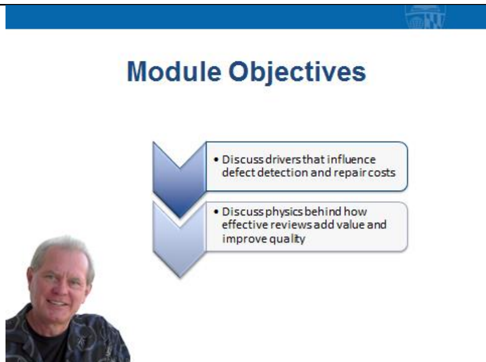
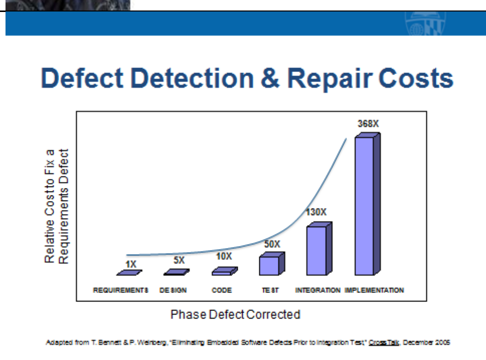
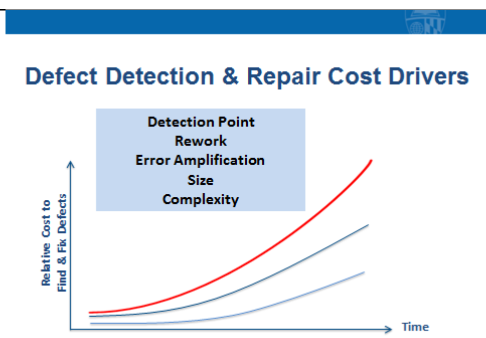
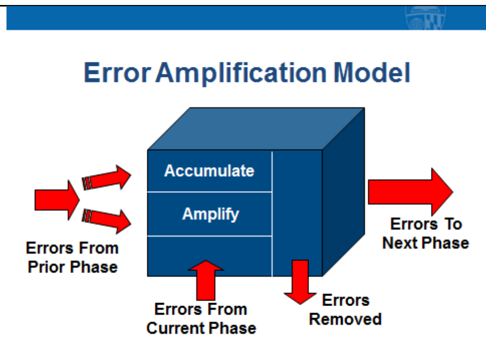
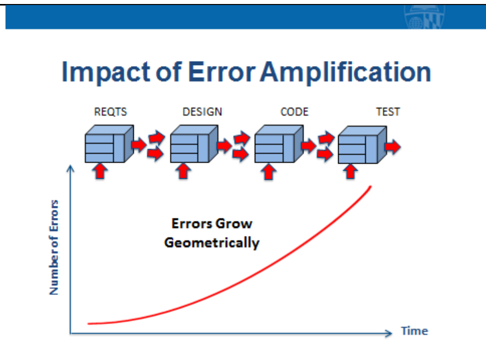
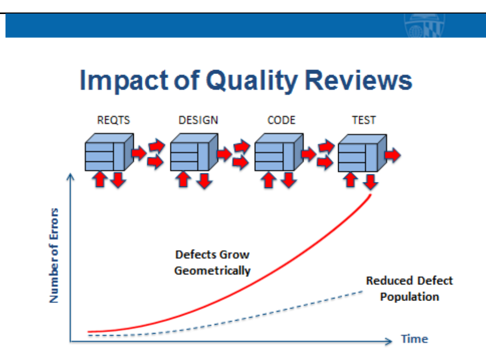
<table>
<tr><td>1</td><td>
</td><td></td></tr>
<tr><td>2</td><td>
</td><td>In this lecture I’ll discuss some additional drivers that influence defect detection and repair costs, and discuss more about the “physics” behind how effective quality reviews can add value and improve quality.</td></tr>
<tr><td>3</td><td>
</td><td>I’ve already talked about rework being a major driver behind the exponential-like defect detection and repair cost escalation, and I explained how the detection point influences the amount of rework. There are some other drivers as well.</td></tr>
<tr><td>4</td><td>
</td><td>Others worth talking about are a phenomenon called error amplification and project size and complexity.

Error amplification is a phenomenon that exists on every single software project. The one sentence description of this phenomenon is that defects introduced in any project phase can cause additional defects in later project phases…producing a compound growth effect. I’ll talk more about this shortly.

Additional drivers are size and complexity. I’ve lumped them together here because they have a similar effect on the cost function. By size I mean the size of the software product…measured, say, in lines of code. By size I also mean the number of people</td></tr>
</table>

		working on the project. Complexity refers to the intellectual and technical complexity of the software product. For example, real-time embedded products typically have a higher complexity than batch-oriented products. All of these drivers influence the relative cost multipliers either increasing or decreasing them. For example, early defect detection reduces rework and the combined result is to decrease the multipliers, whereas higher complexity and more people working on a project increase the multipliers.
5		I want to say a little more about the error amplification phenomenon, because it exists on every software project. Here's a model that describes it. This model was actually formulated at IBM back in the early 1980s…and it is still valid today. Here's how it works. We have a box like this for every phase in our project life cycle. For the sake of discussion, let's assume we are in the coding phase of a project. Some coding errors may be introduced during this phase…but that's only one source of error. Another source is the errors that were made in prior life cycle phases, and that weren't detected and removed. In our example, these would be requirements and design errors. Now, some of this prior phase errors will simply accumulate…they won't cause any additional coding errors to be introduced. IBM found that about 50% of the errors that get through the requirements and design phases will accumulate…but the other 50% will be amplified. That means they will cause additional errors to be introduced in the coding phase that otherwise would not have been introduced. An example of this would be an ambiguous requirement. A programmer may interpret this requirement incorrectly. So…if we added up the errors in the three compartments that would be the total number of errors in our project during the current phase. Depending upon the error detection and removal

		activities that we implement, some of these errors will be removed, and the remainder will pass on and be inherited by the next phase in the project, where the amplification phenomenon repeats itself.
6		Let's take a look at the impact of error amplification. Here, I'm just taking into account a requirements, design, code, and test phase. I mentioned earlier that IBM found that on average about half the errors that got through the requirements and design phases were amplified. They also found that the amplification multiplier in the design phase was 1.5 and in the coding phase was 3. To see how that works let's assume that 10 errors passed through the design phase undetected. Half of them would simply accumulate, and half would be amplified by a factor of 3. So…5 errors are amplified, yielding a total of 15 amplified errors…5 original ones plus 10 additional. By the way, I've actually applied this model in several client situations. One of my clients, a division of NASA, found that 80 percent of errors passing through the requirements and design phase were amplified, and the amplification multiplier in the coding phase was near 10. Because of this phenomena, the number of errors at any given life cycle phase is a function of the number of errors at the prior phase…and the population of errors tends toward a geometric growth rate. This further impacts the amount of rework that may need to be done, particularly if most or all of the defect detection activity is concentrated in the testing phases of the project…because the defect population is growing and compounding in the earlier phases.
7		Effective software quality reviews can have a significant impact on the amplification phenomenon. By performing quality reviews during the requirements phase, the design phase, and the coding phase, defects can be detected and removed earlier…resulting in a significant dampening of the error compounding. This is another example of what I referred to as the

		“physics” of software quality reviews.