**Cornell Cup**

**The “Test Early, Test Often” Testing Guide**

People, and particularly students, have a natural fear of testing. Let’s face it, if you are given the chance to have a pop quiz or be able to put off the test, 9 times out of 10 you would probably put off the test. It’s ingrained in us, because we figure if we have more time to prepare or work on the material, we feel there is a greater chance of success. Or maybe, just maybe, if things get pushed back far enough there might not be enough time in the semester and we might be able to get out of test all together.

The problem is that many students naturally transfer this aversion of testing to every kind of testing, when in reality, testing is one of the most valuable steps in the entire design process. Testing, as most students are used to, can be used to verify how well you have achieved your project’s goals. More importantly however, testing is often the ***only*** way to identify problems early on before significant time, energy and resources have been invested in continuing work that is based upon the assumption that something will work. No matter how confident you are that something should work, it is critical to perform tests at ***every*** step of the process. For if nothing else, testing not only helps you but your entire team, and even your boss, move forward more confidently and with less stress.

The reason this testing guide was created was due to the fact that the majority of problems encountered by previous student teams, that prevented them from reaching all of their original goals, could be traced back to them not testing at least some part of their project early enough. This document is split into 2 parts which can be read in either order first. The Testing Basics section offers an introduction into how to setup and use testing effectively. But if you are skeptical or have a natural aversion to testing yourself you may want to first read the Tales of Testing Aversions section first. This later section clarifies a variety of the common misconceptions and issues that students have had in order to help you recognize common pitfalls. Some of these are specific example quotes from students who had to learn these lessons the hard way, but hopefully by reading them here you can help minimize these problems in your own team.

**TALES OF TESTING AVERSION**

**The “Fear of Failure” Issue**

Part of the reason for wanting to put off a test, is a fear of failure, i.e. if the test results come back poorly, it means that you are stupid. No, that is not the case. What it means is that something was overlooked or an assumption was incorrect and you have helped identify a problem that can now be addressed. Perhaps the reason for the “failed” test results were:

* you got a bad component (and can be more wary of them in the future)
* a data sheet for a component was wrong
* a research paper did not include they made an assumption in their work that renders their method useless to you
* a start-up or operation procedure was not documented properly
* there was a typo error in your spreadsheet, circuit diagram, simulation, etc.
* there was a miscommunication in how someone’s work and your work is supposed to interface
* it was being used in a way you didn’t intend or expect

These last two reasons are important aspects that are discussed later in this document and in the separate Interface Guide. Some of these reasons may also be more your fault than others but you’re human and the way we overcome these issues is by checking ourselves and being honest with the results. Yes, you might have been able to catch some of these issues yourself if you continued to review all of your equations, and diagrams, and research papers, etc. over and over again but there quickly comes a point of diminishing return for these efforts. And if you follow the mantra of this document “Test Early, and Test Often” the consequences of these “failed” test results can be minimized. After all would you rather test a smaller component of your system, have it fail, but only have it cost a small amount of time and money, or would you rather wait until that same component fails in a larger system test where the entire system then fails as a result and perhaps will never work because it was assumed that the small component would be able to work but in truth never can. So start testing early and build a foundation that you can rely upon.

**Example 1: The “Ignorance is Bliss” Issue**

**Issue:** “I just got my code for a complex algorithm to compile, which was a good feeling. But as I hadn’t run it yet, I had no evidence that said this wouldn’t work. So it was a wonderful ignorant bliss, where I felt the accomplishment of getting the code to compile and feeling that the project completion could be very near. At this point of code development, I would often stop working for a while and even find ways to put it off because I didn’t want to have to face the potential flood of problems that may become evident once I did start testing.”

**Solution:** “Eventually I came to terms with the fact that the problems are potentially there whether I choose to recognize them right away or not. I am far better off just biting the bullet and running the tests so I can be better aware of what I am facing next and can plan accordingly.”

**Example 2: The “It’s Standard” Misstep**

**Issue:** “We were developing a robotic arm of sorts where we were using servo motors and encoders to actuate the arm. We had used servo motors and encoders in lots of applications before… just never these exact ones but they were from the same company we had used before and their other ones had worked great. It turns out that the servos we needed to use for the arm however had a slight but significant slippage issue that caused a variable miscalibration issue upon start up. Furthermore the encoders sent a slightly different signal that could be misinterpreted by our software causing the arm to “flip out”. This additionally caused some of the wire connections to loosen or become disconnected without anyone noticing and the proximity of all of the wires when the arm was pieced together made it easy to reconnect things improperly. What’s worse is that we didn’t formally document the causes and effects we were seeing and hence we wound up repeating a lot of tests and spent a lot of time guessing”

**Solution:** “As a result we developed a testing procedure to test each major component when it first came in, regardless of whether we’ve worked with it before or not. It actually became a good way to train some of the team’s younger members. Furthermore we also established a policy of testing each combination of components separately before putting the whole thing together. Tracking the results of these tests also helped a great deal with troubleshooting later problems when they did arise.”

**The “I’m Confident Enough that I Don’t Need to Test Right Away” issue:**

Getting a full system running of your own design is an incredible feeling and getting it done even faster can make us feel like a superhero. It is a highly attractive to rush for the goal but regardless of how good you are, it is inevitable (yes it is) that at some point you will make a mistake. After all, how many times is the excuse for something not working or someone getting hurt that they had done it “a million times before”.

**The “But Testing Takes Time…” Aversion, or the**

**“But if I can push off all of my testing until later on, I’ll get the testing done more efficiently” Excuse**

Part of the aversion to testing, is that it can appear to students as something that takes “additional” time. Classroom examinations have also unfortunately taught us to try to get as much work done in as little time as possible and to only go back and check if there is time, i.e. checking your work (a.k.a. testing) is something extra to be done only if time allows. In projects however, (i.e. what you’ll be doing for the rest of your professional careers) the time invested in testing will pay itself back several fold, in the time and resources saved from going too far down an unproven path only to find out much later that something does not work and you have to redo much of your work again.

**“It Works the Way it’s Supposed to” Example**

**Issue:** “We developed a new power board for our system and had even worked closely with an EE prof. using some fancy CAD software to test the board before we ordered it. When we ordered and populated the board, we tested it for the normal operating conditions and it worked great. When we used the board in our device however, there were a lot of operating conditions that exceeded the normal conditions and we were blowing board components left and right. This was especially true when other people where testing their new features and their new features didn’t work the way they expected.”

**Solution:** “In developing our next board, we spent a lot more time investigating what was the possible operating range of the main components that we knew were going to be connected and how exactly those components were going to be used. We then were able to come up with a more rigorous set of operating condition requirements that any new project feature had to be within. We also added a number of safety circuits to ensure that if someone exceeded the operating requirements, that the board itself would be safe. Thinking about the number of ways that the board could be used, even if we don’t want it to be used that way, and the number of ways that it could fail has since become a standard part of design procedures.”

**Minimizing the “Well we ran into a couple things trying this out but it’s basically done” Issue:**

Following the “It’s not done until it’s tested and documented” may earn you some moans from your team but once they have that first experience where a “failed” testing made them rethink their overall approach to a problem, they are more likely to come on board and be eager to “Test Early and Test Often” ideology.

**INTRODUCTION TO TESTING BASICS**

**What Should We Be Testing? And What Exactly Defines a Test?**

Everything. First off, if you don’t test something, it shouldn’t be in your final project. And if you can’t test something, you shouldn’t build it in the first place. All tests are made up of at least 4 basic components

1. **Testing Procedure**: What are the steps that need to be carried out in order to complete the test
2. **Testing Facilities**: A list of what are the equipment, the laboratory setting, and the other resources you will need to carry out the testing procedure.
3. **Entry Condition**: What is the state of the project required in order for the test to begin
4. **Exit Condition**: What criteria must be met in order for the test to be considered a success or for the test to earn a “passing” score.

**Testing Procedure:**

The testing procedure, (a.k.a. the test method) is an ordered list of steps that the tester should carry out during the test. In order to check that the written test procedure is clear and can be conducted in a consistent manner repeatedly, it is often required that the test procedure be reviewed by someone other than its author, and/or that the first time the test is carried out that both the author and a non-author be present and update the procedure so it can be carried out later on without the author being present.

Documenting is always a necessary and valuable step for ensuring that your work will continue to be utilized after you have left a project, and documenting test procedures is no exception. As described in the re-testing section later on, documenting test procedures is also important for being able to repeat them consistently throughout the project. Just by writing out the procedures it is not uncommon to also identify new potential issues/concerns/needs that have yet to be addressed.

**Testing Facilities:**

As said above this is a list of what are the equipment, the laboratory setting, and the other resources you will need to carry out the testing procedure. Often this serves as a good check list to use to make sure that all of the proper equipment, lab space, etc is reserved or will be available at the time you want to conduct the test.

**Entry Condition:**

Different tests may only be applicable during different phases of a project. Hence the entry condition specifies what must already be accomplished in order to begin this tests, i.e. what tasks must have already been completed, what parts must have been ordered, made, etc., what clearances must have been obtained (for example human testing clearance), or anything else that is needed.

**Exit Condition:**

The exit condition is essentially a measure of how good the results of the test are and whether they can be considered good enough. Many times the exit condition may be the way to interpret the results of the test. Often this is related to a project performance metric(s) or whether a certain requirement has been sufficiently met. Some of the best designed tests also incorporate a way to tell whether the test was conducted correctly, perhaps through the use of a control.

**Interface Testing**

Every major or critical component should be tested to validate that it operates as it is reported to. This often only takes a few minutes to do but it can save hours. More so, the most common place for systems to fail is at the interface between 2 or more subsystems. This is easy to imagine as one person may test their subsystem well and another person may test their subsystem well but when the two are combined, miscommunicated specifications can lead to integration issues that may result in significant rework for both subsystems.

One way to help deal with interfaces, is to specify interface requirements and interface testing exit conditions (a.k.a. success criteria) in the early stages of the subsystems development. There is a saying “what get’s measured, get’s done” and so if both subsystem developers recognize that their own performance will be measured based upon how well it performs in the interfacing testing, the interface testing is often given more of the priority it deserves.

**What Makes for a Good Test and What Should a Good Test “Tell Me”?**

For starters, many good tests are fairly specific or narrow in scope. In fact, it is helpful and often necessary to make several smaller tests where you are only testing one thing at a time in order to be sure that the test results are clear and easy to interpret. Smaller tests also tend to be easier to conduct with

A good test should help you to make a decision on what direction to take next. This can be a binary “Go / No Go” or this can be detailed performance results that indicate the current strength and weakness of your project at meeting your overall challenge’s needs.

The only real bad test results are those that are inconclusive. So the key thing is to design your tests to be as definitive as possible. A way to do this is to make a list of what are all of the possible outcomes that could result from the test you are considering. For each outcome, determine what decision(s) or courses of action that you would take as a result. If it is not clear what course of action you should take, modify the test to or create additional follow-on tests that will make your course of action clear.

**What Should I Test First?**

Arrange your timeline so that you perform early on any tests that could result in you taking a drastically different direction from what your current plan is. Many good timelines are actually test driven, i.e. how can I order my work so that I can test my most critical or uncertain components as early as possible while dedicating as few resources as possible?

Now it is true that some critical tests should be left until later because supporting work must be completed first. In these cases, the supporting work should often be given precedence. However, the amount of supporting work itself can be quite substantial, and in some cases you may only be able to find out whether the supporting work is worthwhile once the more critical tests are complete. This can be very risky for a project’s overall success. In order to help mitigate this risk, it is common to perform additional “proof of concept” and/or “mock-up” testing. Proof of concept testing is where instead of building the full system, you builder a smaller system that can test the general feasibility of critical features. Mock-up testing is often when you create something that is representational of a more complex system to gain additional insight about critical features.

**Simplified Examples of Proof of Concept and Mock-Up Tests.**

As a simplified example, let’s say your team is trying to create a solar power airplane and the top performance criteria is the overall flight time. The critical question you have is whether the shape of the plane will provide sufficient aerodynamics allowing enough solar panel space and secure attachment on the plane body so as to provide enough power. Building the entire plane body would be time consuming and expensive and if you’re wrong about the shape it may cause you to have to redo a lot of work associated with it. So some possible early steps, proof of concept, and mock-up tests might be:

* Determining the solar panel / surface area power generation and then purchase 1 of each of your top solar panel choices. Also purchase a sample of the possible plane body materials. Verify their power generation and attachment methods on the lab bench as an initial proof of concept that the panels can be attached in a secure and non-damaging to the plane material manner.
* Creating a CAD version of the plane, i.e. a virtual mock-up, which could be tested for aerodynamics via simulation.
* Creating a physical model of the plane using cheaper and easier to work with materials as a mock-up of the plane. This could then be tested in the wind tunnel to gain further insight into the aerodynamics not captured by the simulation’s equations.
* Attach the best solar panel choices to the physical mock-up plane to see how they weather in the wind tunnel.
* Finding an existing model air plane of roughly similar size to the one you’re thinking of that you could attach solar panels. Then measure the power generation during a test flight to determine “real world” power generation and check the attachment at the conclusion of the flight.

Doing these intermediate tests can increase the length of your overall timeline but your project will have a much better guarantee of success in the long run. Each of these testing steps could help inform you more about the problem and help you make project planning corrections earlier on.

Also notice in this example the solar panel tests help inform the plane shape design and in turn the plane design tests help inform the solar panel. Both the solar panel aspect and the plane shape aspect could be designed separately, where one is asked to completely conform to the needs of the other. However if you can create tests to also inform interfaces requirements early on, this can allow you to make trade-offs between both more effectively for an overall better design.

**Testing New Features or Feature Changes**

Whenever a new feature is added or an existing feature is changed it is important to redo any features that could be affected by this addition/change. In Computer Science this idea is often referred to as Regression Testing and is the cornerstone of many software reliability approaches. This may seem excessive but how many times has it happened where you’re testing a new feature and all of a sudden something else breaks. Or how about the feeling when you’re troubleshooting/debugging and you suddenly get results that say the problem lies in something that you already tested long ago. “That can’t be the problem!” you proclaim, “I tested that before and it was fine!” The “what the heck is going on here” feeling that ensues can cause you to start to second guess all kinds of assumptions and results haphazardly as you try to piece together the “truth”. So even though it may seem tedious at times to retest old functionality, it can be critical for long term reliability.

The key to being able to reliably conduct any kind of regression testing however is to make sure that the test procedure, test facilities and entry and exit conditions are well documented so that they can repeatedly conducted consistently.

**Non-Computer Science Example of this Regression Testing**

Regression testing is easy to envision in computer science as it’s easy to imagine that if you added a new function to a class that modifies some of that class’s variables, that you should probably re-test any other features that make use of those variables. Regression testing in other fields however is not quite as clear, so as a mechatronic example, consider the development of trajectory controls an autonomous vehicle. In developing the vehicle’s trajectory controls, the following series of tests were conducted with the hope of improving the vehicle’s performance at each step, i.e. how close the vehicle went to where you told it to go. (Although you do not need to have a controls background to understand the value of this example, just notice that each test is a progressively more complicated and involved combination of techniques that builds upon past tests, where the first two levels are largely about the dynamics of the vehicle and the later ones focus on control techniques).

1. Open loop testing in just 1 dimension
2. Open loop testing in 2 dimensions
3. Closed loop testing using a simple proportional control
4. Closed loop testing using an initialization “turbo” feature
5. Closed loop testing using a proportional control and an initialization “turbo” feature
6. Closed loop testing using a proportional control with lag correction
7. Closed loop testing using a proportional control and integral control
8. Closed loop testing using a proportional control and integral control with lag correction
9. Closed loop testing using a proportional control and integral control with an initialization “turbo” feature
10. Closed loop testing using a proportional control and integral control with lag correction and an initialization “turbo” feature

After completing some testing at test level 9, it was found that the friction within the wheel shafts was causing the vehicle to drift slightly to the left. The friction issue was resolved mechanically but instead of continuing onto 10 or resuming testing at level 9, or even returning to the first controls techniques tests of level 3, the testing began back at level 1.

Returning to the simple proportional control schemes of test level 3 would have allowed the control techniques to be at least recalibrated at each step and may have even appeared to work, but this would not have been sufficient as the wheel shaft mechanical solution may have modified the dynamics of the vehicle. Therefore the results of test level 1 and 2 were no longer accurate, and additionally the intuition developed by the team in performing test level 1 and 2 was also inaccurate. Had the team not restarted at level 1, it is likely that as the system became more complicated at the higher test levels that they could more easily misjudge

**Organization of Test Records**

Tests are often numbered and organized into sets according to the features and/or requirements they test. This way when you want to perform regression testing it is easy to identify which sets of tests should be run. These sets of tests are often referred to as a test plan.

A sample record of test plans may look something like the following excerpt below. Please note that the titles of all columns include the word “Summary”. That is because it is obviously hard to fit the entire test procedure within a single cell of a table, and hence there is also often a column that includes references to a separate document(s) that contain more of the details.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test #** | **Supporting Docs** | **Test Procedure Summary** | **Test Facilities Summary** | **Entry Condition Summary** | **Exit Condition Summary** |
| **T.1-50** | **…** | … | … | … | … |
| **Basic 2 Board Communications Feature** |
| **T.51**  | **B2B\_CommTest51.docx** | Message Completion Test: 2 Boards send messages back and forth checking for any dropped, corrupted, or incomplete messages | 2 Main boards, 2 WiFi cards, 2 external monitors set ups  | Communications software installed and passing all single board tests | 99.99% complete message send & receive rate, |
| **T.52-55** | **…** | … | … | … | … |
| **T.56**  | **B2B\_CommTest56.docx** | Sensor communication test: A sensor connected to 1 board controls the actions of a servo motor on another board | 2 Main boards, 2 WiFi cards, 2 external monitors set ups, 1 Servo motor, 1 IR sensor | Communications software installed and passing all single board tests. Servo motor controls software installed and tests passed. Basic IR Sensor I/O passed tests.  | Messages delivered within acceptable latency bounds, Servo motor performs all actions within acceptable limits |
| **Base Station to Board Communications Feature** |
| **T.57** | **…** | … | … | … | … |

A good test plan should cover all of the possible ways a feature may be used or affected, not just the ways that are intended, but all the ways that could occur in the operation (or mis-operation) of the system.

**Testing as a Project Leadership/Management Tool**

Many of you are likely to move into careers that involve project leadership/management. Developing good testing habits now can be remarkably beneficial. From a project leadership perspective, testing also helps you (and your own boss) have a more confident understanding of the progress, accomplishments, and outstanding issues that remain. Tests results which are part of a test’s exit conditions can make for more concrete evidence for how well that a task has been performed. As stated before can also

Testing can also be an excellent way to get members of the team who were not directly involved with the features being tested more exposure to and even training on these features. The testing procedures can provide insight as to the context and operation of the tested feature, and the exit conditions can provide insight into the important performance criteria or other associated metrics involved with evaluating the feature’s value. As a result more of your team will have a better grasp of all of the trade-offs that may be occurring across the entire system design.

Involving new members in testing is also a good way to introduce them to the project and once they have gained some initial insight into the project thru testing, they may be able to offer more valuable new perspectives as well. As writing up test procedures may at times seem tedious to more experienced members, reminding experienced team members that these documents can help with training new members can sometimes help them understand that it’s more worthwhile. In some cases it may even be possible to pair less experienced and more experienced members together on testing and having the less experienced member be the scribe and the more experienced member be the reviewer.