D-BUG

TEST SPECIFICATION REPORT FOR

TSL-KINECT

DUYGU ARALIOĞLU
BEDİA ACAR
ZÜLFÜ ULAŞ ŞAHİN
GÜLNUR NEVAL ERDEM

MIDDLE EAST TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
2012

Table of Contents

1. Introduction

TSL-Kinect is an application for recognition of Turkish Sign Language with a user-friendly interface. In this project apart from gesture recognition, visual impact of the interface, its smooth control via gestures and design of the game module for teaching TSL have an extra important for the success and intended use of the application. In order to develop a durable, consistent, reliable and well-integrated system, there has to be a testing process with a satisfying procedure and plan. This TSR document is prepared for that purpose. In this document, after a brief introduction, we will present our test plan by describing overall testing strategy, and detailed test procedure including test tactics and test cases for the project. Then, information about testing resources and staffing, test work products, test record keeping, and test log will be given. Moreover, organization and responsibilities will be clearly explained. Finally, our test schedule for the whole test period will be described.

1.1. Goals and Objectives

While developing software, the testing phase is significant to ensure from the quality of the product. We will test the modules of the system individually to verify that they works correctly as expected. Then, their communication and performance as a whole is tested to release a final product which has no bugs, well-integrated, consistent and user-friendly.

1.2. Statement of Scope

This document is prepared for the specification of testing process of TSL-Kinect project.
In this document, we mainly focus on:

- What is to be tested
- The major constraints in the phase of testing

- The method of handling bugs (user-facing / internal)
- The testing strategies & procedures
- The responsible team members from each tests
- The testing schedule

1.3. Major Constraints

The major constraints that will impact the manner in which the software is to be tested are conceived as follows:

Time:

Optimization of all modules for better performance and any possible speed up in the implementation have an essential importance because of the limited time for the project to be completed. So in TSL-Kinect implementation and testing are done in parallel. In other words, whenever a new functionality is added to the system required tests are done immediately.

Data:

Since TSL-Kinect is composed of many modules running concurrently and passing data to each other during the run, minimizing the amount of data being transferred between modules is a goal in the sense of data constraint.

Number of People:

Since the main functionality of the project is gesture recognition and recognition ability of the system mostly depends on users' physical characteristics, the number of people with different characteristics will speed up the testing process for recognition.

Hardware Device Capabilities:

As developers, we have to rely on the quality of Kinect's motion sensors and at that stage

we are at the receiving end of the hardware testing for Kinect since it is not possible to use any other camera other than Kinect. However, we are trying to implement a real-time system, so project is dependent on CPU capabilities. During both functional and behavioral testing, since we use hardware devices testing of the project will be affected.

1.4. Definitions, Acronyms and Abbreviations

TSL: Turkish Sign Language

TSR: Test Specifications Report

TRAC: Trac is an open source, web-based project management and bug-tracking tool.

1.5. References

[1] Test Specification Template, METU Computer Engineering, Spring 2012

[2] Pressman, Roger S. Software Engineering: A Practitioner's Approach, Sixth edition. New York, NY: McGraw-Hill

[3] IEEE Standard for Software Test Documentation

[4] Detailed Design Report prepared by D-BUG

2. Test Plan

In this section we provide the description of the overall testing strategy and the project management issues that are required to properly execute effective tests.

2.1. Software to be tested

This application consists of four major modules namely; InputHandler, InterfaceHandler, Recognizer and HMM. All modules will be tested whenever a new property is added to the system since modules work in cooperation. Moreover, the final release will be tested sufficiently in order to provide an application with user-friendly and error-free.

2.2. Testing strategy

In this part, the overall strategy for testing process of TSL-Kinect is identified. First of all, unit testing is performed for each module separately. After integration of the whole system is finished, integration test will be done. Then, we will check whether the all requirements are provided or not by validation testing. Finally, high-order testing will be conducted to ensure the robustness of the whole application.

2.2.1. Unit testing

TSL-Kinect has four modules and all modules need to be tested separately. During unit testing of a module, other modules will be considered as working properly since each module has dependency on one or more other modules. The purpose of this phase is to ensure that each component does not have any internal errors.

2.2.2. Integration testing

After all modules passed the unit testing, they need to be tested while they are running concurrently and communicating to each other. Since the InputHandler module is running along the center of the system, the integration of InputHandler module with the other components need to be tested carefully. Moreover, InterfaceHandler module is the one which activate and deactivate Recognizer module and uses the return values coming from Recognizer, so concurrent working tests of InterfaceHandler and Recognizer modules are

to be handled. Recognizer module composed of HMM sub-module, so their collaboration also needs to be tested.

2.2.3. Validation testing

Validation testing will show whether the expectations stated in design phase are met or not. All the functional and non-functional requirements of our project is defined in the Software Requirements Specification document. The requirements stated in this document should be handled one-by-one and all must be satisfied by final product of TSL-Kinect project.

2.2.4. High-order testing

While performing unit testing and integration testing, various high-order testing strategies also need to be performed. Those testing strategies are performance testing, stress testing and alpha & beta testing.

- **Performance Test**

Performance Tests will be made to see whether the program runs in real time as it should be and especially for the “Education mode” performance of the practice game will be tested on the different hardware.

- **Stress Tests**

Although the software being tested is not “mission critical”, i.e., failure of the software would not have disastrous consequences, because no developer wants to see the crash of his/her software, stress tests will be performed to see the limits and endurance of the project, without crash.

- **Alpha and Beta Tests**

For the “Communication mode” application will be tested by speech-impaired people to see whether their gestures are recognized correctly by the system. For the “Education mode” application will be tested by speech-impaired people or anyone who has the ability to perform TSL.

2.3. Test metrics

Following test metrics will be used during the testing activity:

- Number Of Test Cases Executed
- Number Of Test Cases Passed or Failed
- Number Of Bugs Detected
- Number Of Bugs Fixed
- Number Of Priority Bugs Fixed

2.4. Testing tools and environment

As a testing tool, we will only use the debugger of Microsoft Visual Studio 2010 Express to check minor errors while developing the application. Since our product is developed in Windows 7 and Kinect SDK works only on Windows 7, we do not perform the test in any different environment in terms of operating systems. However, we will test the software with different computers for performance measurements. Moreover, we will test the system with different people to check the recognition function works correctly because our project mainly depends on users’ physical characteristics.

2. Test Procedure

In this section, the test procedure including test cases and tactics for the software will be explained in detail.

3.1. Unit test cases

There are four main components to be tested individually.

3.1.1. InputHandler

The test cases to be used are:

- It correctly calculates hand positions scaled with the size of the screen box
- SpaceZoningMatrix is created correctly
- The positions of the both hands are mapped to the appropriate number in the SpaceZoningMatrix.
- Efficient and correct noise elimination

3.1.2. HMM

The test cases to be used are:

- The related information about any gesture like hand states in SpaceZoningMatrix, time between states, number of squares passed, hand angles is given correctly.
- The probability threshold for each gesture should be sufficient.

3.1.3. InterfaceHandler

The test cases to be used are:

- Render and display video on the back stage
- The hand positions and movements of the hands are represented in the correct place in real time.
- All the buttons and other components of the user interface are represented correctly.
- The clicking function of the buttons (represented as ordinary rectangles on the user interface) work properly
- The current mode of the program is activated when the button for this module is clicked.
- When a new interface mode is activated, related text-boxes and button images are shown on front stage
- Effect functionalities for every menu objects run correctly at the appropriate time
- The string that the recognizer module served is displayed on the correct textbox
- Practice game runs with the gesture recognition in parallel without any latency

3.1.4. Recognizer

The test cases to be used are:

- Correctly arrange and manipulate the input stack for HMM
- Picks the most likely HMM structures for the gestures
- Memory management for stack variables to provide against memory leaks
- Return the correct most likely gesture name

3.2. Integration testing

Since there is a strong communication between modules, after integration process, they must be tested as a whole system. The test cases to be used are:

- Does the InputHandler module pass the required data to InterfaceHandler and Recognizer correctly and in a reasonable time?
- Does the InterfaceHandler module activates and deactivates the recognizer properly?
- Does the Recognizer module starts as soon as it is activated by InterfaceHandler?
- Does the Recognizer module stops as soon as it is deactivated by InterfaceHandler?
- Does the HMM module return the recognized gesture string in a reasonable time for the Recognizer module?
- Does the Recognizer module return the recognized gesture string in a reasonable time for the InterfaceHandler module?
- Does the InterfaceHandler module response in a reasonable time as Recognizer module sends its output while playing the game?

3.3. Validation testing

During the development process validation testing procedure will be performed to determine whether the software satisfies the specified requirements and design constraints in the previously written Software Requirements Specification and Detailed Design Reports. The test cases to be used are:

- Are the functional requirements consistent with the reports?
- Is the data design consistent with the reports?
- Is the architecture design consistent with the reports?
- Does the software meet the expectations from the perspective of the intended use?

3.4. High-order testing (a.k.a. System Testing)

3.4.1. Performance Tests

The test cases to be used are:

- Does the project work in real time?
- What is the speed of recognizing a gesture?
- What is the speed of interface mode changes?
- Is the slide effect in sub-menus realistic in terms of speed?
- Does the practice game run without any crash while running concurrently with gesture recognition?

3.4.2. Stress Tests

The test cases to be used are:

- How differs the performance of the software on computers that have significantly fewer computational resources (such as memory or disk space) then the computers used for testing?
- Does the running time of the application affect the performance and speed of response or cause any memory leaks?

3.4.3. Alpha/Beta Tests

The test cases to be used are:

- In alpha testing, application will be tried among group members and close friends
- Bugs and design problems will be fixed if observed after alpha testing

- After that beta version will be released and will be shared with people who deal with gesture recognition applications by using Kinect
- Beta version will also be tested by speech-impaired people, since meeting their needs is our basic motivation; their criticism has an extra importance for us.
- With the feedback of the experts and the target users, the problems in the application will be solved and the final product will be released.

4. Testing Resources and Staffing

The unit tests of modules are to be performed by the non-developer members of the module, in order to gain more critical view on each module. Integration tests, on the other hand, will be held by the participation of all group members. Again, validation testing will be conducted by the non-developers of the related module for getting the benefit of outside perspectives. System testing will be conducted by the whole team.

5. Test Work Products

The result of the testing process is the identification of the bugs. When a team member finds a bug, s/he will assign the correction of the bug to the developer responsible for that module by using the TRAC system.

6. Test Record Keeping and Test Log

After a team member finds a bug, it must be recorded in the TRAC system and the related ticket must be opened. When this bug is resolved, the new version of the application will be uploaded to SVN to get versioned and archived software product.

7. Organization and Responsibilities

The testing will be done by all the group members, yet since the non-developers are responsible for testing of each module an arrangement is needed. Each member responsibilities for testing different modules are illustrated in Table 1.

Module Name	**Responsible Team Members**
InputHandler	Ulaş,Neval
InterfaceHandler	Bedia,Duygu
Recognizer	Bedia,Neval
HMM	Ulaş,Duygu

Table 1- Module Testing Teams Assignment

8. Test Schedule

Table 2 shows a detailed test procedure schedule:

TASK	**DEADLINE**
Unit Test	07.05.2012
Integration Test	12.05.2012
Validation Test	16.05.2012
Performance and Stress Test	20.05.2012
Alpha Beta Test	23.05.2012
Bug Corrections	26.05.2012

Table 2 - Test Procedure Schedule