EventNarrator Test Plan (Sample exam)

Identifier: Narrator Test Plan
Version: 0.9, 2020.03.27
Author: Bogdan

Introduction

This is a master test plan for the EventNarrator example of the ITP 2200 – Introduction to Software Testing course.
Reference documents: SampleExam.pdf defines the system under test (EventNarrator), its description and structure, and its requirements.

The relevant code for the EventNarrator that this plan refers to can be found at
https://github.com/bogdanmarculescu/itp2200
The relevant module is ex10.

Change control is handled by the author. Communication and coordination is carried out via daily standup meetings within the team.

Change control process
Updates to this document are carried out as a result of planned discussion and reflections on the current test process, its outcome, as well as a result of feedback from relevant parties. Scheduled updates are the result of sprint review meetings. Unscheduled updates are the result of feedback from relevant stakeholders (in this case, students).

Test Items

The System Under Test (SUT) is to be tested to assess the degree to which it conforms to the Requirement Specification and the Design Specification, as defined by the SampleExam document.

The SUT and relevant documentation can be freely obtained at
https://github.com/bogdanmarculescu/itp2200

Features to be tested:

Requirement 1:
Requirement 2:
Requirement 3:

As defined in the SampleExam document (for a start. Once a more well defined and detailed view of the requirements begins to emerge, this will be updated to reflect the new understanding).

Approach

Overall, the test strategy is to prioritize acceptance level expected behavior. First test cases will be developed to assess expected behavior of the system.

Unit testing will be handled at class level, focusing on a suitable coverage of relevant input space partitions.

Metrics collected: The quality of the test suite will be assessed by line coverage, measured automatically as part of the testing process.

Regression test rules: Changes and additions to the code will trigger running the entire existing test set. Failure of integration tests will freeze the integration process until relevant faults have been identified and removed. Integration can only be conducted with a test suite that passes. More severe defects will be prioritized for fixes, but integration should be contingent on a completely passing test suite.

Undefined or underdefined behavior: Elements of the requirements or design that are undefined, don't make sense, or are untestable will be discussed and clarified within the team. If necessary, the team will decide on an interpretation acceptable to team members, and document their decision and its rationale in this document.
Scheduled meetings for such definitions can take place at the beginning of a sprint. Unscheduled changes can result as a result of an emergency meeting. Emergency meetings can be triggered by any team member, upon deciding that the requirements confusion is urgent enough that any delay will result in a failure to meet the deadline.

Regular meetings will take place as scheduled, over Skype. In cases of emergency or when time critical decisions are needed, extraordinary meetings can be called, during working hours.

Pass/Fail Criteria

Unit tests
Unit tests will be deemed to be passes of all assertions present in the test are true. This means that the desired behavior of the system, as defined in the requirements and assessed in the tests, matcher the implemented behavior. A unit test that lacks assertions is not considered a useful test, even if it does not fail.

Each class should have at least one unit test. Unit tests should be linked to requirements via the acceptance test criteria defined below. Individual tests will be independent (i.e. they will not depend on other tests) and they will be run both as part of a test suite and individually.

The line coverage for unit test should be maximized, with at least 50% line coverage.

Acceptance tests

Requirement	Criterion	Test Cases

1.Based on the date of a past	a. For an even between 1 and -30 days in the past, the string	elapsedTimeSince_Days()
		
		elapsedTimeSince_Days_lessTimeSensitive()

event, have a method that returns a String, containing the amount of time that has passed since that event.	should be of the form “The **eventName** was **X** days ago”. Where **eventName** is the name given to the event, and **X** is the appropriate number of days.	*Note, as discussed in that seminar, it may be a good idea to separate valid and invalid inputs, and tests addressing different criteria in separate test methods.
	b. For an even between 1 and -12 months in the past, the string should be of the form “The **eventName** was **X** months ago”. Where **eventName** is the name given to the event, and **X** is the appropriate number of months.	testElapsed_months()
	c. For an even between more that 12 months in the past, the string should be of the form “The **eventName** was **X** years ago”. Where **eventName** is the name given to the event, and **X** is the appropriate number of years.	testElapsed_years()

Test deliverables

Test plan (including the above table, containing an up to date connection between requirements, criteria, and the test code).

Test reports – in the case of a failing test, a report will be communicated to the responsible for the module or class under test. This report will contain:

- The failure message from the test code
- In the case of a crash, a stack trace to help identify where the failure happened.
- The exact test case that was being run, the input values, and the expected output values.

Note: Since development for the exam topic is expected to be carried out as a small cross-functional team, these reports do not have to be formal documents. Your teams will have to establish how such failures are communicated on a regular basis. If such failures are easily found and quickly solved, then communication to the person responsible for the relevant class can be just a verbal note of the failure found. In the case of more troublesome failures, however, it may be a good idea to have a log of known faults.

If a fault is not trivial (if it will require a significant amount of time to fix), then the team will decide together (based on their own judgement of the time available, and of their current planning) if the fault is to be fixed as soon as possible, planned for some time in the future (for example, “Issue 2 for the EventNarrator will be addressed only after the complete implementation of Requirement 1, since requirement 1 is of higher priority”).

Except in the case of system-breaking bugs, the team will prioritize delivering critical functionality first, then ensuring the quality of critical functionality, then implementing and testing non-critical functionality, then other bug fixes and performance improvements.

Test Tasks

Due to the relatively simple nature of the project, there is little dependency to other tasks.

A unit (class, module) will not be deemed complete until all the relevant unit tests are in place, the minimum coverage requirement is exceeded, and all the tests pass. Further tests should be added when relevant.

Responsibilities

Bogdan Marculescu is responsible for the testing and test process of the EventNarrator.

Note: In team projects, I will expect all the relevant roles mentioned in the plans to be filled. You can skip some, but be aware that you have to justify your decision. Some roles are critical: solving scheduling conflicts and critical go/no-go decisions, are such roles. (Critical go/no-go decisions refer to decisions that will crop up that are not covered in any plans, but still must be taken).

Note: If a person is responsible for something, that does not mean that they work alone on the topic. For example, Bogdan is responsible for deciding if a new module will be introduced – a decision which is not accounted for in the plan. The team will convene, discuss, and reach a decision together. Bogdan is responsible for setting up the discussion and ensuring that the team reaches a decision, recording that decision, ensuring everyone understand the decision in the same way. He neither takes the decision alone, nor imposes his own views on the team.

Schedule

Normally schedule concerns are part of a large project plan. For example, if a project plan milestone is that the Event class is ready by April 1st, that also imposes scheduling restrictions on the test plan.

In this case, the Event class being ready by April 1st also means that all the relevant unit and integrations test are ready, achieve the desired coverage, and pass, before that date. If any faults or bugs are found, the team must decide if they will they will take the time to address the fault (and therefore postpone delivery of the module) or not.

Note: Both decisions have consequences: allowing the delivery deadline to slip may cause further delays. Keep in mind that the deadline for the exam is hard (so not delays on the final deadline are acceptable). Not allowing deadlines to slip may cause the system to be delivered with faults. Depending on the faults, this may be a significant problem.

Note: if you find yourselves in a situation where you have to deliver a system (for example, the exam deadline has arrived and you submit what you have) with known faults, it is a good idea to write up the known faults. This will both help the customer understand what to expect, as well as help the team, should you want to fix those faults (or further maintain or develop that system in the future).

Since the exam has no project plan, you are free to plan your development and testing together, as you see fit.

Risks and contingencies

The main risk for the EventNarrator is that, due to deadline slippage and limited time during the lectures and seminars, the project may not be complete. This risk is severe (so it will have significant consequences) and likely.

In this case, priority will be given to fully implementing requirement 1, with all the relevant tests. This will serve as an example for the following two requirements. This falls under "the scope of the plan may be changed".

Note: often, teams promise (and quite often do) work overtime. This is possible, but overall not recommended. I would advise the teams to try other methods first, and leave this as a last resort. Overtime has an impact on team morale, productivity, and the quality of the overall work. Some contingencies are not possible during an exam: for example, adding resources to the team or changing the delivery deadline are not possible in an exam.

Approvals

The process is complete when approved by the entire team, in a common (online) meeting.