Example Amber Report

Gary A. Howard

June 2, 2018

Abstract

This report demonstrates the automation framework components amber, autodoc, docbld, and tlc-article.

Contents

1 Introduction

1.1 Overview

This report demonstrates the Amber driver consumes YAML Test Plans, Test Suites, and Test Cases and produces output that is properly formatted for reports designed to tlc-article and audotoc conventions.

1.2 Purpose

This Report Package is a detailed record that provides a verification test overview, a list of the software requirements tested, one or more Test Reports, evidence the Test Reports ran along with the output produced by the Test Reports including pass/fail result for each Test Step and Test Report, a statement indicating the Configuration Item has a Configuration Identification, and a conclusion that the software requirements for the target device have been verified.

1.3 Scope

This Report Package applies to a Company medical device software projects that have determined a Configuration Item must have verification tests. This Report Package covers activities associated with verifying a Configuration Item has met its software requirements.

1.4 Verification Test Objectives

1. Describe the intended use of the verification tests.
2. Set the purpose and scope for the verification test effort.
3. Enumerate software requirements.
4. Disclose compliance criteria.
5. Define verification tests acceptance criteria.
6. Identify responsible persons and their roles.
7. Document required deliverables.
8. Define specific test cases and test steps to confirm that the software requirements have been met.
9. Collect test evidence.
10. Record Verification Test conclusion.

1.5 General Terms

Configuration Control The systematic process for managing changes to and established baseline.

Configuration Identification A unique identifier used to associate a collection of software artifacts.

Configuration Items Software source code, executables, build scripts, and other software development and software test artifacts relevant to creating and maintaining a software project.

Configuration Status Accounting The recording and reporting of the information needed to effectively manage the software and documentation components of a software project.

Report Package A detailed record that provides a Configuration Item overview, a list of its Intended Use Requirements (IUR), one or more Test Reports, evidence the Test Reports ran along with the output produced by Test Report including a pass/fail result for each Test Step and Test Report, and a statement indicating the Configuration Item has a Configuration Identification, and conclusion that the Configuration Item has been validated for its intended use.

Test Plan A test plan is a collection of one or more test suites a tester has determined to use to challeng requirements.

Test Suite A test suite is a collection of one or more test cases a tester has determined to use to challeng requirements.

Test Case A test case is a set of conditions under which a tester will determine whether the test is working as it was originally established for it to do.

Test Step A unique test identifier with predetermined expectation, confirmation criteria, and pass/-fail result.

Test Report A test report consists of Detailed instructions for the set-up, execution, and evaluation of results for a given test. The test protocol may include one or more test cases for which the steps of the protocol will repeat with different input data. Test cases are chosen to ensure that corner cases in the code and data structures are covered. A test protocol may be a script that is automatically run by the computer.

1.6 Configuration Item Specific Terms

TBD You should add terms that are specific to your Configuration Item or delete terms.tex.

1.7 General Acronyms

FDA Food and Drug Administration

IUR Intended Use Requirements

LMS Learning Management System

SOP Standard Operating Procedure

SOUP Software Of Unknown Provenance

1.8 Configuraiton Item Specific Acronyms

n/a Not applicable

1.9 Verification Test Compliance Criteria

This Test Plan complies with SOP-PRC02004 Software Development Procedure SOP-PRC02004 Software Development Procedure; Software Configuration Management FDA Quality System Regulation 820.70(i) Automated Processes, and General Principles of Software Validation. Configuration Items are often used to assist with formal verification and validation, and for the management of configuration.

Configuration Items are often used to assist with formal verification and validation, and for the management of configuration. All such tools must be validated for intended use or rationale provided otherwise. Throughout this Report Package the term Verification Test is synonymous with Configuration Item.

1.10 References

1. SOP-PRC02004 Software Development Procedure
2. SOP-PRC02004 Software Development Procedure; Software Configuration Management
3. 21 CFR 820.70(i) utomated Processes, and General Principles of Software Validation
4. General Principles of Software Validation; Final Guidance for Industry and FDA Staff; January 11, 2002

1.11 Training

Company's Learning Management System (LMS) is used to record all training records. Additional training is not required for one to use this Tool or run this Report Package. It is expected that the individuals

conducting the tests are experienced in use of the process, subsequently not requiring prescriptive setup information in conducting the validation.

1.12 Verification Test Approach

This Test Plan describes a series of Test Suites, Test Cases, and Test Steps When executed, each Test Step determines if the Configuration Item satisfies one or more software requirements. When a Test Step indicates that the software requirements are satisfied, the Test Step's result is "pass". Otherwise, the Test Step's result is "fail". The computer records all "pass" and "fail" results in the Test Plan record. The Configuration Item is considered verified when all Test Steps are executed and the Test Plan record contains no "fail" results. Each Test Step that results in a deviation, observation, incident, or failure shall be represented in the final report.

1.13 Configuration Management

When a Configuration Item is changed by its manufacturer or by a Company, a Company will review the manufacturer's release notes or a Company's design history file to determine if regression testing or adjustment to this Report Package is necessary, and re-run the Report Package to confirm the manufacturer's changes or a Company's changes do not impact product operation, product quality, or quality decisions made before upgrading to a new release of the Configuration Item.

1.14 Test Plan Instructions

This Test Plan describes Test Suits, Test Cases, and Test Steps that demonstrate how the Configuration Item satisfies the IUR. Each Test Plan describes any setup criteria needed to conduct the Test Steps. Each Test Plan contains a list of IUR and the Test Steps that demonstrate how the Configuration Item satisfies the IUR. Each Test Step is marked passed or failed as it is completed. Each Test Plan is marked passed when all Test Steps pass or failed if a single Test Step fails. This serves as a record of completed Test Plans and Test Steps.

Test Plans are automatically run by the computer generating a report in PDF format. Test Plans are code-reviewed prior to use using the same code-review practice that is used for the medical device software. Test Reports are stored in the Configuration Management tool that is used by the medical device software. Test Steps that fail during computer execution or observations made during document routing, result in tickets being created to address the issue. This results in another execution of the Report Package and document routing.

When it becomes necessary to annotate a computer generated document, a Company's documentation practices must be followed. The documentation practices are summarized below:

1. When necessary to write down test values, they are written using an ink pen.
2. Any cross-outs of results or modifications to any test step shall be initialed and dated.
3. All appropriate blanks shall be filled in or annotated as “N/A” (not applicable).
4. No whiteout or other text-obscuring technique is to be used.
5. Any annotation must be signed and dated.

In most cases, a Test Plan is to be judged as "fail" if any Test Step within that Test Plan fails. If the failed Test Step does not impact proper evaluation of the safety and effectiveness of the device, the Test Plan in which the Test Step failed may be judged "pass". For such failed Test Steps a rationale will be noted in the comments section and in the final Report Package. If a failed Test Plan is re-executed a record of the failed results are included with the Test Plan when it passes and attached to the Test Plan. If a Test Plan is re-executed a new report is created and routed for approval in the document management system.

1.15 Test Plan Storage and Review

This Test Plan is part of a Company's automated validation framework. The framework consists of following parts:

LaTeX files are used to provide an Abstract, Introduction, Intended Use Requirements, Test Plan Overview, Test Equipment, Configuration Item Validation, Conclusion, and Change Summary. LaTeXfiles are converted assembled into PDF documents. PDF documents are routed using the Company's document management system for approval.

Ruby software is used to run the automated framework to collect test evidence.

Git is used as the storage repository for LaTeX& YAML files, a Git pull-request is used to review the LaTeX& YAML files prior to use.

Evidence Test Plan output includes one Test Suite, Test Plan, and Test Step, and Test Evidence.

YAML files define the Test Plan, Test Suite, and Test Steps that are processed to generate test evidence.

2 Requirements

Intended-use requirements are defined using the following story format:

As a <type of user>, I want <some goal> so that <some reason>

IUR01 As the designer, I want to demonstrate Amber can process Test Plans, Test Suites, and Test Cases so that I can product a testing report.

3 Test Plan Overview

This section describes Test Plans, Test Suites, Test Cases, and Test Steps that demonstrate how a Configuration Item satisfies the IUR. Each Test Plan describes any setup criteria needed to conduct the Test Steps. Each Test Plan contains a list of IUR and the Test Steps that demonstrate how the Configuration Item satisfies the IUR. Each Test Step is marked passed or failed as it is completed. Each Test Plan is marked passed when all Test Steps pass or failed if a single Test Step fails. This serves as a record of completed Test Plans and Test Steps.

Each Test Plan is described in its own section. The order the Test Plans are listed is the order they are run. Each Test Plan defines:

name Each Plan, Suite, and Case has a unique name.

purpose Each Plan, Suite, and Case has a purpose.

Test Steps Each step has a confirmation and expectation along with the command needed to challenge the UIR.

Objective Evidence A record the Test Plan was run along with any evidence collected while the Test Steps were run.

Traceability Suites and Cases are traced to an IUR that is challenged. IUR can be traced to multiple Suites and Cases.

Each Test Plan, Test Suite, Test Case, and Test Step has been designed to be run by the computer. However, a person may choose to manually run the Test Plans, save the test results, and generate this test report as specified in the appropriate design documentation.

The example below runs two commands: 1) git help and 2) cat /gitconfig. The output from both commands are written to the console

```
plan:
  name: A Test Plan Name
  purpose: purpose of the plan

suite:
  name: A Test Suite Name
  purpose: a suite purpose
  requirement: UIR01 and UIR02

  - case:
    name: A Test Case name
    purpose: A Test Case purpose
    steps:
      - confirm: Confirm git help is written to the console output.
        expectation: Git help is displayed.
        command: git
        argument: help

      - confirm: Confirm .git config is written to the console.
        expectation: .gitconfig is written to the console output.
        command: cat
        argument: .gitconfig
```

4 Prerequisites

This report was assembled by gathering artifacts that were produced by an automated test framework. The prerequisites are established to setup the enviornment and productd all artifacts including this report. How a computer is configured is outside the scope of this document. However, the minimum component parts are listed.

1. Ruby 1.9
2. Amber 0.0.129
3. docbld 1.0.0

4. tlc-article

5 Test Evidence

The Company's automation framework assembles the content in this section. The secion has one or more Test Plans, Test Suites, Test Cases, and Test Evidence. The evidence provided is used to conclude the Tool has meet the Intended Use Requirements.

5.1 Test Plan: unit-test

Purpose: This plan uses white box methods. The output is sufficient for DHF purposes. However, the output is generally only used by engineers.

5.1.1 Test Suite: unit-test

Purpose: This test suite ensures the system has meet the mimimum requirements for further testing.
This Text case is also demonstrating LaTeXcommands, as well as, amber's ability to substitute unit-test.

Requirement: R123, R456, R789

Test Case: t001

Purpose: Demonstrate requirement R123 is met.

Requirement: R123

```
       Step: 1
    Confirm: Program echo has been installed.
Expectation: echo installation location is displayed.
    Command: sudo which  echo
Test Result: PASS
   Evidence: Starts on next line.
/bin/echo
```

```
       Step: 2
    Confirm: Program date has been installed.
Expectation: date installation location is displayed.
    Command: which  date
Test Result: PASS
   Evidence: Starts on next line.
/bin/date
```

```
       Step: 3
    Confirm: Program man has been installed.
Expectation: man installation location is displayed.
    Command: which  man
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/man
```

Test Case: t002

Purpose: Demonstrate requirement R456 is met.

Requirement: R456

```
       Step: 1
    Confirm: Program grep has been installed.
Expectation: grep installation location is displayed.
    Command: sudo which  grep
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/grep
```

```
       Step: 2
    Confirm: Program dumper has been installed.
Expectation: dumper installation location is displayed.
    Command: which  dumper
Test Result: FAIL
   Evidence: Starts on next line.
```

```
       Step: 3
    Confirm: Program sed has been installed.
Expectation: sed installation location is displayed.
    Command: which  sed
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/sed
```

```
       Step: 4
    Confirm: Program tr has been installed.
Expectation: sed installation location is displayed.
    Command: which  tr
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/tr
```

Test Case: t003

Purpose: Demonstrate requirement R789 is met.

Requirement: R789

```
       Step: 1
    Confirm: Program nice has installed.
Expectation: nice installation location is displayed.
    Command: sudo which  nice
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/nice
```

```
       Step: 2
    Confirm: Program nl has installed.
Expectation: nl installation location is displayed.
    Command: which  nl
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/nl
```

```
       Step: 3
    Confirm: Program man has been installed.
Expectation: nroff installation location is displayed.
    Command: which  nroff
Test Result: PASS
   Evidence: Starts on next line.
/usr/bin/nroff
```

```
       Step: 4
    Confirm: Program man has been installed.
Expectation: ed installation location is displayed.
    Command: which  ed
Test Result: PASS
   Evidence: Starts on next line.
/bin/ed
```

```
       Step: 5
    Confirm: A developer is able to access Git help.
Expectation: Git help is displayed.
    Command: git  help
Test Result: PASS
   Evidence: Starts on next line.
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
```

```
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

6 Verification Test Conclusion

This Report Package has demonstrated that the Configuration Item has satisfied the software requirements described herein thus is considered verified for its intended use.

Change Summary

Change	Justification
Initial version.	New document.