Software Test Documentation

Version: 3.0

Date: 19/04/2014

Author: Youssef Boudiba (QAM)

Abstract

The software test documentation of the xiast project, a scheduling application for students, is based on the IEEE standard for software test documentation (829-1998).

Record of revision

Revision	Revision date	Description
V0.1	8/11/2013	Initial draft
V0.2	10/11/2013	Added additional content
V0.3	8/12/2013	Removing unnecessary content and updating topics
V1.0	12/12/2013	Ready for delivery: iteration 1
V2.0	02/03/2014	Specified testing framework
V3.0	19/04/2014	Revision of the whole document

---

1. Scope

This document ensure that the quality of the xiast project will be maintained by following a set of testing standards and practices based on the above mentioned IEEE document.

2. References

1. ***Software project management plan:*** https://github.com/se1-1314/xiast-docs/blob/master/management/project/spmp.md
2. ***Software requirements specifications:*** https://github.com/se1-1314/xiast-docs/blob/master/management/requirements/requirements.md
3. ***Software Design Document:*** https://github.com/se1-1314/xiast-docs/blob/8315fbf1ad93e8144d2f2e146196e5c8150084a9/management/design/design.org
4. ***Midje:*** https://github.com/marick/Midje
5. ***jasmine:*** https://github.com/pivotal/jasmine

3. Definitions

3.1. **SPMP:** Software Project Management Plan.

3.2. **SRS:** Software Requirements Specifications.

3.3. **SDD:** Software Design Document.

3.4. **STD:** Software Test Documentaion

3.5. **Software item:** Source code, object code, control data, or a collection of these items.

3.6. **Software feature:** A distinguishing characteristic of a software item.

3.7. **Test item:** A software item which is an object of testing.

---

4. Test Plan

4.1. Purpose

The test plan describes the scope, approach, resources, features to be tested, the testing tasks and the risks of the project.

4.2. Test plan identifier

XIAST_TP

4.3. Introduction

Software items and features defined in the SRS wil be tested.

4.4. Test items

Following items are tested:

Name	File type	File location	Description
query	clojure	se1-1314/xiast/src/xiast/query.clj	Provides protocols for querying and updatinginformation accessible through Xiast information stores
api	clojure	se1-1314/xiast/src/xiast/api.clj	Handles routes for the JSON api
core	clojure	se1-1314/xiast/src/xiast/core.clj	HTTP requests and handlers are processed here
session	clojure	se1-1314/xiast/src/xiast/session.clj	Contains methods to store information about the current session
scheduling	clojure	se1-1314/xiast/src/xiast/scheduling.clj	Contains several schedule proposal checks

4.5. Features to be tested

Features to be tested are listed in the SRS:

4.5.1 Functional Requirements

(section 3.2 of SRS)

4.5.2 Software system attributes

(section 3.5 of SRS)

4.5.3 Other requirements

(section 3.6 of SRS)

4.6. Features not to be tested

Source files containing: constants, translations, mockdata, getters and setters will not be tested.

4.7. Approach

Our test strategy will consist of a series of different tests. These tests are specified below and are performed by the whole team.

4.7.1. Unit testing

Every small amount of source code (or low level software module) that cannot be divided in smaller part is called a unit. Whenever a developer has finished working on such a unit, it will be tested to check whether or not it is conform to its design and requirements.

4.7.2. Integration testing

Integration testing combine and test low level software modules that have been unit tested. This test ensures that the combined higher level module still works as specified.

4.7.3. System testing

System testing tests and combines all integrated (higher level) software component. The test will be performed on the whole system in order to find deviations from the specified requirments.

4.7.4. Security testing

Detect and eliminates security errors such as invalid inputs, etc.

4.7.5. Regression testing

Whenever changes has been made to a module, tests are needed to ensure that those changes didn't introduced new faults.

4.8. Item pass/fail criteria

A test pass if no error occurs and if it's conform to its design and requirements.

4.9. Suspension criteria and resumption requirements

Test are suspended whenever a bug occures. Once the bug has been resolved, the testing activities can resume.

4.10. Test deliverables

The following documents should be delivered:

1. Test plan
2. Test design specifications
3. Test procedure specifications
4. Test logs
5. Test incident reports
6. Test summary report

4.12. Responsibilities

Each member of the team is responsible for his own sets of tests. The quality manager must perform weekly checks to ensure that the test are following the test plan.

4.13. Schedule

A set of milestones can be found at: https://github.com/se1-1314/xiast/issues/milestones. At the end of each iteration a due date is decided and depending on the activity a corresponding sets of tests must be delivered.

4.14. Risks and contingencies

Whenever a testing activity has not (or cannot) been completed, the quality manager must figure out a solution to the situation.

---

5. Test Design Specification

5.1. Purpose

To detail refinements of the test approach presented in the Test Plan.

5.2. Test design specification identifier

XIAST_TDS

5.3. Features to be tested

Features are listed in section 4.5 of the STD.

5.4. Approach refinements

Testing tools

Two testing framework are used:

1. Midje

Midje is a TDD testing framework for clojure that works by means of facts. A fact is the smallest checkable unit in midje. A fact contains one or more checkables. A checkable consist of three part: a left side, an arrow and a right side. The left side is the function to be tested (with its input if needed), the right side is the expected output. The arrow is midje's own extended equality and makes the code more readable to the user.

General structure

```
(ns a-namespace
  (:use midje.sweet))                          ;; Main namespace for Midje


(facts "Test description"                       ;; Facts encapsulates multiple fact.
  (fact "specific test case description"      ;; A fact is midje's smallest unit
test and contains checkables.
    (a-function input) => expected-output))   ;; This is a checkable.
    (provided (a-function-to-be-mocked) => mocked-data)) ;; The provided statement
enables us from mocking functions in facts.
```

2. jasmine

Jasmine is a behavior-driven development/automated framework for testing JavaScript code. It does not rely on browsers, DOM, or any JavaScript framework. With Jasmine-JQuery plugin it's also possible to test jQuery code easily.

General structure

```
// Every tests begin with the global jasmine function describe (also known as a
suite).
describe("Test description", function() {
// A suite contains it function (called spec) that describes what a part of a
program should do.
  it("contains spec with an expectation", function() {
    expect(true).toBe(true); // (straightforward) This is the matcher of the spec.
  });
});
```

6. Test Procedure Specification

6.1. Purpose

To specify a sequence of actions for the execution of a test.

6.2. Test procedure specification identifier

XIAST_TPS

6.3. Set up

All tests are located in a specific test file. The clojure code are located at: se1-1314/xiast/test/xiast.

Setting up Midje

To start using the Midje testing framework we first need to install the lein-midje plugin. We do this by adding lein-midje to the :plugins list in the :user profile in ~/.lein/profiles.clj e.g.

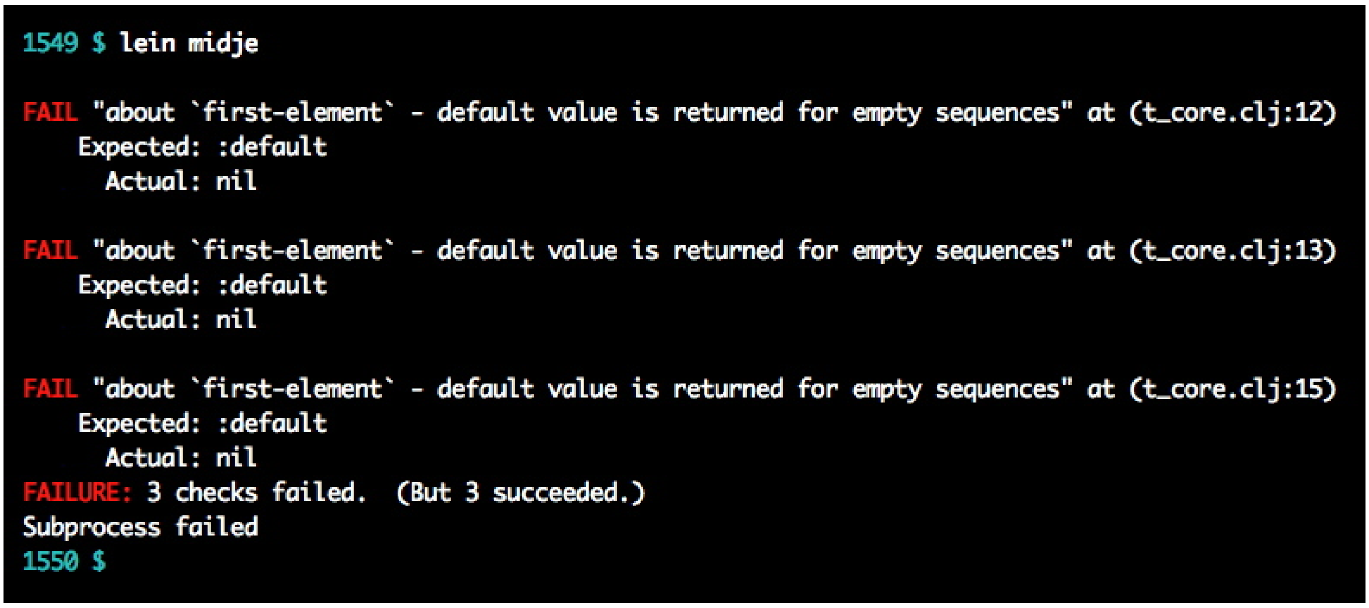
```
{:user {:plugins [[lein-midje "3.0.0"]]}}
```

6.4. Start

Start testing the clojure code with Midje by running **lein midje** in the repl.

6.6. Measure

When the tests are exececuted a similar output wil be shown in the repl:



```
1549 $ lein midje

FAIL "about `first-element` - default value is returned for empty sequences" at (t_core.clj:12)
    Expected: :default
      Actual: nil

FAIL "about `first-element` - default value is returned for empty sequences" at (t_core.clj:13)
    Expected: :default
      Actual: nil

FAIL "about `first-element` - default value is returned for empty sequences" at (t_core.clj:15)
    Expected: :default
      Actual: nil
FAILURE: 3 checks failed.  (But 3 succeeded.)
Subprocess failed
1550 $
```

Failures are highlighted in red.

7. Test Log

7.1. Purpose

Keeps track of relevant details about a test.

7.2. Outline

A test log shall have the following structure:

7.2.1. Test log identifier

XIAST_TLxxx , with xxx being an unique set of numbers (e.g. XIAST_TL000).

7.2.2 Description

The following information should be provided:

1. Indentify the items being tested including their version.
2. Indentify the attributes of the environments in which testing is conducted (framework used, etc).

7.2.3 Activity and event entries

Record the date and time along with identity of the author.

7.2.4 Procedure results

Record results of the test.

7.2.5 Environmental information

Record any changes made before the test was conducted

7.2.6 Anomalous events

Record anomalous events such as strange output, etc.

7.2.7. Incident report identifiers

Whenever an incident report is generated, record its identifier.

8. Test Incident Report

8.1. Purpose

To keep track of any incident (bugs, unexpected crashes/results of a test, etc.) that occurs during a test.

8.2. Outline

A test incident report shall have the following structure:

8.2.1 Test incident report identifier

XIAST_TIRxxx , with xxx being an unique set of numbers (e.g. XIAST_TIR000).

8.2.2 Summary

Indentify the items being tested including their version. Test log should be supplied.

8.2.3 Incident description

This description should include the following items:

1. Inputs
2. Expected results
3. Actual results
4. Anomalies
5. Date and time
6. Environment
7. Attempts to repeat
8. Testers
9. Observers

8.2.2 Impact

Describe the impact of the incident on other components.