In this class we discussed testing models, testing principles, and test plans. The essential thing to take away from this information is not simply that testing is really important (duh), but also something that requires diligent planning. Stop thinking about testing as something you do at the end, and start thinking about testing as something that is planned for. This switches the model to be validation, as described by Davis:

In this model, you can clearly see that each stage of the development lifecycle builds in the development of the exit criteria for that stage. In other words, as you are generating user requirements you are also building the acceptance test plan. Then, at the very end of the project you execute the plan. This occurs for each stage of the lifecycle. If it sounds like a lot of work, it is! However, consider the costs associated with not testing…

One approach to testing is based on rigorous review, which is often claimed to result in enormous savings (up to 1000% ROI). I think these types of savings are unrealistic for well managed professional projects – like yours, but it follows a well known principle of “Garbage in – Garbage out”.

When it comes to developing test plans, you will need to consider the following issues:

- How much testing?
- How much is enough testing?
- How much is too much testing?
- What level of detail is necessary?
- What is risk of failure?
- What is cost if failure occurs?
Then, develop a concrete set of steps to verify that there are no errors:

1) Select what is to be measured
2) Decide how to test
3) Develop test cases
4) Determine expected result of test
5) Test
6) Compare results to expected results

We can measure errors, and a common approach is to track the rate of error reporting, and the pace at which errors are corrected. If you are falling behind (more errors reported than you are fixing) it is considered a growing stage. While in a growing stage, you can’t predict completion with any certainty. Conversely, when errors are coming in slower than you are fixing them you are in a shrinking stage. In a shrinking stage you can start contemplating completion. Of course, at any given time you might encounter a system-killer type of error.

It is interesting to note that often, the distribution of errors and effort follows a Pareto distribution, or the “80/20 rule”:

- 20% of modules consume 80% of resources
- 20% of modules contribute 80% of errors
- 20% of errors consume 80% of repair costs
- ...

Most errors are simple errors that don’t cost much to fix.

A set of principles guide our testing strategy:

1) Testing is the process of discovering errors
   How do we measure success?
   Finding errors?
   Finding no errors?
2) It is impossible to completely test a non-trivial system
   There are acceptable and unacceptable risks
3) Testing is hard work
   Knowledge of programs, testing, and creativity
4) Testing prevents errors
5) Testing by more than one tester is best

There are four common practices:

- Inspection
- Walk-through
- White box
- Black box

Copyright 2010 – Dennis P. Groth
Inspection:

- Insufficient for just finding defects
- Goal:
  - Gather data to
  - Determine common causes of defects
  - Improve processes
  - Reduce costs

Walk through:

- Useful for design and requirements
- Time consuming
- Involves different perspectives

Black box:

- You don’t get to see how the module works
- Can only evaluate based on assumptions

White box:

- You get to see how the module works
- Can violate assumptions

We did a couple of different practices in the class. One was to just come up with all the different test cases for a module that tests triangles:

- Consider a program
  - Input: 3 integer values, with each value representing the lengths of the sides of a triangle
  - Output: message that states whether the triangle is scalene, isosceles, or equilateral, or an appropriate error message

Test cases in the form:

<table>
<thead>
<tr>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equilateral</td>
<td>1,1,1</td>
<td>Equilateral</td>
</tr>
</tbody>
</table>