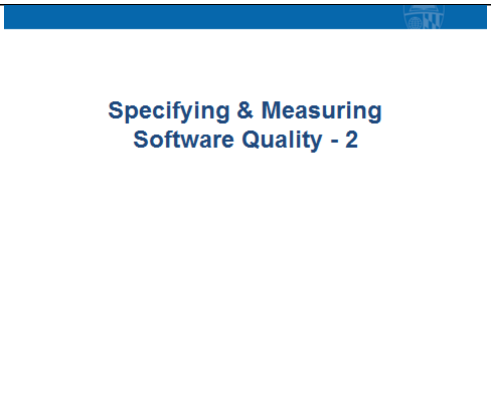
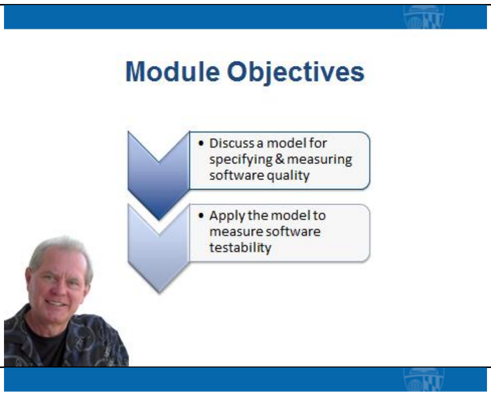
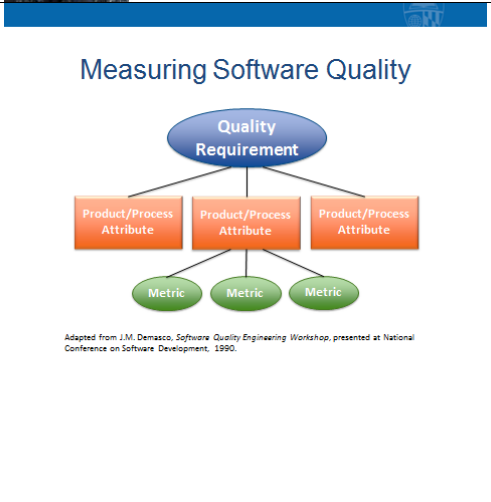
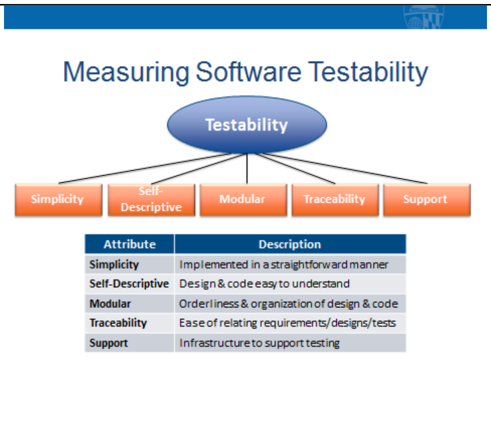
<table>
<tr>
<td>1</td>
<td>
Specifying &amp; Measuring
Software Quality - 2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>
Module Objectives
• Discuss a model for specifying &amp; measuring software quality
• Apply the model to measure software testability</td>
<td>In this lecture, I’ll discuss a model for specifying and measuring software quality, and I’ll apply that model to illustrate how one can measure software testability.</td>
</tr>
<tr>
<td>3</td>
<td>
Measuring Software Quality

Quality Requirement
Product/Process Attribute
Product/Process Attribute
Product/Process Attribute
Metric
Metric
Metric

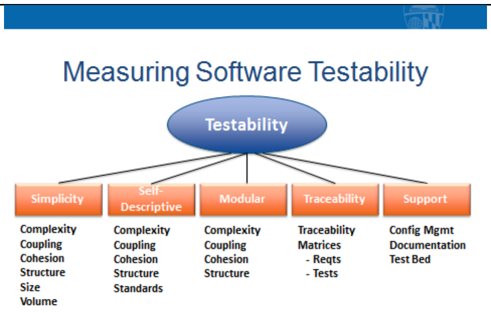
Adapted from J.M. Demasco, <i>Software Quality Engineering Workshop</i>, presented at National Conference on Software Development, 1990.</td>
<td>Here’s a model that can be used to develop metrics to indicate that a quality requirement is built-into a software product.

At the top is the quality requirement that needs to be satisfied…such as testability. The next level defines product or process attributes that lead to a product having that quality characteristic. Then, for each of the defined product and process attributes, one or more metrics is defined.

I’ll illustrate how this is done using a specific quality requirement.</td>
</tr>
<tr>
<td>4</td>
<td>
Measuring Software Testability
<table>
<tr><th>Attribute</th><th>Description</th></tr>
<tr><td>Simplicity</td><td>Implemented in a straightforward manner</td></tr>
<tr><td>Self-Descriptive</td><td>Design &amp; code easy to understand</td></tr>
<tr><td>Modular</td><td>Orderliness &amp; organization of design &amp; code</td></tr>
<tr><td>Traceability</td><td>Ease of relating requirements/designs/tests</td></tr>
<tr><td>Support</td><td>Infrastructure to support testing</td></tr>
</table></td>
<td>Let’s take testability as an example, and see how we can develop some metrics that indicate whether the software product is being built so that it is easy to test.

What are some attributes of a software product or a development process that might make testing easier? Let me start by listing three product attributes that help to make testing easier: simplicity, self-descriptiveness, and modularity. I’ll define them more precisely in a minute, so for now use your intuitive understanding of what you think they might mean.</td>
</tr>
</table>

		Next, I’ll define two more attributes that aren’t direct attributes of the software product itself, but have to do with related work products and environment. I’ll add traceability and support. Now, let’s formally define what these attributes mean. Simplicity is a characteristic of a products design and code. Are they implemented in a straightforward manner. Self-descriptiveness also applies to the design and code. Are they easy to understand by visual inspection. Modular applies to the design and code as well. Is it easy to see which components of the product perform certain functions. These three attributes help make product testing easier…particularly for developer-level testing, because they help developers understand which components need to be exercised to test specific functions and they make it easier to instrument test cases. They also make it easier to determine what needs to be fixed if tests indicate there are problems. Traceability has to do with the ease of relating test cases to the requirements that are being tested and to the specific product components that implement those requirements. Traceability helps testers understand which tests may need to change if a requirement changes, and which tests need to be re-run. Support has to do with the infrastructure that is in place to support the testing effort. The next thing we need to do is to identify some metrics that can be used as indicators of these attributes.

5	 Measuring Software Testability Testability Simplicity Complexity Coupling Cohesion Structure Size Volume Self-Descriptive Complexity Coupling Cohesion Structure Standards Modular Complexity Coupling Cohesion Structure Traceability Traceability Matrices - Reqts - Tests Support Config Mgmt Documentation Test Bed	I'm going to just name the metrics right now, and discuss most of them in more detail later in this lecture. Let's start with simplicity. There are a number of metrics that are associated with simplicity…control flow complexity, coupling, cohesion, code structure, code size, and something called information volume. Most of these metrics can be applied to product design, and all of them can be applied to code. Most of the same metrics apply for measuring self-descriptiveness. Additionally, standards such as documentation and coding standards can help to make product components easier to understand. This would include things like source code documentation, statement of pre-conditions and post-conditions, and so forth…things that make code easier to read and understand. Modularity metrics are also pretty much the same as simplicity metrics, and can be applied to both design and code. The more modular the product is the easier it is to determine which components implement which requirements and which may need to be retested when test failures indicate problems. Traceability can be measured by using traceability matrices. One type of traceability matrix traces each requirement to the design and code components that implement it. This is very useful for developers as it helps them more easily instrument tests and determine what code may need to be changed if tests indicate problems. Another type of traceability matrix traces each requirement to the test cases used to verify that requirement. This can be very helpful to both developmental testers and independent testers in helping to determine which tests may need to be changed if requirements are changed and which tests may need to be re-run. In terms of support, an effective configuration management process can help gain access to the correct versions of components that need to be tested. Documents such as test plans and test reports also help make testing easier, as well as having what is commonly called a test bed…a repository of tests that can easily be re-run.

		Now…in practice…we wouldn't necessarily use all of these metrics…just the ones that have the most value-add for our particular project. What I wanted to demonstrate here is how metrics can be applied to get early insight as to whether quality characteristics are being built into the product. This example also illustrates different types of metrics: complexity, size, structure and volume are examples of what I like to call computational metrics…they are calculated using simple counts or formulas. Coupling and cohesion are examples of what I call classification metrics…we classify design and code components according to whether they implement good coupling and cohesion characteristics or not so good. Traceability matrices are yet another form of metric that don't yield numeric values, but are used to help find problems and make testing easier. They can also be used to construct quantitative quality metrics…for example, to calculate the percentage of requirements that can be traced to design components or test cases.