Journal of Software Engineering and Applications, 2014, 7, 187-194
Published Online March 2014 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2014.73020

Doorstop: Text-Based Requirements Management Using Version Control

Jace Browning, Robert Adams

School of Computing and Information Systems, Grand Valley State University, Allendale, USA
Email: jacebrowning@gmail.com, adams@cis.gvsu.edu

Received 24 January 2014; revised 20 February 2014; accepted 27 February 2014

Abstract

Effectively managing the requirements and traceability in a complex software project can be a challenging task. Many tools exist to support the initial creation and management of changes to text-based requirements. The most popular commercial solutions use a centralized server to host a database with a front-end desktop or web interface. Some downsides to this approach include user interface bloat, server costs, and an inherent disconnection from the project's source files. To provide an alternative to traditional requirements management, Doorstop was created as a tool to allow requirements to be stored as text files in version control. This solution allows a project to utilize its existing development tools to manage versions of the requirements using a lightweight, developer-friendly interface.

Keywords

Requirements Management; Version Control; Software Tools

1. Introduction

The primary goal of a "requirements management" tool is to capture requirements in a structured fashion and provide a means to create, edit, link, and baseline requirements collaboratively. While software requirements exist in many forms, the scope of the tool described by paper is the textual requirements used to define prescribed and proscribed behavior of a system and its subsystems. This is the style of requirements used by most aerospace, medical, automotive, and other industrial projects.

Within this scope, a "requirement" typically exists as one or more sentences where keywords such as "shall", "should", and "may" are given specific meanings to define the behavior of a system. A collection of requirements is often organized in outline form to create a document structure. Other documents might contain test cases or references in source code files. Items within a document are "linked" to items in another document to convey fulfillment of implementation or testing. These links are ultimately used in a verification phase to measure the accuracy of requirements decomposition and test coverage. A sample collection of documents is shown in **Figure 1**.

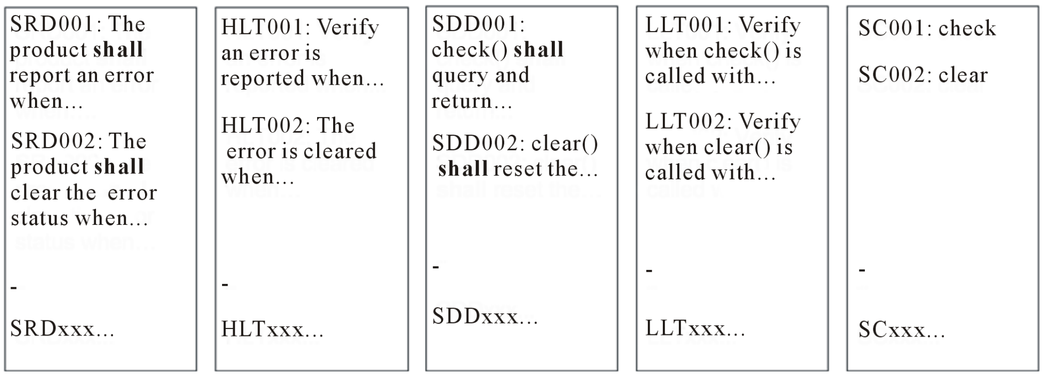
This figure shows a software requirements document (SRD), high-level test document (HLT), software design document (SDD), low-level test document (LLT), and source code document (SC). The SRD contains the main requirements for the product. The HLT items will link to the items in the SRD to trace testing, while the SDD items will link to the SRD to trace design. The SDD contains the design requirements. The LLT items will link to the SDD items to trace testing, while the SC items will link to the SDD items to fulfill implementation. A sample linking hierarchy can be seen in **Figure 2**.

Effectively tracking the linkages between requirements, design, implementation, and test is important to the success of a program and requires a good tool. Inadequate tool support can have a costly impact on an organization's development and maintenance efforts [1]. The motivation to create a new text-based requirements management tool stems from the perceived inadequacies of existing tools used to manage textual requirements. Traditional requirements management tools have redundant and disconnected functionality from the tools used to manage source code. Many exiting solutions are also cumbersome to use and expensive to purchase and maintain.

This paper presents a new approach to managing text-base requirements called Doorstop. The hallmarks of Doorstop are its utilization of existing version control systems and usage of tools developers are already familiar with (namely, the command line and text editors). Section 2 outlines the necessary features of any requirements management solution. Section 3 provides an overview of existing requirements management tools and their current limitations. Based on that review, Section 4 discusses the design of Doorstop to overcome these limitations, and Section 5 covers some implementation details. Section 6 provides an initial evaluation of the tool, and Section 7 concludes with areas where the tool can still be improved.

2. Required Features

Regardless of the approach taken, all tools designed to help manage requirements must have certain features,



SRD001: The product **shall** report an error when...	HLT001: Verify an error is reported when...	SDD001: check() **shall** query and return...	LLT001: Verify when check() is called with...	SC001: check
SRD002: The product **shall** clear the error status when...	HLT002: The error is cleared when...	SDD002: clear() **shall** reset the...	LLT002: Verify when clear() is called with...	SC002: clear
-	-	-	-	-
SRDxxx...	HLTxxx...	SDDxxx...	LLTxxx...	SCxxx...

Figure 1. Sample requirements, test cases, and functions.

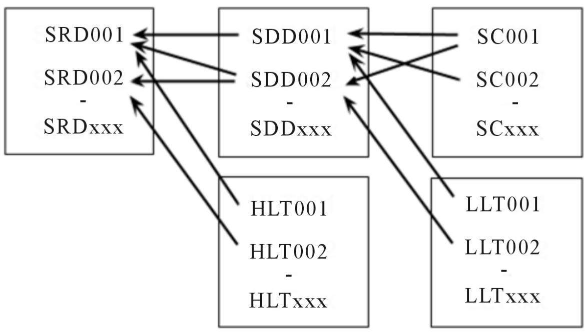


Figure 2. Sample linking hierarchy.

which can be discussed based on application.

2.1. Composition Features

The tool must provide uniquely identified versioned and linkable sections of text. Items in a requirements document must be assigned a permanent identifier to maintain the integrity of links between items when content changes. The text should support formatting with the option to reference external content outside of pure text. The requirements management solution must also be expandable. An open architecture allows interfaces to communicate with other tools and support integration throughout the entire development process [2].

2.2. Presentation Features

The tool must provide a way to represent a set of requirements in a useful fashion. Not all consumers of software requirements are programmers, so multiple presentation formats must be provided to support the needs of less technical users. This includes providing the ability to flexibly control filtering based on attributes as well as viewing the requirements in a more traditional document-oriented manner [3]. The requirements documents are also used to track changes and analyze impact. The tool must provide an auditable trail of baselines approved through the development process [4].

2.3. Administration Features

The tool must provide storage of requirements in a permanent and secure manner. It must also handle change management so that modifications to requirements can be formally requested and reviewed [3]. The ability to connect requirements change management to existing user accounts provides an added benefit. The tool must be scalable to support thousands of requirements, but also maintain a lightweight installation option across multiple sites and platforms [2].

3. Existing Solutions

Many tools exist to facilitate the management of requirements, and fall into a few distinct categories.

3.1. Centralized Tools

By far the most popular solutions for corporate environments are requirements management tools utilizing a centralized database with a graphical front-end. Commercial examples of this type include Rational DOORS [5], Dimensions RM [6], and MKS Integrity [7]. Open Source Requirements Management Tool (OSRMT) [8] offers a less feature-rich, but free option. The users of these tools interact with the requirements document in a tabular form where linkable items in a document are stored as entries in a centralized database. These tools tend to have extensive reporting capabilities to publish documents in a desired format.

These kinds of tools have some disadvantages. High user license and server costs make the paid options unsuitable for smaller companies. All tools in this category require network access to function making them susceptible to server downtime and remote access issues. Scripting interaction with the database typically involves using a proprietary DSL (domain-specific language) and hurts interoperability with other tools. Finally, these kinds of tools have limited connection to the source code making functional tracing more difficult.

3.2. Cloud-Based Tools

Several companies have created websites to serve as hubs for requirements collaboration. Commercial examples of this type include Accompa [9] and Jama [10]. An open-source option is provided by aNimble Platform [11], which is compatible with OSRMT. The users of these tools interact with the requirements document using a web browser, often using a cleaner and more modern interface. Cloud-based solutions also have the added benefit of letting someone else manage the server infrastructure.

These kinds of tools still have some disadvantages. Many companies have policies restricting any parts of their software development artifacts from being hosted by a third party. Self-hosting requires dedicated hardware and support. Like the centralized solutions, these types of tools also use proprietary formats and provide limited extensibility and integration with other tools.

3.3. Decentralized Tools

In recent years, several tools have been created that attempt to change the model of requirements management. These tools avoid requiring a dedicated server by storing requirements as files rather than database entries. File storage enables much tighter integration between the requirements and source code.

One example is rmtoo [12], which is a pure command-line interface with dependency graph generation. This tool uses the model of storing requirements as text files. Unfortunately, the tool is not specifically designed to support Windows and lacks a GUI interface, which may deter less-technical users.

4. Doorstop Model

Doorstop's model of requirements management is based on text files stored in a version control repository. Each text file represents one linkable section of text. These "item files" can contain the text of a requirement or test, a list of other item files they link to, and optionally an external reference to source code or another file. Doorstop assigns unique, sequentially numbered names to these item files for the purposes of linking and historical review.

Folders (directories) in the filesystem provide the hierarchical organization. Doorstop creates a "document directory" for the storage of related items files. The tool recognizes the parent-child relationship of these document directories and uses this information to map the relationship of documents. The collection of all documents in a directory forms a document hierarchy or tree. An overview of this model using parts of the documents from section 1 is shown in **Figure 3**.

Doorstop creates and validates these document directories and item files. The tool allows users to create new requirements and tests as item files and establish links between them. Doorstop also provides a mechanism to publish a collection of items in various document formats.

Doorstop expects that item files and document directories are stored in a version control system (VCS), which handles the responsibility of managing the history and change management of the requirements and tests. After item files are created, they can be committed to the version control repository for storage. Baselines of the requirements and tests can be created using the VCS's tagging mechanism. Branching and merging are used to isolate requirements changes still under review. Existing user credentials are associated with changes to lines of text by the VCS. Utilizing a distributed version control system (DVCS) to store the item files enables Doorstop to be scalable to users in multiple locations and provide offline access. The VCS can also display the differences between different versions of the requirements and tests throughout history.

5. Implementation Details

Python 3 [13] was chosen as the implementation language so that Doorstop can run on multiple platforms and be

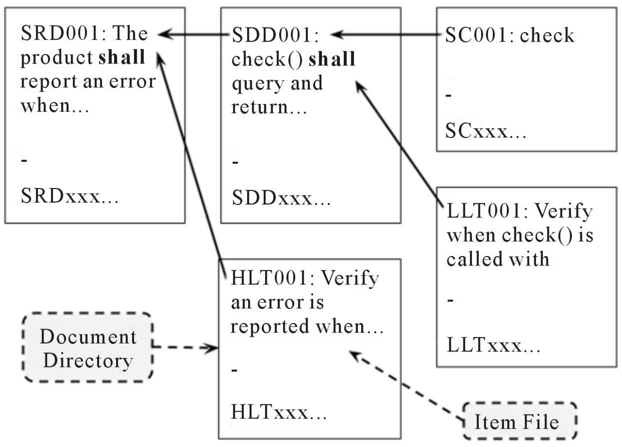

SRD001: The product **shall** report an error when...
-
SRDxxx...
SDD001: check() **shall** query and return...
-
SDDxxx...
SC001: check
-
SCxxx...
LLT001: Verify when check() is called with
-
LLTxxx...
HLT001: Verify an error is reported when...
-
HLTxxx...
Document Directory
Item File

Figure 3. Doorstop's requirements model.

easily extensible. Each text file controlled by Doorstop contains data in the YAML format [14], which is easily read by humans and parsed with a Python library. The Markdown language [15] is used for blocks of text in the YAML files. Markdown mimics standard conventions for formatting plain text and can be translated into many other markup languages for presentation.

5.1. Project Structure

The users of Doorstop designate a location to store the items of each document. These document directories can exist anywhere in the version control working copy. A sample project structure can be seen in **Figure 4**.

This figure shows a possible project structure for the example project described in earlier sections. Doorstop creates a "doorstop.yml" in each location selected to be a document directory. In this example, there are four documents: a software requirements document (SRD), high-level tests document (HLT), software design document (SDD), and low-level tests document (LLT). In the figure, each of the document directories contains a couple of item files, which were sequentially named by Doorstop using the document's designated prefix.

5.2. Text Format

When Doorstop creates a new item file in a document directory, it is assigned a unique, sequentially numbered filename. The tool populates the item file with fields to store the text of a requirement or test along with additional metadata. A sample item file can be seen in **Figure 5**.

This figure shows the standard attributes for an item file. The "level" field indicates the presentation order of the item within its document. Any number of outline depths can be used. Links are represented as a list of item file IDs (filename without the extension) or an empty list. The item's text is stored in a multi-line format so that line breaks are preserved. Item files can also include external text references as shown in **Figure 6**.

This figure shows an item file with an external reference to the string "test_check_std_args". During document validation and publishing, Doorstop will confirm this external reference is found in the version control working copy. External references provide Doorstop with a simple mechanism for linking to source code or other text files. When an external reference is defined, the item file's text becomes optional.

5.3. User Interfaces

Currently, Doorstop has two main interfaces: command-line and scripting. The command-line interface is used to create document directories, add item files, and create links between items. It can also validate the document hierarchy and provide warnings about common issues such as missing links or text. A selection of the available commands is shown in **Figure 7**.

In this sequence of commands, the SRD document is created first. Next, an item is added to the SRD document and opened for editing. After that, the HLT document is created with SRD as its parent. Finally, an item is added to the HLT document, which is then linked to an item in the SRD document.

The Doorstop Python package can also be used to interact with the document structure. After importing the "doorstop" package, item files can be manipulated and validated using the classes and functions provided. A sample script is shown in **Figure 8**.

```
req/.doorstop.yml
    SRD001.yml
    SRD002.yml
    tests/.doorstop.yml
           HTL001.yml
           HLT002.yml
src/doc/.doorstop.yml
         SDD001.yml
         SDD002.yml
   main.c
test/doc/.doorstop.yml
          LLT001.yml
          LLT002.yml
    test_main.c
```

Figure 4. Sample project structure.

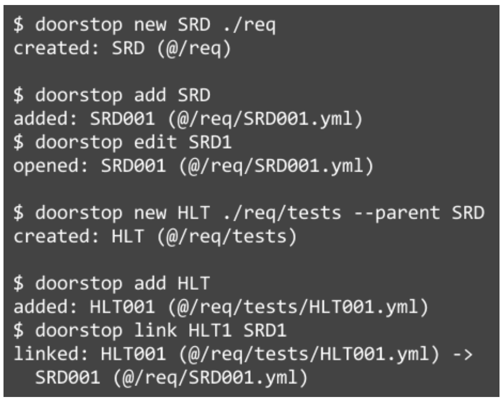
```
SRD001.yml
level: 1.1
link: []
ref: ''
test: | -
  The product **shall** report an error
  when its checksum is invalid.

   *Note: *This is a security feature.
```

Figure 5. Sample item file.

```
LLT001.yml
level: 6.2.3.
links:
 - SDD001
ref: test_check_std_args
text: ''
```

Figure 6. Sample item file with external reference.



```
$ doorstop new SRD ./req
created: SRD (@/req)

$ doorstop add SRD
added: SRD001 (@/req/SRD001.yml)
$ doorstop edit SRD1
opened: SRD001 (@/req/SRD001.yml)

$ doorstop new HLT ./req/tests --parent SRD
created: HLT (@/req/tests)

$ doorstop add HLT
added: HLT001 (@/req/tests/HLT001.yml)
$ doorstop link HLT1 SRD1
linked: HLT001 (@/req/tests/HLT001.yml) ->
  SRD001 (@/req/SRD001.yml)
```

Figure 7. Doorstop command-line interface.

```
#!/usr/bin/env python

"""
Display all requirements containing "shall".
"""

import doorstop

tree = doorstop.build ()
document = tree.find_document ('SRD')
for item in document:
    if 'shall' in item.text:
        print(item.id)
```

Figure 8. Doorstop scripting interface.

This script builds the document tree from the current working directory and locates the document with the “SRD” prefix. It then iterates through the items in the document and displays their IDs if the item contains the word “shall”.

5.4. Interaction with the VCS and Other Tools

In addition to the change management features described in earlier sections, storing requirements as text in version control provides additional benefits not possible with other methods of requirements management. Version control tools are able to provide line-by-line diffs of the item files’ contents throughout history. They also track individual modifications to lines in these item files and record both the author of each change and reason for the change. The development of requirements benefits from having access to this powerful concept, which is already used frequently while developing source code.

Because the item files are stored in an easily parsable format, other tools can be used to manipulate the files Doorstop creates. Popular command-line tools such as **grep**, **sed**, and **awk** can be used to locate and manipulate patterns of text. The item files can be opened in any text editor. Doorstop will adjust formatting of the item files and report syntax errors when it validates the document hierarchy.

6. Evaluation

Doorstop meets all requirements identified in Section 2. The composition of item files and the links between them is accomplished using the tool along with whatever editor the user is comfortable with. The presentation of requirements is provided through Doorstop's storage of text in a format that can be translated to a variety of formats. The administration of document directories and item files is handled by the version control system.

As part of the evaluation, Doorstop was used to manage the traceability of test cases during the development of Doorstop itself. Tight integration between the requirements, source code, and version control allowed the project to develop rapidly. Validation of the document hierarchy and report generation was easily incorporated into the build process.

Compared to other solutions, Doorstop has many advantages. The tight integration with source code enables tracing options not available in tools that maintain requirements in a separate database. Storing requirements in version control allows offline access and the flexibility to branch and merge the contents. The implementation in Python provides cross-platform support and scripting capabilities using a well-established language.

7. Future Work

Doorstop provides many desirable features, but users of a traditional requirements management tool might be hesitant to try Doorstop until it also provides interfaces and formats closer to what they are familiar with. A full-scale trial will also be required to accurately access these needs. Based on this conclusion, several features are planned as future work for Doorstop.

Using the existing core code, a GUI will be developed in Python to provide an alternative to command-line and scripting interfaces. Users comfortable with the text-based interfaces will still be able to use those for advanced tasks. The GUI should provide enough functionality to be useful as a primary interface.

Commercial tools provide many options for publishing requirements documents. Doorstop will likely be more attractive if it can produce reports as web pages or standalone documents. This will allow requirements to be shared with users who might not have access to the source code repository.

Finally, Doorstop will benefit from the ability to import from and export to formats understood by the most popular requirements management tools today. The import feature will be used to let a user transfer exiting requirements documents into Doorstop. The export feature will exist to provide the user an exit if they wanted to migrate to a different solution.

References

[1] Gotel, O. and Mäder, P. (2009) How to Select a Requirements Management Tool: Initial Steps. 17*th IEEE International Requirements Engineering Conference*, Atlanta, 31 August-4 September 2009, 365-367.

[2] Zhang, Q. and Eberlein, A. (2003) Architectural Design of an Intelligent Requirements Engineering Tool. *Canadian Conference on Electrical and Computer Engineering*, Montreal, 4-7 May 2003, 1375-1378.

[3] Hoffmann, M., Kühn, N., Weber, M. and Bittner, M. (2004) Requirements for a Requirements Management Tools. *Proceedings of the* 12*th IEEE International Requirements Conference*, Kyoto, 6-11 September 2004, 301-308.

[4] Zainol, A. and Mansoor, S. (2011) An Investigation of Requirements Management Tool Elements. 2011 *IEEE Conference on Open Systems*, Langkawi, 25-28 September 2011, 53-58. http://dx.doi.org/10.1109/ICOS.2011.6079304

[5] (2012) Requirements Management for Systems and Advanced IT Applications. http://www-03.ibm.com/software/products/en/ratidoor

[6] (2013) Serena Dimensions RM. http://www.serena.com/index.php/en/products/other-products/dimensions-rm

[7] (2013) PTC Integrity—Accelerating Innovation in Software Intensive Products. http://www.mks.com/platform

[8] (2013) Open Source Requirements Management Tool. http://sourceforge.net/projects/osrmt

[9] BA & Engineering Teams (2013) Requirements Management Software for Product Management. http://www.accompa.com

[10] (2013) Jama Requirements Management. http://www.jamasoftware.com/jama-requirements-management

[11] (2009) aNimble Platform. http://nimble.sourceforge.net

[12] (2012) rmtoo. http://www.flonatel.de/projekte/rmtoo

[13] (2013) Python v3.3.3 Documentation. http://docs.python.org/3/

[14] The Official YAML Web Site (2013). http://www.yaml.org/

[15] (2013) Markdown. http://daringfireball.net/projects/markdown