# 4 Testing

## 4.1 Testing Strategy

This system for testing purposes has been split into multiple levels of testing as well as types of testing and a development of distinct testing models which will focus on specific functionality. The eventual goal of the project’s testing was to provide an automated system which could be run to cover all of our use case realisations, as well as provide complete code coverage, while testing for overall system integration and providing basic concurrency tests for the system.

The system has been split into 4 separate test capsules. These capsules provide the following: unit testing for the system provided by UnitTestSystem, unit tests for the cell handler and peripherals found in UnitTestCellHandler, integration tests for the system found in IntegrationTestSystem, and for the concurrency tests found in ConcurrencyTestSystem. These all communicate with the necessary system capsules that are under test through a general purpose protocol called TestProtocol. Furthermore to enable specific changes to the system that are needed to be controlled for testing purposes, a protocol to perform these changes on peripherals called PerhiperalTest was created. All these protocols and capsules will be discussed in the following.

### 4.1.1 UnitTestSystem

This capsule provides black box unit testing on the high level functions of the system. The purpose of this is to provide complete transition coverage for the SystemHandler capsule while at the same time provide 100% coverage of the unit cases which are related to SystemHandler. Examples of the unit test coverings are: arm/disarm the system, enable/disable the system, handling break-ins, phone lines etc.

### 4.1.2 UnitTestCellHandler

This capsule provides white box testing to the functionality needed to provide the functionality stated for the cell handler. This test suite provides tests for arming/disarming each alarm and sensor. After those test have been completed it tests for sensors being tripped and alarms being sounded. Finally it tests for whether or not the cells are able to pass their self tests.

### 4.1.3 IntegrationTestSystem

IntegrationTestSystem provides white box testing for the actual operation of the system. It consists of 6 testing states with their own internal test suites. They are as follows:

* EnableSystemTests: This tests that the system is able to enable itself while also ensuring it does not enable if there is a faulty password to be entered.
* DisableSystemTests: This tests that the system is able to disable itself while also ensuring it does not disable if there is a faulty password to be entered.
* ArmSystemTests: This tests that the system is able to arm itself while also arming all cells while also ensuring it does not arm if there is a faulty password to be entered.
* DisarmSystemTests: This tests that the system is able to disarm itself and its cells while also ensuring it does not disarm if there is a faulty password to be entered.
* SelfTestTests: This tests that the system is able to start and complete its system tests.
* NotifyAndHandleBreakinTests: This tests the white box tests needed to cover all possible cases of alarm use, sounding and handling.

### 4.1.4 ConcurrencyTestSystem

This suite provides the testing for concurrent events such as when an alarm gets tripped during a transition or if the system receives a break-in signal while passwords are being entered. The goal is to ensure that the system is able to transition properly even if there is a flood of un expected signals into the system

### 4.1.5 TestProtocol

TestProtocol is used to act as the communication protocol for our test cases. This protocol has all the signals used in the actual system operation. This allows all test suites to use the same protocol and allow the system to have the same signals be injected as would be expected over the course of standard operation.

### 4.1.6 TestPeripheral

This protocol is used as a tool to control perhiperal states. It can arm, disarm, trigger and fail any perhiperal or cell it is used in conjunction with other tests to get the functionality that we require while maintaining system modularity.

# 4.2 Unit Tests and Harnesses

## 4.2.1 UnitTestSystem

* Arm/Disarm System with both correct and incorrect passwords
* Enable/Disable System with both correct and incorrect passwords
* Break in with password entered correctly and on time
* Break in triggered no password entered during time out
* Break in handled, phone call placed
* Break in handled phone call failed
* Failure with password entered correctly and on time
* Failure triggered no password entered during time out
* Failure handled, phone call placed
* Failure handled phone call failed

## 4.2.2 UnitTestCellHandler

* Arm/Disarm an individual cell (loop through all cells)
* Trigger an individual cell (loop through all cells)
* Check that cells are able to respond to a timeout signals
* Tests that systems are able to respond to self test signals

# 4.3 Integration Tests

### 4.3.1 Use Case: Arm System

* Ensure post condition on the basic flow is asserted (all cells are armed) after valid password entry.
* Ensure system returns to previous state when incorrect password is entered.

### 4.3.2 Use Case: Disarm System

* Ensure post condition on the basic flow is asserted (all cells are disarmed) after valid password entry
* Ensure system returns to previous state when incorrect password is entered

### 4.3.3 Use Case: Enable System

* Assert happy path satisfies basic flow post condition.
* Assert invalid password entry satisfies "The entered password is invalid" global alternative flow post condition (the system remains disabled).

### 4.3.4 Use Case: Disable System

* Assert happy path satisfies basic flow post condition.
* Ensure that use case is not executed when system is not in ready state (system must be enabled).

### 4.3.5 Use Case: Run Self-Test

* Assert happy path satisfies basic flow post condition.
* Assert that the use case “Notify Break-In” is executed if the system is armed and a sensor fails to respond to a heartbeat.
* Assert that the use case “Notify Break-In” is executed if the system is armed and an alarm fails to respond to a heartbeat.
* Assert that failure details are displayed on the Display if the system is disarmed and a sensor fails to respond to a heartbeat.
* Assert that failure details are displayed on the Display if the system is disarmed and an alarm fails to respond to a heartbeat.

### 4.3.6 Use Case: Enter Password

* Assert happy path satisfies basic flow post condition.

### 4.3.7 Use Case: Notify Break-In

* Assert happy path satisfies basic flow post condition (timeout is started, message is displayed on Display, and the event is logged).

### 4.3.8 Use Case: Invalidate Break-In

* Assert happy path satisfies basic flow post condition.
* Assert if password is not valid, “Password is not valid” alternative flow is satisfied and the USE CASE Handle Break-in is executed

### 4.3.9 Use Case: Handle Break-In

* Assert happy path satisfies basic flow post condition.
* Assert if Phone Line has failed “Phone line has failed” specific alternative flow has been satisfied, and execution was able to resume.