PRIMER:
Test Plan

EE Senior Design

Objective

- Brief introduction to Design For Testability (DFT)
- Prepare you to write your Test Plan
- Show the relationship between the Functional Specification and the Test Plan
- Introduce the Test Plan Template

TEXAS STATE

Introduction to DFT

- Design For Testability (DFT), and Validation/Testing have become tremendously important facets of design methodology
  - Devices are increasingly more sophisticated
  - Must be able to
    - Validate that the design meets specs and is bug-free
    - Test the device in a manufacturing sense

- As designers you will need to know DFT

Background

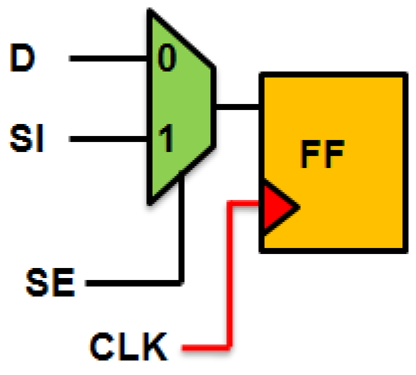
- Design and test used to be separate
  - The final quality of the test was judged by the percentage of defective parts shipped to the customer
  - Parts per million (PPM) was a test score
  - This approach worked for simple systems
- In the 1980's, fault simulation was used
  - It only improved fault coverage to about 80%
- As a result, the increasing test costs and decreasing test quality lead to DFT engineering

Background

- Various testability measures and testability enhancement methods were attempted
  - To improve testability
  - To facilitate sequential Automatic Test Pattern Generation (ATPG)
  - Difficult to reach more than 90% fault coverage

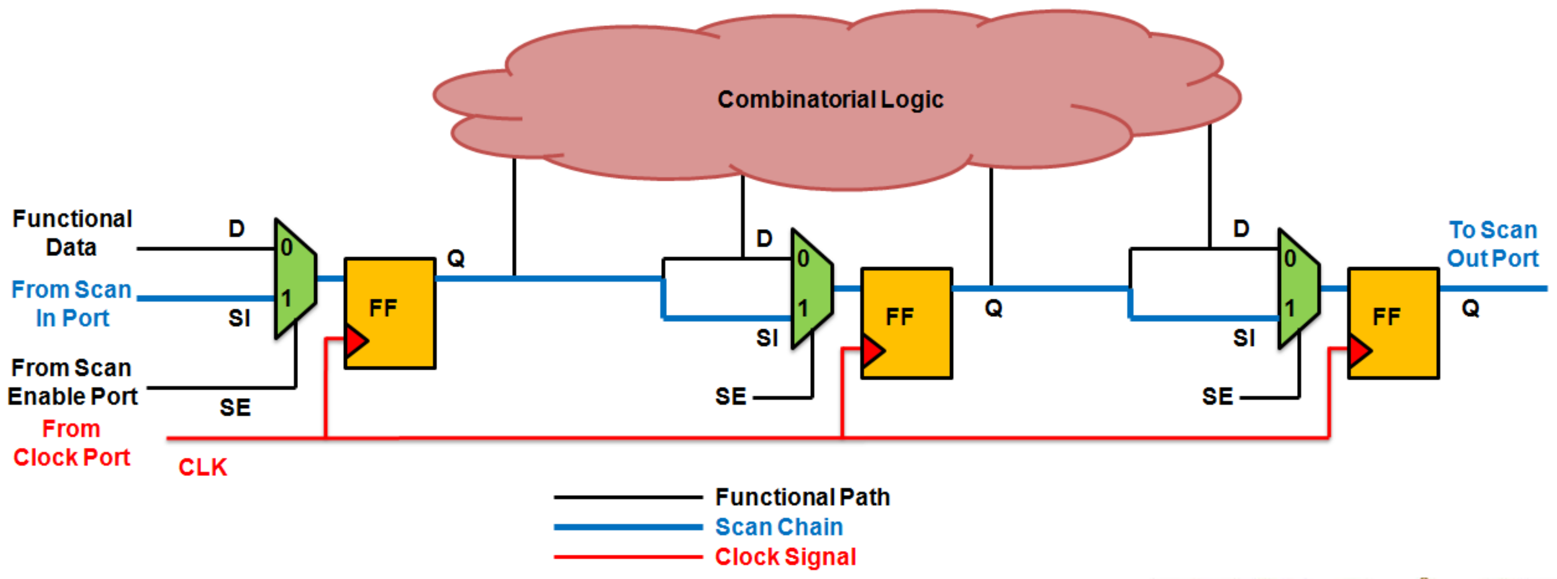
- Structured DFT
  - Aimed to ease the difficulties in controlling and observing the internal states of sequential circuits
  - Scan design is most popular structured DFT approach

Scan chain design



Scan flip-flop

Implementation



Two rules in testing

1. **If you design a testability feature, you probably won't need to use it.**
   - Corollary: If you omit a testability feature, you WILL need to use it

2. **If you don't test it, it won't work,** guaranteed.

Two Checks

- Two basic forms of validation
  1. **Functional test**: Does this ***design*** produce the correct results?
  2. **Manufacturing test**: Does this particular ***unit*** work? (Can I sell it?)

- Functional test seeks correctness
  - May be 50 person-years of effort
- Manufacturing test is done on each unit
  - May send the unit through a burn-in oven and a tester before selling

Types of testing

- Functional test consumes much in the way of labor and cost
  - Architectural validation teams are large & lengthy
  - Embed much in the way of testing
  - Make it a priority to test a new design once available

- Manufacture test constrains high-volume production flow
  - May have to run many tests to identify ‘frequency bins’ (among others)
  - Automated Test Equipment (ATE) can cost several millions of dollars

Senior Design

- Up until now we’ve largely been discussing digital systems
- Mixed signal, and software, systems also need to be testable

- In Senior Design we don’t have the time/resources to do much in the way of DFT
  - However, we planned for how we will test each feature as we create the Functional Specification

Senior Design DFT Examples

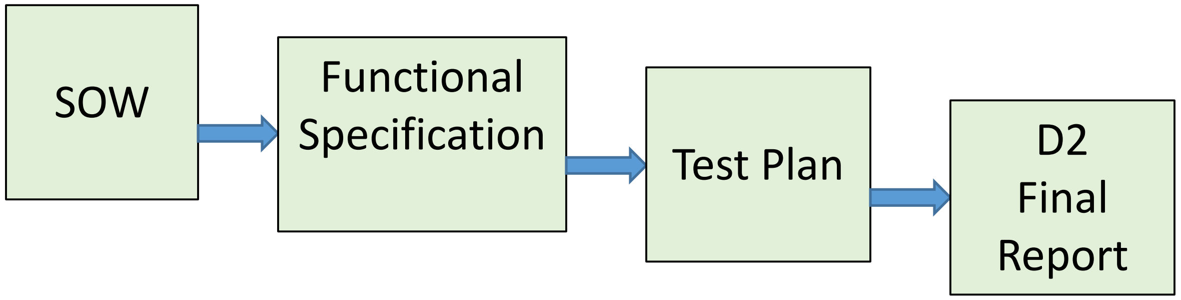
- Modular code using simulated inputs
- Simulated (created) sensor outputs driving inputs
- Physical test points on both sides of boards
- Artificially generated noise / corruption signals

- A plan to test the bits and pieces, and interfaces, <u>**BEFORE**</u> putting the entire system together!

TEXAS STATE

Flow of Documents

- One document sets up, and flows into, the next
  - The Functional Specification is based upon the SOW
  - **The Test Plan tests what is in the Functional Specification**
  - The Final Report states what did and did not meet requirements as stated in the Test Plan, and is the overall summary

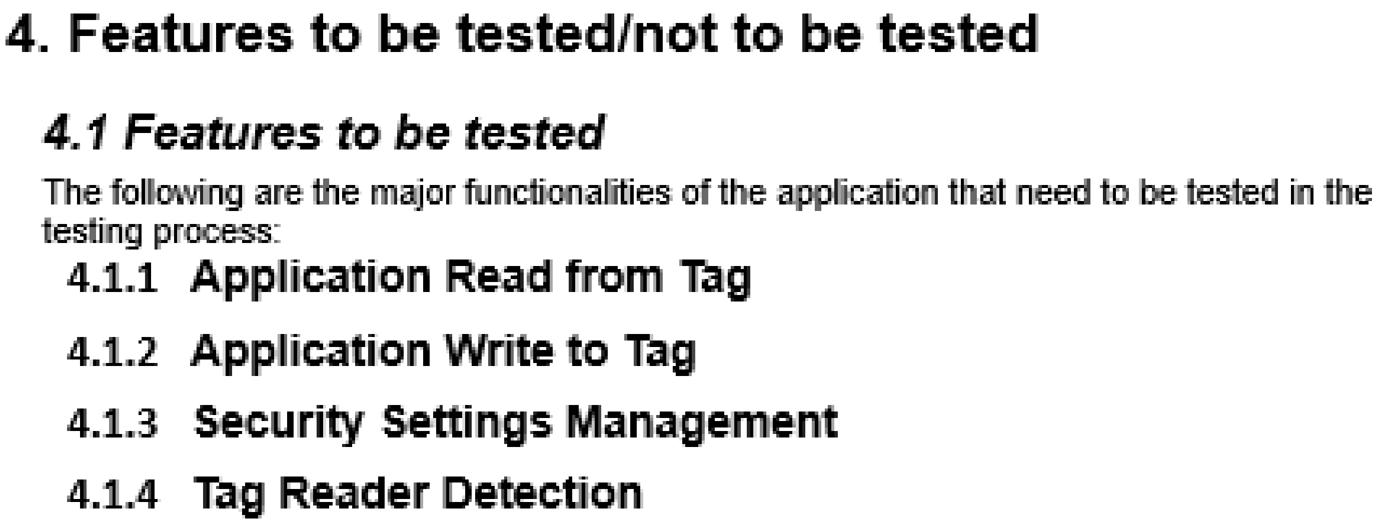


Steps in writing a Test Plan

1. **Take every line item from your Functional Spec**
    - Line Item = performance item
    - State how it will be tested, along with a brief description, or
    - State why it will NOT be tested, along with a brief description
2. **State the Pass/Fail criteria you will use**
    - Will differ between hardware and software
    - Allow for marginal performance as appropriate
3. **State the test approach**
    - How and where the testing will be performed
4. **Don’t test off-the-shelf components unless they are connected to other components**
5. **State the materials (hardware & software) needed to perform the testing**
6. **Write each test item as a test case**
7. **State the testing schedule**

Example: Features to be tested

- From a previous project:



4. Features to be tested/not to be tested

4.1 Features to be tested

The following are the major functionalities of the application that need to be tested in the testing process:

4.1.1 Application Read from Tag

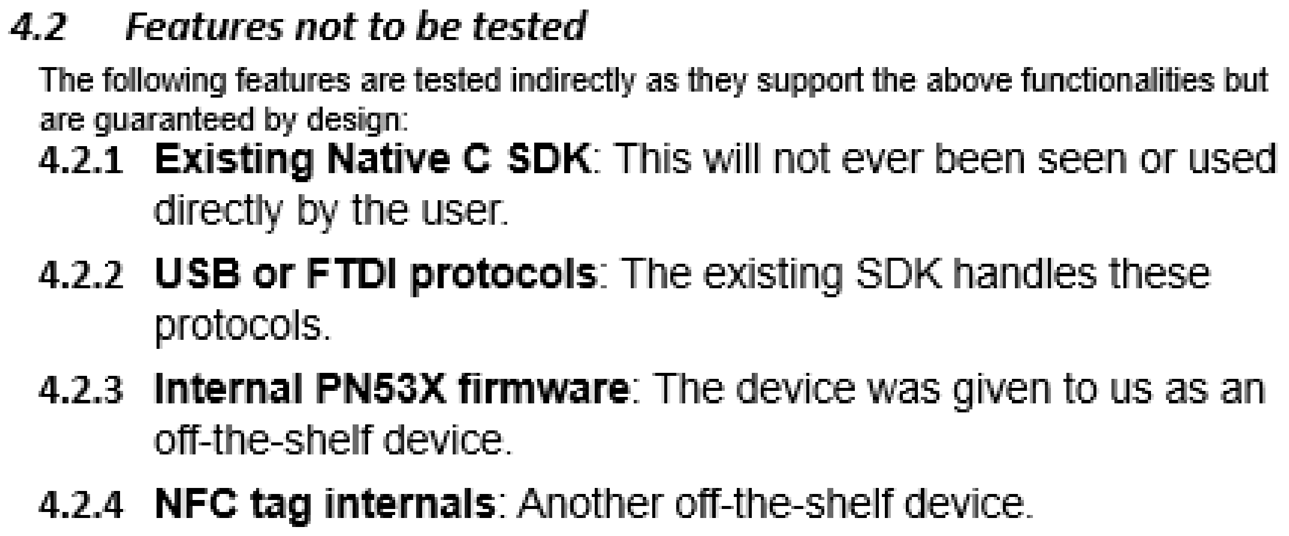
4.1.2 Application Write to Tag

4.1.3 Security Settings Management

4.1.4 Tag Reader Detection

Example: Features not to be tested

- From that same project



4.2 Features not to be tested

The following features are tested indirectly as they support the above functionalities but are guaranteed by design:

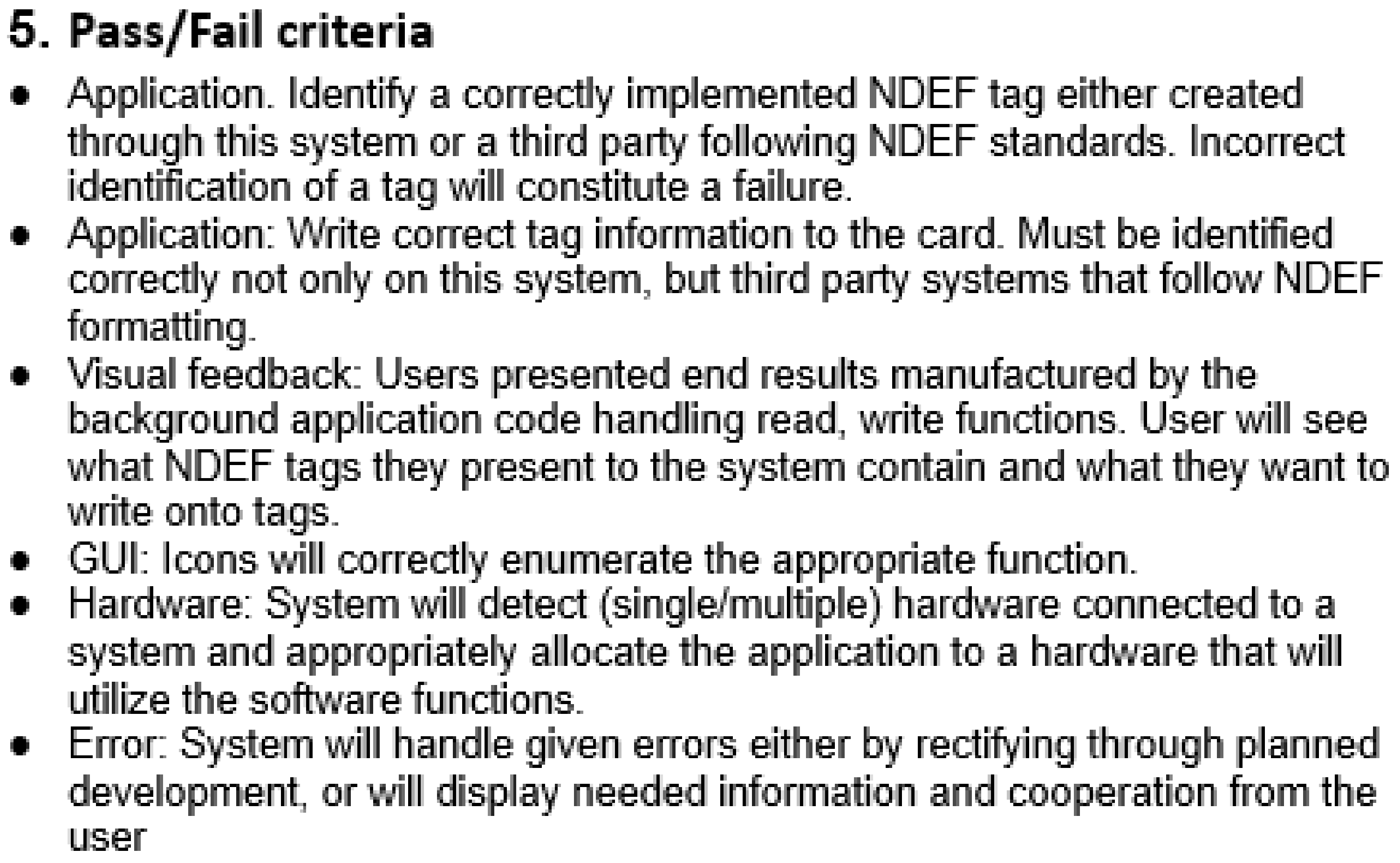
4.2.1 **Existing Native C SDK**: This will not ever been seen or used directly by the user.

4.2.2 **USB or FTDI protocols**: The existing SDK handles these protocols.

4.2.3 **Internal PN53X firmware**: The device was given to us as an off-the-shelf device.

4.2.4 **NFC tag internals**: Another off-the-shelf device.

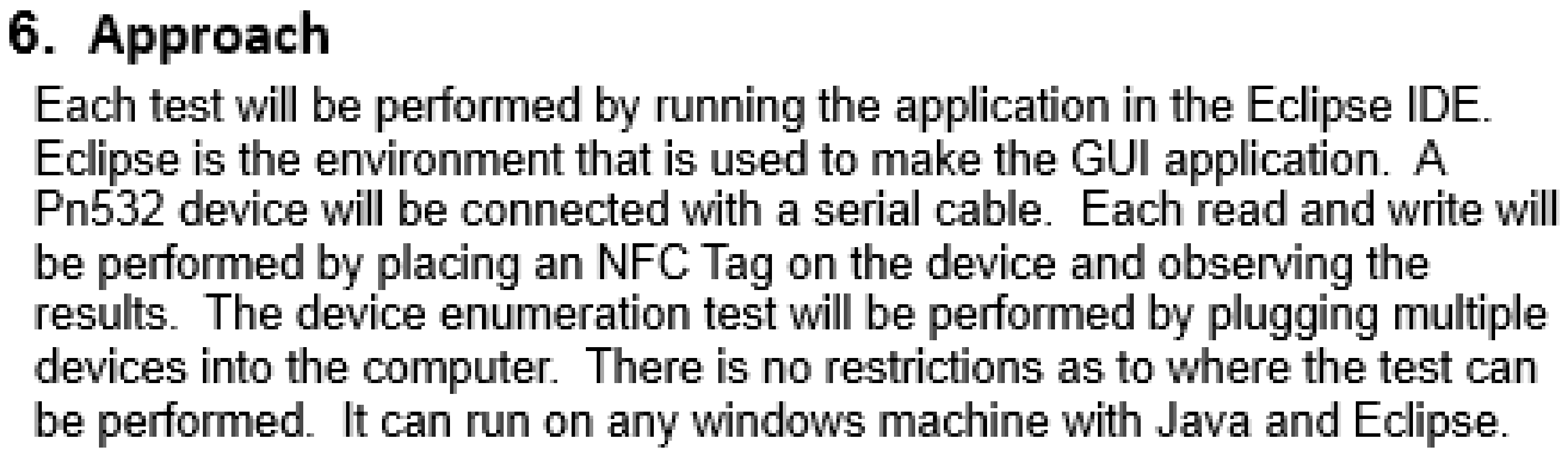
Example: Pass/fail criteria



5. Pass/Fail criteria

- Application. Identify a correctly implemented NDEF tag either created through this system or a third party following NDEF standards. Incorrect identification of a tag will constitute a failure.
- Application: Write correct tag information to the card. Must be identified correctly not only on this system, but third party systems that follow NDEF formatting.
- Visual feedback: Users presented end results manufactured by the background application code handling read, write functions. User will see what NDEF tags they present to the system contain and what they want to write onto tags.
- GUI: Icons will correctly enumerate the appropriate function.
- Hardware: System will detect (single/multiple) hardware connected to a system and appropriately allocate the application to a hardware that will utilize the software functions.
- Error: System will handle given errors either by rectifying through planned development, or will display needed information and cooperation from the user

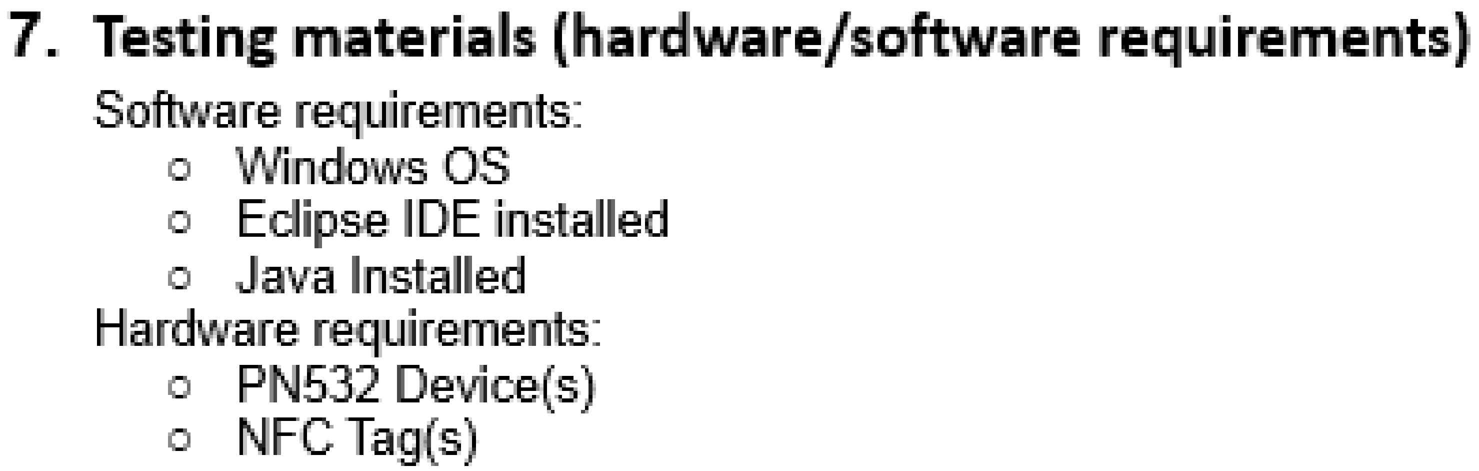
Example: Approach



6. Approach

Each test will be performed by running the application in the Eclipse IDE. Eclipse is the environment that is used to make the GUI application. A Pn532 device will be connected with a serial cable. Each read and write will be performed by placing an NFC Tag on the device and observing the results. The device enumeration test will be performed by plugging multiple devices into the computer. There is no restrictions as to where the test can be performed. It can run on any windows machine with Java and Eclipse.

Example: Testing materials



7. Testing materials (hardware/software requirements)

Software requirements:

- Windows OS
- Eclipse IDE installed
- Java Installed

Hardware requirements:

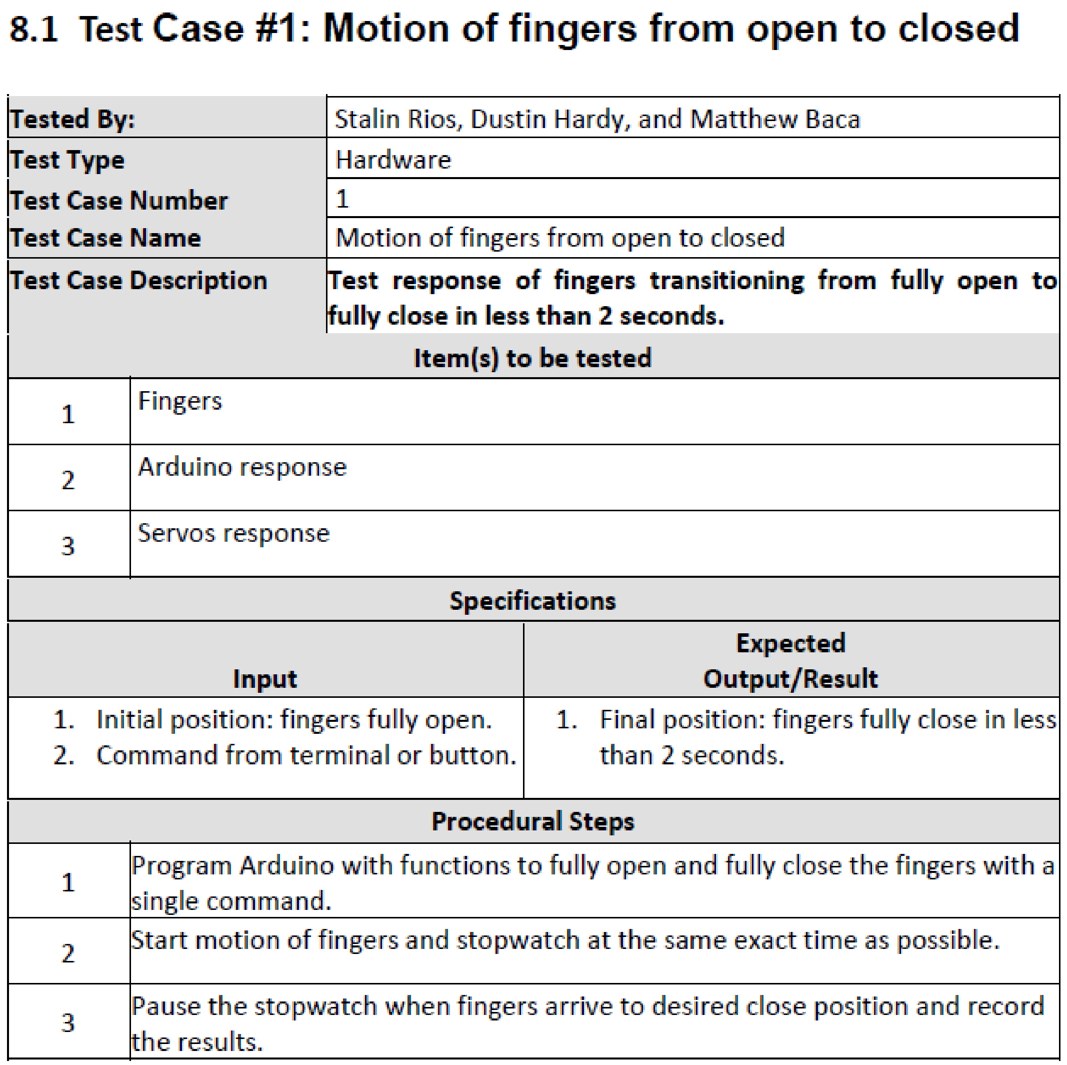
- PN532 Device(s)
- NFC Tag(s)

Example: Test Case from Spec item

Note that this spec was an old format (not tabular)

2.13.2 Response Times

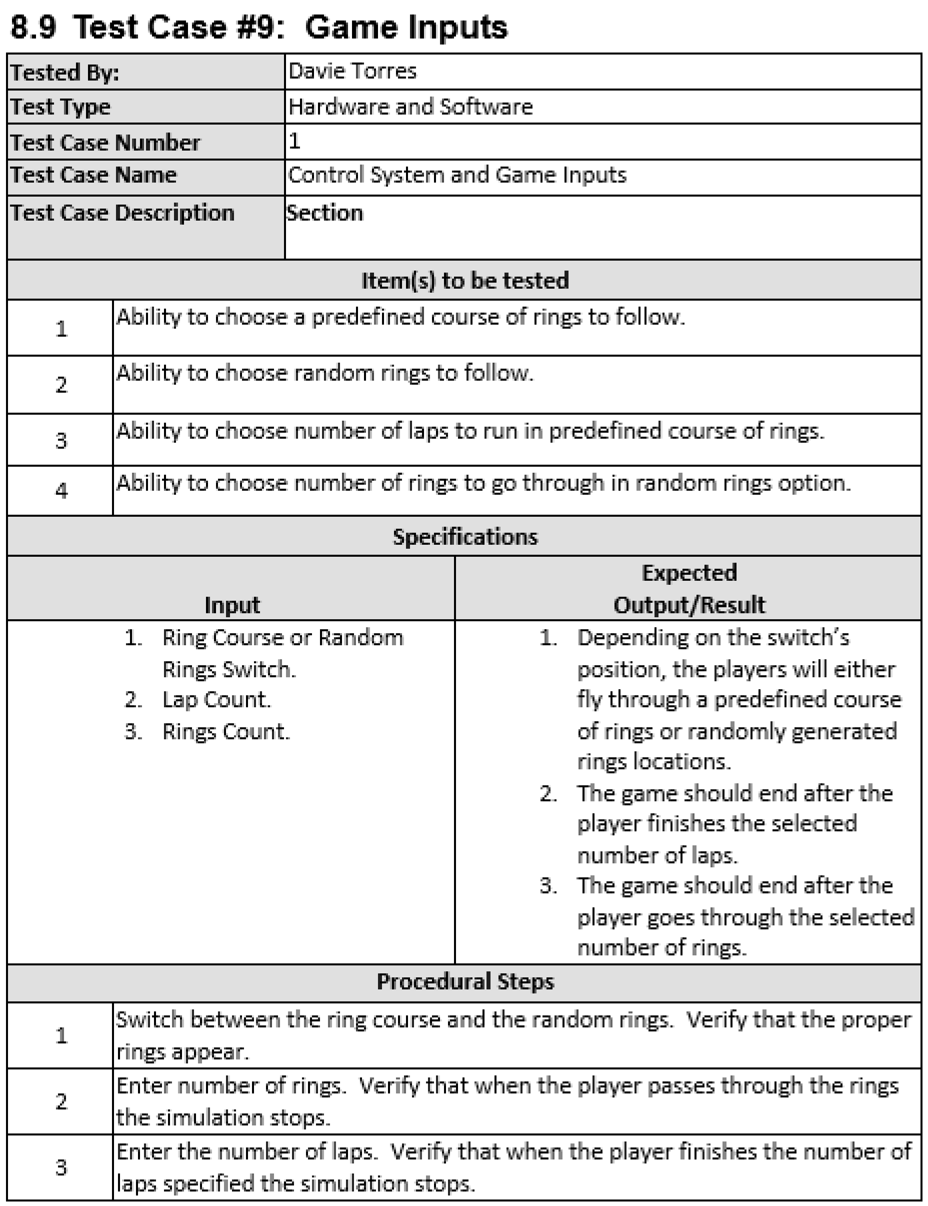
The prosthetic hand must change from fully open to fully closed position in under 2 seconds



8.1 Test Case #1: Motion of fingers from open to closed

Tested By:	Stalin Rios, Dustin Hardy, and Matthew Baca
Test Type	Hardware
Test Case Number	1
Test Case Name	Motion of fingers from open to closed
Test Case Description	**Test response of fingers transitioning from fully open to fully close in less than 2 seconds.**
Item(s) to be tested	
1	Fingers
2	Arduino response
3	Servos response
Specifications	
Input	**Expected Output/Result**
1. Initial position: fingers fully open. 2. Command from terminal or button.	1. Final position: fingers fully close in less than 2 seconds.
Procedural Steps	
1	Program Arduino with functions to fully open and fully close the fingers with a single command.
2	Start motion of fingers and stopwatch at the same exact time as possible.
3	Pause the stopwatch when fingers arrive to desired close position and record the results.

Example: Test Case from Spec item



8.9 Test Case #9: Game Inputs

Tested By:	Davie Torres
Test Type	Hardware and Software
Test Case Number	1
Test Case Name	Control System and Game Inputs
Test Case Description	**Section**
Item(s) to be tested	
1	Ability to choose a predefined course of rings to follow.
2	Ability to choose random rings to follow.
3	Ability to choose number of laps to run in predefined course of rings.
4	Ability to choose number of rings to go through in random rings option.
Specifications	
Input	**Expected Output/Result**
1. Ring Course or Random Rings Switch. 2. Lap Count. 3. Rings Count.	1. Depending on the switch's position, the players will either fly through a predefined course of rings or randomly generated rings locations. 2. The game should end after the player finishes the selected number of laps. 3. The game should end after the player goes through the selected number of rings.
Procedural Steps	
1	Switch between the ring course and the random rings. Verify that the proper rings appear.
2	Enter number of rings. Verify that when the player passes through the rings the simulation stops.
3	Enter the number of laps. Verify that when the player finishes the number of laps specified the simulation stops.

Functional Spec line items

- EACH line item in the Functional Spec must be addressed
  - Stated that it will be tested, or
  - Stated that it will not be tested – and WHY

- Use the Functional Spec as your starting point
  - Then, visualize having to do each test
  - Be as specific as possible in the Test Cases

- Good idea to run the Test Plan past your Sponsor at least once before it’s due!

Template is on TRACS

- The template on TRACS must be used
- It will simplify the process