivers. The goal of this research is to enable organizations to compare their systems to regulation in a repeatable and traceable way through the use of BDD. To evaluate our approach, we developed seven scenarios based on the HITECH Act Meaningful Use (MU) regulations for healthcare. We then created system-specific code for three open-source electronic health record systems. We found that it was possible to create scenarios and system-specific code supporting scenario execution on three systems, that iTrust can be shown to be non-compliant, and that emergency access procedures are not defined clearly enough by the regulation to determine compliance or non-compliance.**

Keywords- Behavior-Driven-Development; Healthcare IT; Regulatory Compliance; Security; Software Engineering; Software Testing

I. INTRODUCTION

In regulated domains such as finance and healthcare, organizations must ensure their software systems comply with applicable laws and regulations. Failure to comply often carries financial, civil and even criminal penalties. While systems vary widely among organizations, they must all check compliance against the same regulatory requirements.

For regulations, compliance is ultimately assessed by an external regulatory agency. A number of quality assurance techniques have been developed to assist organizational compliance. These approaches vary across industries, but typically include elements such as staff training, manual and automated monitoring, internal and external audits, and software certification. For software system development, compliance is a concern over the entire software lifecycle, from requirements [1] [2] to ongoing maintenance [3].

Behavior-Driven-Development (BDD) is a software development practice that organizes development effort around the creation of constrained natural language, termed 'scenarios', that describes user interactions with the proposed system and the system's responses in terms of the vocabulary used by system stakeholders [4]. These scenarios are then automated through the creation of system-specific test driver code that binds each scenario to the system. Each scenario, combined with the system-specific test driver code, serves as a test of the system's behavior. The collection of scenarios forms a test suite for the system. Proponents of BDD hold that by keeping the scenarios free of technical details, system users, subject matter experts and developers can share a common language for describing the expected behavior of a system. Frameworks that support this style of development include FIT [5], FitNesse[1], JBehave[2] and Cucumber [4].

The typical use case for BDD is in custom software system development. The scenarios and the system-specific test driver code are both associated with a single custom software system [6][7].

The goal of this research is to enable organizations to compare their systems to regulation in a repeatable and traceable way through the use of Behavior Driven Development. Tests suites built from scenarios can help to confirm that important issues have been addressed. Such test suites could become a shared asset for use by all systems subject to these regulations and standards. Each system, then, need only create their own system-specific test driver code to automate their compliance checks. System owners and auditors can gain confidence in the compliance of a system by running the compliance test suite(s) on their systems, obtaining indications of how their systems will respond to external audits through use of the test suite. At an industry level, a test suite for a regulation provides a target for implementers and a basis for comparison among systems.

To illustrate our approach, we created a compliance test suite consisting of seven scenarios based on the United States Health Information Technology for Economic and Clinical Health Act (HITECH) meaningful use (MU) regulations for security. To evaluate the test suite's reusability and generalizability, we implemented system-specific test driver code for three open source EHR systems; iTrust[3], OpenEMR[4] and Tolven[5].

The contributions of this paper are:

[1] http://fitnesse.org/
[2] http://jbehave.org/
[3] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php
[4] http://www.oemr.org/
[5] http://home.tolven.org/

- A proposal for using BDD technology to implement reusable test suites for regulatory-related system behavior.
- An example implementation of this proposal in the form of a BDD scenario test suite for security-related HITECH regulations
- Reporting on the use of this test suite for three EHRs

The remainder of the paper is organized as follows; Section 2 provides background for HITECH, EHRs, BDD, and test suites and reviews related work in testing EHRs for compliance and in the application of scenarios to checking requirements satisfaction. Section 3 presents related work. Section 4 describes our research methodology. Section 5 presents our application of the methodology. Section 6 presents our evaluation. Section 7 presents discussion and conclusions, section 8 is a discussion of limitations, and section 9 reviews and summarizes the paper.

II. BACKGROUND

In the United States, healthcare organizations must comply with the HITECH and Health Insurance Portability and Accountability Act (HIPAA) Acts, among others. HITECH regulations stipulate that failure to protect personal health information can lead to fines of up to $50,000 per violation and imprisonment for up to one year. The seven security scenarios we chose from HITECH parallel the published HIPAA Security Rule technical safeguards. Compliance with US regulations for medical record privacy is measured by observations of systems and organizational behavior by a US government agency, the Office of Civil Rights (OCR) [8].

BDD[6] is a software development practice that organizes development effort around the creation of scenarios of desired system behavior in collaboration with stakeholders [4]. These scenarios are then used to guide and then verify the results of the development process. Depending on the project, these scenarios may be associated with one or more requirements, or, for some teams, the scenarios themselves may serve as the requirements document. BDD begins with developers meeting with customers and other stakeholders to create a structured natural language requirements document, including scenarios of each requirement being developed during the next iteration. Over time, the collected scenarios accrete to serve as a regression test suite as well as a specification of system behavior to be implemented. These documents also serve as a critical part of the acceptance test infrastructure for the project. A number of teams use the produced documents directly as both requirements specification and acceptance tests [9]. Confirmation that the behavior described by a scenario is achieved by a system is accomplished through the development and execution of system-specific test driver code that links the scenario text to the concrete system functions. The key to BDD is that the scenario documents serve as a critical part of the acceptance test infrastructure for the project. When the scenarios are matched with the system-specific test driver code they can be automatically executed to verify system behavior.

Several BDD frameworks have been developed. We discuss two of the most common, the Framework for Integrated Tests (FIT) and Cucumber.

FIT [5] was, developed to enhance collaboration in software development and to help stakeholders learn what their software should do and what it does do[7]. FIT uses a tabular notation, stored as HTML, to describe a scenario. FIT fixtures are programming language code that connects the tabular descriptions to the system under test; each fixture must be built in conjunction with the tabular design and with the system code being tested.

Cucumber [4] is a framework for building suites of automated acceptance tests, based on the ideas of BDD. The framework is accessed through the use of 'feature files' and 'step files'. *Feature file* is Cucumber's term for a plain text file containing structured natural language descriptions of scenarios. Software developers and client stakeholders read feature files, and both groups are welcome to write them, though typically only developers write the files. *Step files* contain code that translates feature file vocabulary in to actions run against the system under test. Typically, only software developers read and write the code in the step files.

Although the details differ, FIT and Cucumber are based on similar concepts. We now describe a small example using both FIT and Cucumber, to highlight how these tools work. For both examples, we set a goal of testing authentication for an open source EHR, iTrust. The non-functional requirement is phrased 'The system shall enable multiple simultaneous users, each with his/her own exclusive authentication.'[8] We embellish this definition by providing a scenario describing steps taken during authentication[9], illustrating the definition's application in the system's context. The first step in the scenario is 'A user enters their MID [user identification] and their password to gain role-based entry into the iTrust Medical Records system'.

Both FIT and Cucumber support persistent text-based semi-formal instance and type scenarios [10] that describe system-user interactions and that record and report on the system's behavior in response to the scripted actions. Cucumber scenarios, Tab. 1, are plain text, and their viewing and authoring can be accomplished through any text editor. Cucumber's use of natural language text aligns well with the common use of natural language text for scenario description [9]. FIT requires that text be embedded in HTML tables, Tab 2., which has the benefit of built-in linking to relevant information, while incurring the cost of an enforced structure and HTML authoring for the production and editing of scenarios.

[6] http://behaviour-driven.org/

[7] http://fit.c2.com

[8] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=requirements

[9] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=requirements:uc3

TABLE I. CUCUMBER FEATURE EXAMPLE

```
Feature: Authentication

  Scenario Outline: Verify Authentication

    When I enter <username> and <password>
    Then  I <should_not> be able to log in

    Examples:
      | username | password | should_not |
      |   casper |   pass12 |     should |
      |   casper |    wrong | should not |
```

TABLE II. FIT TABLE EXAMPLE

A user with correct credentials should be able to log in.		
Authentication.Fixtures.Login		
Username	Password	Result
casper	pass12	Success
casper	wrong	Failure

III. RELATED WORK

A. Legislation and Software

A growing body of research examines how to link regulations and software requirements [11][1][12]. Within that, there has been some focus on how to measure the performance of running systems [1][13] against a requirements baseline. These approaches depend on the development of sophisticated monitoring layers by software experts. Our approach treats the test suite as the monitoring system, based on commonly available BDD technology, and each scenario is written in terms of the applicable regulation rather than a requirements specification.

In the United States, the U.S. Congress passes bills, which must then be signed into law by the President. For complex domains, laws often contain instructions for an Executive branch agency to create regulations that meet the standards outlined by the law. Often, even the regulations are too far removed from the problem domain for organizations to comply without additional guidance from legal counsel. Given the significant consequences of not addressing regulatory compliance issues, attention has been paid by the requirements engineering community to eliciting requirements from legislation [12], [14]. The legislative process produces laws and regulations that may serve as sources for requirements elicitation [14].

B. EHR Software Certicfication

Three sources of guidance in the EHR domain are the Certification Commission for Health Information Technology (CCHIT) [10], the Office of the National Coordinator for Health Information Technology (ONCHIT)[11], and the National Institute of Standards and Technology (NIST)[12]. CCHIT is an organization providing certification of EHR systems according to a set of internally developed criteria and test scripts. CCHIT makes these criteria and test scripts available on the web[13]. The certification process applies these scripts in a controlled environment. Each step is read from the script and keyed manually in to the candidate system. Each step has an 'Expected Result' column indicating the expected result of the step. A blank 'Actual Result' column allows manual entry of the step results. Comparison of the actual and expected results leads to a pass/fail decision, which is also recorded in the script. This process requires significant manual effort to execute each time, and further effort to review the results. The scripts exercise a wide range of functionality, however they do not necessarily cover all aspects of EHR security [15]. The HITECH act established the ONCHIT, which is charged with the development of nationwide Health IT infrastructure, including standards definition and the establishment of certification criteria and certification of bureaus that certify EHRs. CCHIT is the first of such bureaus, but a number of others have begun operations. NIST develops and publishes standards across a wide range of industries and topic areas, including a suite of test procedures targeting the regulations and guidelines established by the ONCHIT and HITECH[14]. The NIST-developed test procedures form the basis of our BDD scenarios, as there are explicit, documented links made between the NIST procedures and the regulations they are designed to check. This does not, in principle, limit the creation of scenarios to the presence of preexisting test procedures; however, the development of test procedures for a given regulation is a significant research challenge that also requires legal guidance [14]. This is beyond the scope of the present work. Narrowing the focus of the scripts to the content of the regulation allows clear traceability between the intent of the regulation and the actions taken to confirm the implementation of this intent.

C. Test Suites

Test suites are collections of test cases organized around some unifying purpose. *Validation test suites* check a piece of software's relationship to a set of requirements. Conformance test suites check a piece of software's relationship to a set of requirements embodied in some standard. To date we have found no formal definition of either phrase, and they appear to be used interchangeably in practice.

In the telecomm domain, a set of test suites for various network interoperability standards was built based upon TTCN-3, a telecomm industry standard for test specification.[15] In the domain of programming languages, validation suites consisting of executable acceptance tests establish conformance for a given language implementation

[10] http://www.cchit.org/

[11] http://healthit.hhs.gov/

[12] http://www.nist.gov

[13] https://www.cchit.org/cchit-certified

[14] http://healthcare.nist.gov/use_testing/index.html

[15] http://www.ttcn-3.org

to its specification. For example, Plum Hall[16] builds compiler validation test suites for C and C++. RubySpec is an open-source executable specification for the Ruby programming language. Java's Technology Compatibility Kit 3 serves a similar function for the Java language. Although licensing agreements vary, proper execution of a validation suite provides vendors, customers, and users confidence in the software's compliance with the official specification.

Morgan Stanley built a BDD test framework for validating financial time series data [15], although the test suite was applied to a single application rather than the multiple applications we propose..

D. BDD concepts

Grigori Melnik empirically evaluated 'Executable Acceptance Test-Driven Development'[7], finding that the practice enhances communications within software development teams, that executable acceptance tests can specify functional requirements in a consistent, verifiable and usable way, and that executable acceptance tests provide sufficient evidence of requirements traceability in regulated environments. He also found that tooling (FIT) for developing these tests suites negatively impacted their maintainability and scalability.

An industrial experience report [17] on Automated Test Driven Development (ATDD) describes a scheme of developing 'acceptance test case specifications' ('ATC-Specs') that are natural language descriptions of system behavior. In a case study they found 'the biggest advantage of using ATDD was that the customers understood the eventual behavior of the system better via the ATC-Specs than via the more formal SRS (Software Requirements Specification)'. They further commented 'Customers often do not 'think in' system use cases but 'think in' user interfaces where they enter data, press 'commit' and get the results displayed' [17].

E. BDD Frameworks

A position paper [9] described development using FIT, and proposed an equivalence hypothesis about the relationship between acceptance tests and requirements: '*As formality increases, tests and requirements become indistinguishable. At the limit, tests and requirements are identical'*. Based on this hypothesis, they argue that a suitable set of FIT tests can act as a requirements specification, a practice they maintain, and one that they claim other teams maintain. In practice, there is evidence that in some environments FIT-based scenarios are adequate to document requirements [9].

Over time, a number of academic studies have evaluated various aspects of FITs attributes and use. A 2009 review of these studies [17] found that, contrary to intent, FIT tests were typically authored and used only by developers rather than as a communications tool between stakeholders. The stakeholders preferred plain text to the browser-based HTML tables used by FIT. Developers did find the tests helpful in guiding development and in reducing time to discover and resolve errors, especially when the tests were used to verify proper behavior after changes [17].

One academic study conducted a series of experiments focused on measuring the utility of FIT in assisting requirements understanding [13]. A set of requirements was presented to students. The control group received no FIT tests, while the treatment group received FIT tests along with the requirements. In the words of the study, "When Fit tables are present, the chances of correctly understanding a requirement are in most cases (95%) at least two times higher than without them, and on average 4 times higher" [18].

The structured natural language statements contained in Cucumber feature files are as follows [4]: **Given** some initial context, **When** an event occurs **Then** verify some outcome. **Given** specifies a set of preconditions, allowing both documentation and confirmation of necessary conditions for a successful test. **When** describes the actor, objects, and an action. **Then** describes the expected results of the action taken. Two other terms are used; Scenario collects a related set of Given/When/Then statements, and Feature collects multiple related scenarios that describe a single system feature. When Cucumber runs, it parses the feature files according to this grammar, and connects statements from the feature files to the system-specific test driver code in the step file(s) that implement each statement.

To date, Cucumber has not been the subject of academic studies. However, a number of aspects of its design commend it to our purposes; The separation of the structured natural language feature files from the system-specific test driver code contained in the step files allows each system to have a set of step files that implement the shared tests described in the feature files. The plain text nature of the feature files allows them to be read and written by any text editor, minimizing needed tool support.

IV. BDD FOR REGUATORY TEST SUITES

The goal of this research is to enable organizations to compare their systems to regulation in a repeatable and traceable way through the use of BDD. To achieve this, we must, at a minimum *Identify Regulations*, *Develop Scenarios* and *Automate Scenarios*. The tasks listed above form the outline of our methodology. We now discuss them in greater detail.

Step One: Identify Regulations

Select all or part of a regulatory text. Within the selected text, identify each regulation with which a system must comply. Regulations can be identified in the regulatory text by phrases of the form 'An <actor> [must|must not] or shall perform some action', and by the heading 'Implementation specifications'. This parallels requirements extraction. In general, identifying requirements in regulatory texts is a difficult problem that requires not only engineering expertise but legal advice [13].

Step Two: Develop Scenarios

[16] http://www.plumhall.com/

Regulations are typically phrased in declarative language, identifying required behavior, constraints, and limits, but lacking description of how to identify whether the expected behavior has been accomplished. A scenario, a step-by-step test procedure, must be associated with each tested regulation to validate its achievement. The scenario takes the form of a detailed set of instructions readable by a person. CCHIT and NIST are two sources of test procedures that test various aspects of health care systems, but custom scenario development may be also done.

For traceability, each scenario should be clearly named, and each scenario should contain a reference to the specific section of regulation that is being exercised by the structured natural language scenario.

Step Three: Automate Scenarios

Once a scenario for checking regulatory conformance is available, code for adapting it to a system must be written, including appropriate roles, sequences of steps and verification conditions. The code is split in our approach in to a structured natural language component that describes a scenario independent of a given system, and system-specific driver code that is used to execute on a given system.

For traceability, code should be clearly named, and should contain references to related scenarios and regulations. The scenario name and regulation reference should be displayed when the scripts are executed.

Completing these steps for all or part of a regulatory text establishes a baseline for the development of the acceptance test suite, and provides a suite of tests that can be compared against other means of testing.

V. EVALUATION

We now describe the use of this three-step process on three electronic health record systems.

Step One: Identify Regulations

Our research group has focused on EHR system security [15][19][20]. One of the most studied regulations in this area concerns HIPAA technical safeguards [12][11][20]. No test procedures for these have been published, but there are an analogous set developed by the NIST for testing the HITECH act meaningful use provisions [21]. MU covers a wide range of EHR functionality requirements, and the NIST has developed test procedures for each of them. The language of HITECH sections 170.302(o)-(u) [22] closely matches the language of HIPAA 164.302(a)-(g), to the point of verbatim language in some sections. Rather than attempt to claim that these regulations are directly comparable here, we choose to mention the correspondence, and to base our test suite on the test procedures associated directly with the HITECH regulations in CFR 170.302. The development of custom test procedures, and the linkage of related pieces of regulation are both open and active research areas.

Step Two: Develop Scenarios

Our choice of HITECH 170.302(o)-(q)[22] regulations leads directly to the selection of the NIST test procedures 170.302(o)-(q)[21] for translation to executable scenarios. We now illustrate process with an excerpt from one of our seven scenarios, Authentication.

The text of HITECH meaningful use regulation 170.302(t) is "*Authentication.* Verify that a person or entity seeking access to electronic health information is the one claimed and is authorized to access such information.*"*

The text of the associated NIST 'Required Test Procedure', 170.302(t), is as follows:

1. TE170.302.t – 1.01: Using the Vendor-identified EHR function(s), the Tester shall create a new user account and assign permissions to this new account
2. TE170.302.t – 1.02: Using the new user account, the Tester shall login to the EHR using the new account
3. TE170.302.t – 1.03: The Tester shall perform an action authorized by the assigned permissions.
4. TE170.302.t – 1.04: The Tester shall verify that the authorized action was performed
5. TE170.302.t – 1.05: The Tester shall perform an action not authorized by the assigned permissions
6. TE170.302.t – 1.06: The Tester shall verify that the unauthorized action was not performed
7. TE170.302.t – 1.07: The Tester shall log out of the EHR
8. TE170.302.t – 1.08: The Tester shall delete (e.g., deactivate or disable) the new account
9. TE170.302.t – 1.09: The Tester shall attempt to login to the EHR using the deleted account
10. TE170.302.t – 1.10: The Tester shall verify that the login attempt failed
11. TE170.302.t – 1.11: Using the NIST-supplied Inspection Test Guide, the Tester shall verify that:
    a. an account has been created
    b. can sign-in to the account
    c. can authorize the assigned permissions can delete (e.g., deactivate or disable) the account the log-in attempt has failed

The procedure provides a relatively concrete set of steps that have been approved to check the regulation.

Step Three: Automate Scenarios

In order to translate the relatively concrete steps laid out by the test procedure in to code that can be executed on each of our EHR systems, we chose the BDD tool Cucumber. We chose Cucumber over FIT primarily because Cucumber is less restrictive in the form of input it accepts, while FIT requires all input to be formatted as HTML tables. We think this is important because it allows flexibility in both the phrasing of the scenarios and in the tooling required to read and change the feature files that contain scenarios.

```
Feature: Authentication

NIST Â§170.302(t) Authentication

Background:
       * Using the Vendor-identified EHR function(s), the Tester shall create a new user
account and assign permissions to this new account

Scenario: Verify authorization

DTR170.302.t 1:  Verify authorization     evaluates the capability to verify that a person
or entity seeking access to electronic health information is the one claimed and is
authorized

* Using the new user account, the Tester shall login to the EHR using the new account
* The Tester shall perform an action authorized by the assigned permissions.
* The Tester shall verify that the authorized action was performed
```

Figure 1: Cucumber feature file excerpt, Authentication

We now illustrate our automation implementation by showing steps 1.01-1.03 of the NIST 170.302(t) procedure, including excerpts from the Cucumber feature file (Figure 1), and supporting step files for iTrust (Figure 2) and Tolven (Figure 3). Complete listings of all files are available from the project's BitBucket site[17].

a) Automating NIST 170.302(t), step 1.01

The first step of the test procedure, 1.01, calls for the creation of a new user account and the assignment of permissions to the account. Since the new account is needed more than once over the course of the test procedure, we perform the account creation at the beginning of the feature file. We do so through the use of Cucumber's 'Background' element, which performs common setup required by each 'Scenario' in a feature file.

Our Gherkin text for this, 'Using the Vendor-identified EHR function(s), the Tester shall create a new user account and assign permissions to this new account', is associated with the first 'do'..'end' blocks in the system-specific test driver code shown in Figures 2 and 3. These functions log in a user that can create other user accounts, calls a user creation account routine with an example user, and logs out the original user for iTrust and Tolven, respectively.

Our translation of the language of the test procedure requires comment on four points. First, note that the NIST test procedure refers to a Tester, and that the feature file does not; we have written the feature file from the perspective that the feature file itself can be treated as 'The Tester'. Secondly, where the test procedure calls for 'Vendor-identified EHR function(s)', we relied upon our own investigation of each system's documentation. While we believe this to be suitable in our circumstances, more thorough interaction with a vendor may be appropriate to assure that the vendor's intent is reflected by the choices made for the step files. Thirdly, note that the implementations for each step vary between iTrust and Tolven; while the feature file is constant, there are system-specific details that the step files abstract away. Fourthly, we used different roles for each system because roles are treated differently from system to system. For example, in iTrust patients have appropriate permissions differences suitable for testing, but in Tolven they do not. We created a patient for iTrust, and a Health Care Professional (HCP) for Tolven. We found an alternate role in Tolven that exhibited the behavior to be tested by the scenario.

b) Automating NIST 170.302(t), step 1.02

The second step of the test procedure, 1.02, calls for the new user to log in to the system. The iTrust and Tolven step files both delegate logging in to a system-specific test driver routine, 'login', not shown. The iTrust step also includes a password reset, something required of new iTrust users on their first login.

c) Automating NIST 170.302(t), step 1.03

The third step of the test procedure, 1.03, calls for the new user to perform an authorized action. The iTrust and Tolven step files each implement this by navigating to a link available to the logged in user in the role the user is in.

Each step of each scenario called for an assessment of the test procedure and of how to execute the test procedure on the system, followed by experimentation to discover the users, roles, objects and actions provided by each system, and the technical means by which to automate the required actions.

VI. EVALUATION

We evaluate our test suite by measuring and reporting on our implementation and execution of the methodology tasks against seven regulations on three EHR systems and reporting the degree to which we were able to automate the test procedures. We also place our initial test suite in context against the complete list of NIST test procedures and their relationship to the HIPAA regulations.

[17] http://bitbucket.org/icasperzen/hipaa_cuke

```
Given /^Using the Vendor\-identified EHR
function\(s\), the Tester shall create a
new user account and assign permissions to
this new account$/ do
    @user = default_hcp
    login(driver,@user)
    @new_user =
         create_new_patient(driver,ITrust
         ::User.new(first_name:'Ted',
         last_name:'Nugent',
         email:'ted@nugent.com'))
    driver.find_element(link:'Logout')
         .click
end

Given /^Using the new user account, the
Tester shall login to the EHR using the
new account$/ do
    reset_password(
        driver,@new_user,
        'password')
    @user = @new_user
    login(driver,@user)
end

Given /^The Tester shall perform an action
authorized by the assigned permissions\.$/
do
    driver.find_element(
        link:'My Demographics')
        .click
end

Given /^The Tester shall verify that the
authorized action was performed$/ do
    driver.title.should
        == 'iTrust - Edit Patient'
end
```

Figure 2: iTrust step file excerpt, Authentication

```
Given /^Using the Vendor\-identified EHR
function\(s\), the Tester shall create a
new user account and assign permissions to
this new account$/ do
    login("admin","sysadmin")
    create_new_staff
    add_testaccount_to_chr
   logout
end

Given /^Using the new user account, the
Tester shall login to the EHR using the
new account$/ do
    login("testaccount","twk27kox")
end

Given /^The Tester shall perform an action
authorized by the assigned permissions\.$/
do
    driver.get(base_url + "/Tolven")
    driver.find_element(:link,
        "Appointments").click
end

Given /^The Tester shall verify that the
authorized action was performed$/ do
    driver.title.should
      match /Appointments/
end
```

Figure 3: Tolven step file excerpt, Authentication

The goal of the test suite is to establish the link between regulations and the behavior of the tested system. Practically, this means ensuring each aspect of the regulation is represented by one or more scenarios, that a given system implements the function required by the scenario, that there is step code to implement every aspect of each scenario, and that the system under test responds correctly when tested. We use a coding scheme 'SFIP', described below, to indicate these attributes for a single scenario on a single system. The presence of the letter indicates the presence of the attribute, and the absence of the letter indicates the absence of the attribute.

When evaluating the link between the regulation(s) and scenarios, we need to consider whether a scenario correctly represents a segment of the regulation, whether the collected scenarios represent the selected regulation(s), and what portion the selected regulation represents of the complete regulation. The latter measures are aggregates, while the first measure considers the link between one segment of regulation and one scenario.

- 'S' indicates that we were able to define a scenario that correctly represents its segment of the regulation. Absence indicates that we could not define a behavioral example for the regulation. It is beyond the scope of this work to fully address the relationship between regulation and scenarios. We make the simplifying assumption that the NIST test procedures correctly reflect their like-numbered HITECH regulations. Developing a scheme for mapping regulatory requirements and scenarios is an open problem, an area of active research and a candidate for future work.
- 'F' indicates whether the system provides the function required by the scenario. Absence indicates that the system does not support the function called for by the scenario.
- 'I' indicates whether the step code implements the steps called for by the scenario. Absence indicates that step code was not defined or completed, either because a scenario was not defined or because the system did not provide the necessary feature.
- 'P' indicates whether the system passes the scenario as executed. Absence may be because the test failed, or because one of the preceding definitions – scenario, function, step code, has not been made.

TABLE III. EVALUATION SUMMARY

Regulation	Procedure	Automation	Electronic Health Record System		
			iTrust	*OpenEMR*	*Tolven 2.1*
Access Control - CFR 170.302(o)	NIST 170.302.o	login.feature	SFIP	SFIP	SFIP
Emergency Access – CFR 170.302(p)	NIST 170.302.p	emergency_access.feature	S	SFP	SFP
Automatic Logoff – CFR 170.302(q)	NIST 170.302.q	automatic_logoff.feature	SFIP	SFIP	SFIP
Record actions – CFR 170.302(r)	NIST 170.302.r	audit_log.feature	SFIP	SFIP	SFIP
Integrity – CFR 170.302(s)	NIST 170.302.s	integrity.feature	SFIP	SFIP	SFIP
Authorization – CFR 170.302(t)	NIST 170.302.t	authentication.feature	SFIP	SFIP	SFIP
Encryption – CFR 170.302(u)	NIST 170.302.u	general_encryption.feature, transfer_encryption.feature	S	SF	SF
Totals: 7	7	7			

We have tabulated our results for each task and each system in Table III. Each row represents one of the scenarios implemented for the test suite, including the section of the regulation that is addressed, the test procedure used, the name of the feature file in which the scenario is implemented, and a score for each EHR system. Systems receive one point each for presence of the functionality tested in the scenario, step code for executing the scenario, connection between the feature file and the step files, and successful execution of the step code testing the functionality.

We built scenarios for encryption, and system-specific test driver code, but did not configure encryption in our environments. Tolven and OpenEMR support encryption of data documents for transmission, but it can be switched on or off by the using organization, so the EHR can be compliant, but the organization may not be. In both these cases, the unencrypted data stored in the relational database must be protected by proper database server administration. iTrust does not support encryption of data documents for transmission.

We built scenarios and system-specific test driver code for Emergency Access, but did not obtain documentation on specific Emergency Access features in the tested systems and so we cannot comment on how emergency access functionality bears on systems being potentially compliant. iTrust and Tolven did not provide specific Emergency Access functionality. OpenEMR's 'SFP' reflects present functionality, step definition and execution, and a need to finish connecting the step files to the feature files.

We now address the attributes of repeatability and traceability. Cucumber displays the text of each scenario as it executes each step. Successfully executed steps are printed in green, unsuccessful steps are printed in red. This makes it possible to quickly check for the status of a given test. The output of Cucumber test runs can be saved to text files, allowing for machine comparison of test runs against previous successful executions. We believe this supports repeatability. The annotation of each scenario with a section reference for its related test procedure and regulation, together with the display of the section references on every test run, offers the ability to trace from a failing test back to the regulation to which it is related. The annotations also allow the test suite text files to be searched by regulation and test procedure section references in order to assess what the test suite covers, another aspect of traceability.

Because it does not support encryption, iTrust can be recognized as non-compliant. Because emergency access procedures are not defined concretely enough to build scenarios, OpenEMR and Tolven cannot be definitively recognized as compliant. One possible approach for such cases would be to define system-specific scenarios representing how the system implements the feature called for by the regulation.

Where all elements were present, the scenarios and step files created provide a repeatable, traceable link between the regulation text and the behavior of the individual system. Where some elements are missing, it is possible to identify what is missing and why.

VII. DISCUSSION AND CONCLUSIONS

We were able to define scenarios for clearly defined regulatory texts. We were not able to define scenarios for 'emergency access procedures' where regulatory concepts were introduced but not defined. When scenarios could be defined and systems supported the relevant functions called for in the scenarios, we were able to build system-specific step files to execute the scenarios. BDD's concept appears to be a useful one. BDD can be used to describe system behavior in scenarios that both users and developers can use. Those scenarios can then be tied to test system behavior.

Two direct benefits of storing features as text files are that no tooling beyond a text editor is required and that links to original documents can be placed directly within the tests themselves, supporting traceability. The absence of tooling for reading and writing feature files supports non-technical users.

The step definitions are written in a full programming language, and can use any library written for that programming language. The present project used Cucumber's original language, Ruby[18], but many other languages and environments are supported. For example, the Cucumber development team has recently released a pure Java version for JVM languages[19], allowing integration and use of that population of libraries.

The particular EHRs we evaluated are browser-based. Their step definitions required the use of web test framework technology; evaluating such technology is a project in its own right. While the selected tools are frequently used, we did not undertake a comparative assessment of tool chains, and a different tool chain might be more applicable to the needs of the EHR system step definitions.

VIII. LIMITATIONS

We have automated a small number of test procedures for a small number of EHRs. Applying these techniques to further test procedures and to more systems may reveal issues not covered by our efforts to date.

We were able to maintain system-independent scenario files by pushing system-specific differences down in to the step files for the situations where they arose in our examples. This may not generalize to every scenario and every system.

Our parsing of scenario files in to step definitions is based on regular expression parsing. This is a relatively low sophistication practice that does not yet approach the identification of a domain language for describing and reasoning about regulatory scenarios. It appears that there is room here for developing a more sophisticated parsing approach that more accurately identifies parts of speech in the language used to describe the feature.

The specific users and patients maintained by an EHR will vary from installation to installation, even for a given EHR. Developing test suites that exercise system behavior depend on using available users and patients, something that will vary from installation to installation. We selected one approach, but this requires per-installation customization in a way that may introduce unwanted variability in the test results. One approach to solving this would be to supply patient and user creation routines in the scenario language, and to create the users and patients necessary for the test suite as part of the test suite definition. This does not solve the problem of performing these tests against a production system, as the audit logs, for example, will contain test data as well as production data.

Automated acceptance testing of the form described here is a form of object-oriented software development, and so it is subject to all the challenges and limitations of a development project; language familiarity, tool choice and maturity, design choices and developer skill all have a significant impact on the outcome.

Automated testing of web-based systems involves a complicated tool chain, including the system under test, one or more browsers, Javascript, test frameworks, scripting languages, and libraries. The learning curves involved for both the tool chain and systems-under-test, the difficulty of developing a generic language for features, and the difficulty of maintaining proper state in unattended operation, all contribute to difficulties in developing and maintaining the example test suite.

More fundamentally, creating a grammar that meets the needs of non-technical readers and technical writers at the same time may depend on organizational context to the degree that an industry-wide language is not possible. More work is required to assess the viability of this idea.

IX. SUMMARY AND FUTURE WORK

This project has evaluated the feasibility of using BDD acceptance test suites to support checking of regulatory requirements. We used Cucumber, a BDD tool, to implement seven scenarios based on HITECH security meaningful use compliance guidelines on three systems in order to compare system behavior with its governing regulations and to provide traceability between the regulations and system behavior. Two natural next steps would be to add the remainder of the meaningful use regulations to the test suite, and to pursue implementations of the test suite for other EHR systems. Publishing both the generic feature files and system-specific step files on the web will provide a direct measure of the perceived utility of the suite, by enabling measurement of views, downloads and check-ins by other parties. Given the baseline, finding other means to evaluate its speed, simplicity and accuracy compared with existing methods should be sought. A survey of testing procedures and experiences among certification bureaus, developers of EHRs and user organizations (e.g. hospitals, doctor's practices) should be conducted to form a basis for this comparison. A number of objects and actions appeared repeatedly in the system-specific driver code: users, patients, HCP's, navigation. It may prove valuable to extract a vocabulary and grammar based on the nouns, verbs and relationships in the working test suite. Such a vocabulary could prove useful for constructing additional scenarios and test procedures for aspects of the domain that are not currently addressed. Another natural extension would be to apply the idea of a BDD test suite to other regulations and standards, as a check on its generality.

ACKNOWLEDGMENT

The authors would like to thank the members of the Realsearch group, the members of The Privacy Place, and the students of CSC 591, section 01 Fall 2011 and CSC 591, section 007, Spring 2012 for their efforts, reviews and comments on early drafts of this work.

[18] http://www.ruby-lang.org/en/

[19] https://github.com/cucumber/cucumber-jvm

REFERENCES

[1] S. Ingolfo, A. Siena, and J. Mylopoulos, "Establishing Regulatory Compliance for Software Requirements," in *Conceptual Modeling – ER 2011*, vol. 6998, M. Jeusfeld, L. Delcambre, and T.-W. Ling, Eds. Springer Berlin / Heidelberg, 2011, pp. 47–61.

[2] T. D. Breaux and A. I. Anton, "Analyzing Regulatory Rules for Privacy and Security Requirements," *Software Engineering, IEEE Transactions on*, vol. 34, no. 1, pp. 5 –20, Feb. 2008.

[3] N. Chapin, "Software maintenance in complying with IT governance: A report from the field," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, 2009, pp. 499 –502.

[4] M. Wynne and A. Hellesoy, *The Cucumber Book: Behavior-Driven Development for Testers and Developers*. Pragmatic Press, 2012.

[5] R. Mugridge and W. Cunningham, *Fit for Developing Software: Framework for Integrated Tests (Robert C. Martin)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.

[6] B. Haugset and G. K. Hanssen, "Automated Acceptance Testing: A Literature Review and an Industrial Case Study," in *Proceedings of the Agile 2008*, Washington, DC, USA, 2008, pp. 27–38.

[7] G. I. Melnik, "Empirical analyses of executable acceptance test driven development," University of Calgary, Calgary, Alta., Canada, Canada, 2007.

[8] "http://www.hhs.gov/ocr/privacy/hipaa/enforcement/process/howocrenforces.html." .

[9] R. C. Martin and G. Melnik, "Tests and Requirements, Requirements and Tests: A Möbius Strip," *Software, IEEE*, vol. 25, no. 1, pp. 54–59, Feb. 2008.

[10] C. Rolland, C. B. Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois, and P. Heymans, "A proposal for a scenario classification framework," *Requir. Eng.*, vol. 3, no. 1, pp. 23–47, Sep. 1998.

[11] J. C. Maxwell and A. I. Antón, "The production rule framework: developing a canonical set of software requirements for compliance with law," in *Proceedings of the 1st ACM International Health Informatics Symposium*, New York, NY, USA, 2010, pp. 629–636.

[12] A. K. Massey, B. Smith, P. N. Otto, and A. I. Anton, "Assessing the accuracy of legal implementation readiness decisions," in *Requirements Engineering Conference (RE), 2011 19th IEEE International*, 2011, pp. 207 –216.

[13] W. N. Robinson, "Implementing Rule-Based Monitors within a Framework for Continuous Requirements Monitoring," in *Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences - Volume 07*, Washington, DC, USA, 2005, p. 188.1–.

[14] P. N. Otto and A. I. Antón, "Addressing Legal Requirements in Requirements Engineering," *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pp. 5–14, Oct. 2007.

[15] A. Austin, B. Smith, and L. Williams, "Towards improved security criteria for certification of electronic health record systems," in *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, New York, NY, USA, 2010, pp. 68–73.

[16] R. Salama, "A regression testing framework for financial time-series databases: an effective combination of fitnesse, scala, and kdb/q," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, New York, NY, USA, 2011, pp. 149–154.

[17] F. Salger and G. Engels, "Knowledge transfer in global software development: leveraging acceptance test case specifications," in *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, 2010, vol. 2, pp. 211 –214.

[18] F. Ricca, M. Torchiano, M. Di Penta, M. Ceccato, and P. Tonella, "Using acceptance tests as a support for clarifying requirements: A series of experiments," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 270–283, Feb. 2009.

[19] B. Smith, A. Austin, M. Brown, J. T. King, J. Lankford, A. Meneely, and L. Williams, "Challenges for protecting the privacy of health information: required certification can leave common vulnerabilities undetected," in *Proceedings of the second annual workshop on Security and privacy in medical and home-care systems*, New York, NY, USA, 2010, pp. 1–12.

[20] E. Helms and L. Williams, "Evaluating access control of open source electronic health record systems," in *Proceedings of the 3rd Workshop on Software Engineering in Health Care*, New York, NY, USA, 2011, pp. 63–70.

[21] "Approved Test Procedures Version 1.0." NIST, 2010.

[22] *45 CFR Part 170, Health Information Technology; Initial Set of Standards, Implementation Specifications, and Certification Criteria for Electronic Health Record Technology; Interim Final Rule D. o. H. a. H. Services, 2010.* .