Writing Test Plans and Test Implementation

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

Outcomes

At the end of Today's Lecture you will be able to:

- Understand the issues surrounding test plans
  - Issues with Testing Documentation
  - Current Standards
  - Types of Test plans
- Understand issues in Testing Implementation
  - Specifically issues with Integration Testing
  - Approaches to selecting which classes to test first

Writing Test Plans

Testing Documentation

- Organizations tend to document testing using
  - Test Plans
  - Test Plan Reporting
- But, focusing too much on documentation leads to:
  - An environment that produces a lot of **meaningless reports**
  - An environment wherein **nothing gets done**
- Test Plan Contents
  - How the tests were created
  - Why the tests were created
  - How the tests will be run

Standardization

- Companies and organizations have developed many different templates/outlines for test plans
- Unfortunately they are too numerous to discuss here
- Thus we will focus on the IEEE standard -> IEEE 829-2008
  - Current version
  - Original was from IEEE 829-1983
  - Updated in both 1990 and 1998 prior to current edition

Standardization

According to IEEE 829-2008, a **Test Plan** is:

- Ⓐ “A document describing the scope, approach, resources, and schedule of intended activities. It identifies test items, features to be tested, the testing tasks, and any risks requiring contingency planning.”
- Ⓑ “A document that describes the technical and management approach to be followed for testing a system or component. Typical contents identify the items to be tested, tasks to be performed, responsibilities, schedules, and required resources for testing.”

Types of Test Plans

The IEEE 829-2008 identifies two types of test plans:

❶ **Master Test Plan (MTP)** - provides overall test planning and management document for multiple levels of test.

- Useful for either a single project or multiple projects within an organization

❷ **Level Test Plan (LTP)** - describes testing at a particular level.

- Describes the scope, approach, resources, and schedule of testing activities for the level of testing
- Defines items tested, features tested, testing tasks, who is responsible, and risks

Level Test Plan Template

- Consists of four main sections:
  - ❶ **Introduction** - places the test activities in the context of the overall project
  - ❷ **Details For This Level of Test Plan** - describes the general test approach and criteria for test completion
  - ❸ **Test Management** - what will be done when and who will do it
  - ❹ **General** - general testing information including QA procedures, metrics, glossary, etc.
- The following slides describe these sections in more detail

1. Introduction

Should contain the following subsections:

❶ **Document identifier** - documents unique name which encodes document info such as date, author, etc.

❷ **Scope** - describes what is being tested for the document level

❸ **References** - related documents separated by internal and external

❹ **Level in the overall sequence** - presents a figure representing how the testing level described fits within the overall project dev and test structure

❺ **Test classes and overall test conditions** - describes what is unique about the testing activity documented. General this describes what should be tested or the test criteria used

2. Plan Details

Should contain the following subsections:

1. **Test items and their identifiers** - identifies the system under test (or component/subsystem) and also documents details about the component under test (i.e. how to install it, run it, and any environmental needs it has)
2. **Test traceability matrix** - documents the origin of each test (i.e. requirements, test coverage requirements or design elements)
3. **Features to be tested** - explicitly lists all features to be tested using names that are referenced in other documentation
4. **Features not to be tested** - identifies everything that will not be tested and the reason it will not be tested

2. Plan Details

- ❺ **Approach** - describes how this testing should be carried out, including test criteria, level of automation, etc.
- ❻ **Item pass/fail criteria** - identifies the criteria for each item to be tested to identify when it is deemed to have passed testing
- ❼ **Suspension criteria and resumption requirements** - defines criteria for when to suspend testing and wait for dev team to correct the problem
- ❽ **Test deliverables** - documents all data that is delivered during testing

3. Test Management

Should contain the following subsections:

❶ **Planned activities and tasks; test progression** - describes the tasks that must be done to plan for testing and carry out testing. Identifies any inter-task dependencies and constraints.

❷ **Environment and infrastructure** - Describes the test environment, including anything needed before running tests (hardware, software, database, support tools, results capturing tools, privacy issues, security issues)

❸ **Responsibilities and authority** - identifies who is responsible for managing, designing, preparing, executing, checking results, and resolving problems found during testing.

❹ **Interfaces among the parties involved** - describes how the people involved should communicate

3. Test Management

❺ **Resources and their allocation** - describes any needed resources not previously identified

❻ **Training** - identifies the knowledge, skills and training the test personnel need and where this knowledge can be obtained

❼ **Schedules, estimates, and costs** - provides the schedule for testing, including preparation, design and execution of tests. Identifies the major test milestones.

❽ **Risks and contingencies** - identifies any risks that can be foreseen, and provides suggestions to avoid, mitigate and recover from these risks.

4. General

Should contain the following subsections:

❶ **Quality assurance procedures** - describes plan for QA of the testing effort

❷ **Metrics** - how testing is measured and reported

❸ **Test coverage** - describes how coverage is measures and coverage requirements

❹ **Glossary** - list of terms and their definitions (includes acronyms)

❺ **Document change procedures and history** - documents changes to the LTP document

Test Implementation

Test Implementation

- To gain immediate feedback
  - All code must compile
  - Test must not cause collateral damage
  - The process must be repeatable
  - Must compile in a timely manner
- These constraints do not typically pose a challenge to unit testing
- The challenge occurs when testing a fully integrated system

Integration Testing

- **Integration Testing** - The testing of incompatibilities and interfaces between otherwise correctly working components
  - Goal: assure the correct integration of subcomponents into a bigger working component.
- Integration testing is often done with an incomplete system
  - Testers may be evaluating how components work together
  - Testers may be testing integration aspects before the system is complete
  - Testers may be putting the system together piece by piece and evaluating how each component fits within the system

Components

- **Component** - a piece of a program that can be tested independently of the complete program or system
- A Component can be:
  - Classes
  - Modules
  - Methods
  - Packages
  - Non-executable software artifacts such as XML, JSON, or YAML files
  - XML Schemas
  - Databases

Integration Order

- When integrating components, the order classes or subsystems are integrated and tested matters
- For example:
  - If class `A` uses methods from class `B`, and `B` is not available, then we need test doubles for those methods in order to test `A`
  - Instead, if we test `B` first, then when `A` is tested we can simply use actual objects of `B` rather than doubles

CITO

- The previous slide exemplifies the **class integration test order** **(CITO) problem**
- CITO typically applies to components rather than classes
- The general goal is to:
  - Test classes/components in the order that requires the least scaffolding, or additional software
    - Creating test doubles is considered to be a major cost of integration testing

CITO

- Another way of think about this is based on the dependencies of a class.
- **Dependency Graph** - a graph where the nodes are classes, and the edges are dependencies (couplings) between classes
- Using this graph
  - If there are no cycles, then integration testing is easy
    - Classes with no dependencies are tested first
    - Testing proceeds by following a topological sort of the graph
    - Each subsequent class is integrated and the integration tested

CITO

- Cycles in the dependency graph make things more complicated
  - The tester will be required to choose where to “break the cycle”
    - That is selecting one class in the cycle to test first
  - Here some sort of stubbing or test doubles will be required to handle missing functionality
- The good news is
  - Most class diagrams have few if any cycles
  - As designers we should attempt to avoid cyclical designs in the first place

Are there any questions?

ROAR