Webinar_ST

Table of Contents

A Taxonomy of Testing Types 4

Topics 6

Relevant Testing Challenges 7

Goals of the Presentation 8

Polling Question 1 9

What is Testing? 11

What is Testing? 12

Presentation Scope 13

Types of Testing 14

Polling Question 2 15

Types of Testing 17

Types of Testing 19

What: by Object Under Test (OUT) 20

What: by Object Under Test (OUT) – Model Testing 21

What: by Object Under Test (OUT) – Hardware Testing 22

What: by Object Under Test (OUT) – Software Testing 23

What: by Object Under Test (OUT) – System Testing 24

What: by Object Under Test (OUT) – Data Center Testing 25

What: by Object Under Test (OUT) – Tool / Environment Testing 26

What: by Domain 27

Types of Testing 28

When: by Temporal Order 29
When: by Lifecycle 31
When: by Phase 32
When: by BIT Execution Time 34
Types of Testing 35
Why: by Driver 36
Why: by Reason 37
Types of Testing 39
Who: by Collaboration 40
Who: by Organization 41
Polling Question 3 42
Who: by Role 45
Types of Testing 47
Where: by Organizational Location 48
Where: by Physical Location 49
Types of Testing 50
How: by Level of Automation 51
How: by Level of Scripting 53
How: by Technique 54
How: by Technique - Blackbox Testing 55
How: by Technique - Graybox Testing 56
How: by Technique - Whitebox Testing 58
How: by Technique - Experience- Based Testing 59
How: by Technique - Random Testing 60

Types of Testing 62

How Well: by Quality Characteristic 63

How Well: by Quality – Capacity Testing 65

How Well: by Quality – Reliability Testing 66

How Well: by Quality – Robustness Testing 67

How Well: by Quality – Security Testing 69

How Well: by Quality – Usability Testing 70

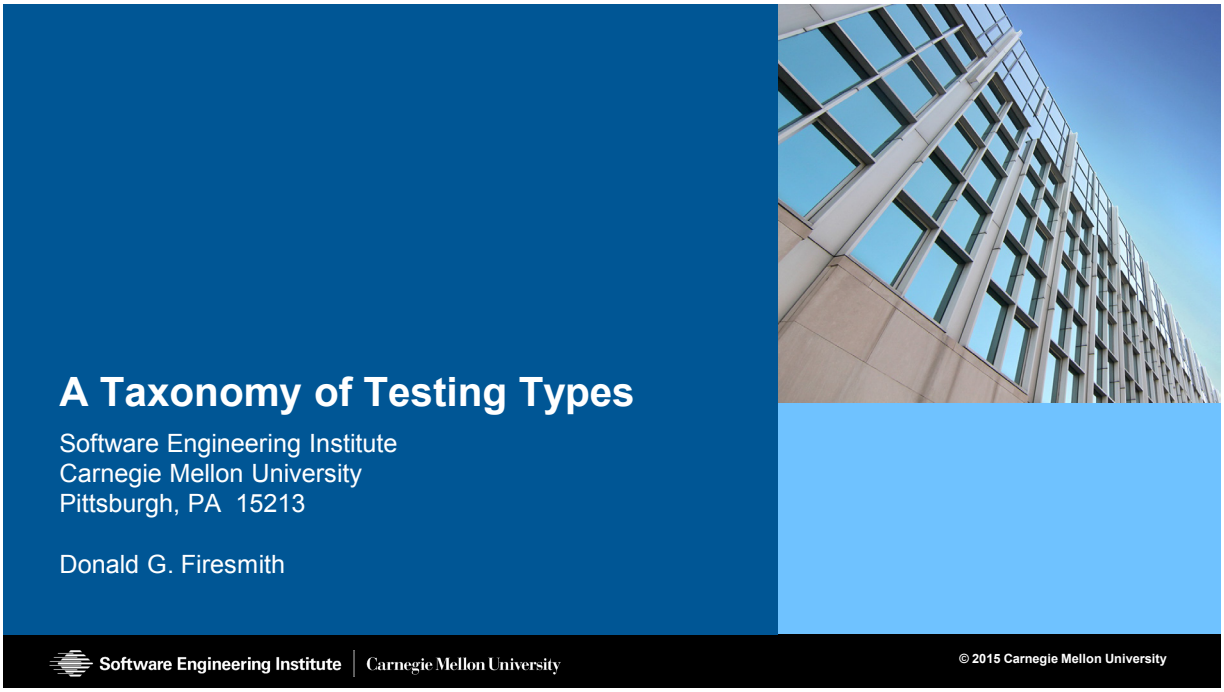
Conclusion 71

Q&A 73

Carnegie Mellon University 77

Copyright 2015 Carnegie Mellon University 77

A Taxonomy of Testing Types



**003 Shane McGraw: And hello from the campus of Carnegie Mellon University in Pittsburgh, Pennsylvania. We welcome you to the Software Engineering Institute's webinar series. Our presentation today is a taxonomy of testing types. Depending on your location, we wish you a good morning, a good afternoon, or a good evening.

My name is Shane McGraw. I will be your moderator for today's presentation. And I'd like to thank you for attending. We want to make today as interactive as possible, so we will address questions throughout the presentation and again at the end of the presentation. And you can submit your questions at any time during the presentation through the questions tab on your control panel.

We will also ask a few polling questions throughout the presentation. And they will appear as a pop up window on your screen. The first polling question we'd like to ask is how did you hear about today's event.

Another three tabs I'd like to point out are the files, Twitter, and survey tabs. The file tab has a PDF copy of the presentation slides there now, along with other testing relating work from the Software Engineering Institute. The survey, we request that you fill out at the end of the presentation as your feedback is always greatly appreciated. For those of you using Twitter, be sure to follow @SEI_SSD, as in the software solutions division, and use the hashtag SEIwebinar.

Now, I'd like to introduce our presenter for today. Donald Firesmith is a principle engineer at the Software Engineering Institute where he helps the U.S. government acquire large, complex, software-reliant systems. With thirty-seven years of experience working as a software and systems engineer, he's widely recognized as an expert in requirements engineering, system and software architecture, object-oriented development, testing, and process engineering. He has written seven technical books including most recently, "Common Systems and Software Testing Pitfalls". He has present numerous papers and given numerous tutorials at international software and systems engineering

conferences and has been published widely in technical journals.

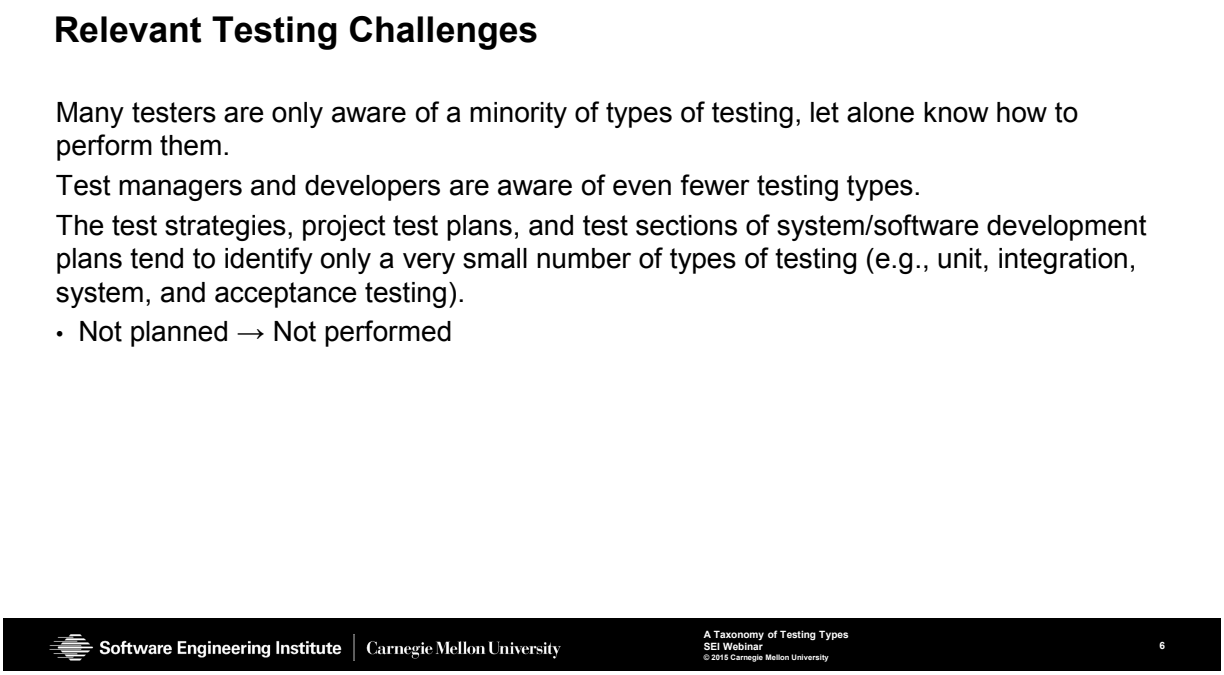
And now, I'd like to turn it over to Donald Firesmith. Don, all yours, welcome.

Topics

**004 Donald Firesmith: Thank you. Okay, today I'm going to be covering several different topics on the idea of a taxonomy of testing types. There are some challenges that testers face when it comes to deciding on what kind of testing to perform. We're going to take a look at what I envision as the goals of this presentation. I'm going to covering over two hundred different types of testing. So, I'm not expecting everyone to know all two hundred by the time I'm done. But at least, you'll get a good idea of what's out there.

I'll talk a little bit about testing because I'm not going to cover the other types of verification/validation techniques besides testing. I'll take a look at, then, the different testing types. That's where the meat of the presentation is. And then we'll go to a brief conclusion.

Relevant Testing Challenges

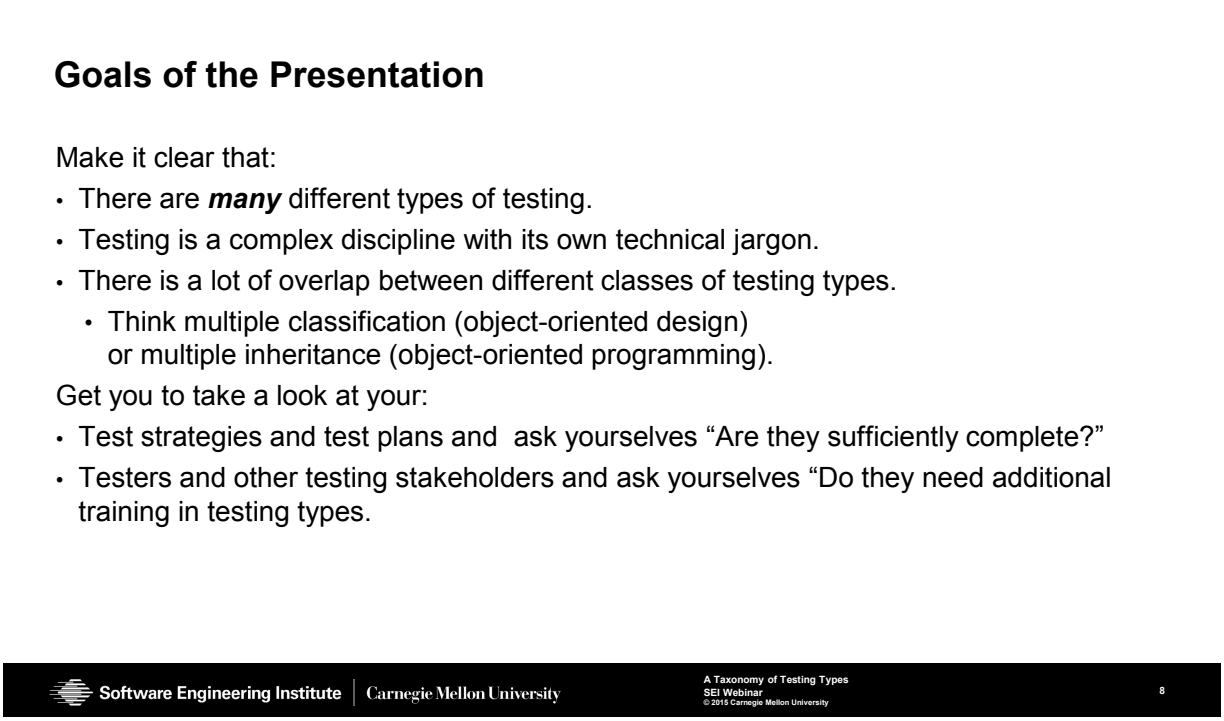


**006 Okay, what are the challenges that we have here? Well, one of the big problems is that there are at least two hundred different types of testing that people talk about, publish, use on projects. And most testers really aren't that familiar with that many of them. And they may know the names of some of them. But that doesn't necessarily mean they know how to actually perform them. And naturally, test managers and project managers and developers are going to be aware of

even fewer types of testing than the testers themselves.

And if you look at the testing planning documentation, whether it's test strategy, test plan, the testing section of a software development plan, they usually only cover a relatively small number of types of testing. And the problem that we have there is if you're not planning to do it, then you're unlikely to do it. And some of these types of testing are very useful.

Goals of the Presentation

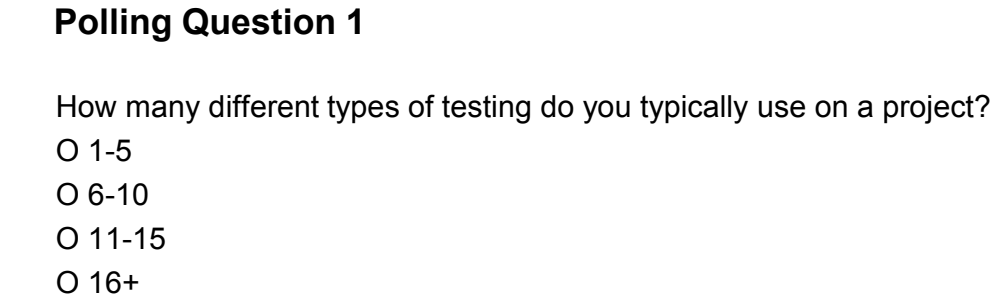


**008 So, what are my goals today? First of all, to make it clear to everyone who is sitting in on this presentation that there is a lot of different ways to perform testing, that it's a complex discipline with its own technical jargon. It's not something where you hire anyone off

the streets and say, "Go do testing." And there's also a lot of overlap between these different types of testing. So, it's a question of you might have four or five different ways of performing a certain kind of test, which one is the appropriate one for you.

So, when we're done today, what I hope you do is you go and take a look at your test strategies, your test plans, and ask yourself are they sufficiently complete. They won't be complete. But are they sufficiently complete for your project? And as yourself, do my testers, do I need additional training? Are there certain types of testing that sound interesting, but I just don't know how to perform?

Polling Question 1



**009 Shane McGraw: Okay, this leads us to our second polling

question that will appear now on your screen. And we'd like to know how many different types of testing do you typically use on a project. So, we'll give you about fifteen or twenty seconds to vote there. While you're doing that, just a reminder for anybody that's logging in now, if you go to the files tab at the bottom of your console, you can get a copy of today's presentation slides along with other work from the SEI. And also for those of you using Twitter, you want to be sure to follow @SEI_SSD, as in software solutions division, and use the hashtag of SEIwebinar.

And I'm going to stop the poll now. And we will preview the results. We've got about sixty-eight percent, Don, saying one through five, twenty-six percent at six to ten, four percent at eleven through fifteen, and two percent at sixteen plus. So, I'm assuming the lesser numbers--

Donald Firesmith: That sounds about right. And that does back up what I was saying before.

What is Testing?

**010 So, moving along here, what do I mean when I say testing because a lot of people equate testing with QA? And there are actually multiple ways of doing--

What is Testing?

What is Testing?

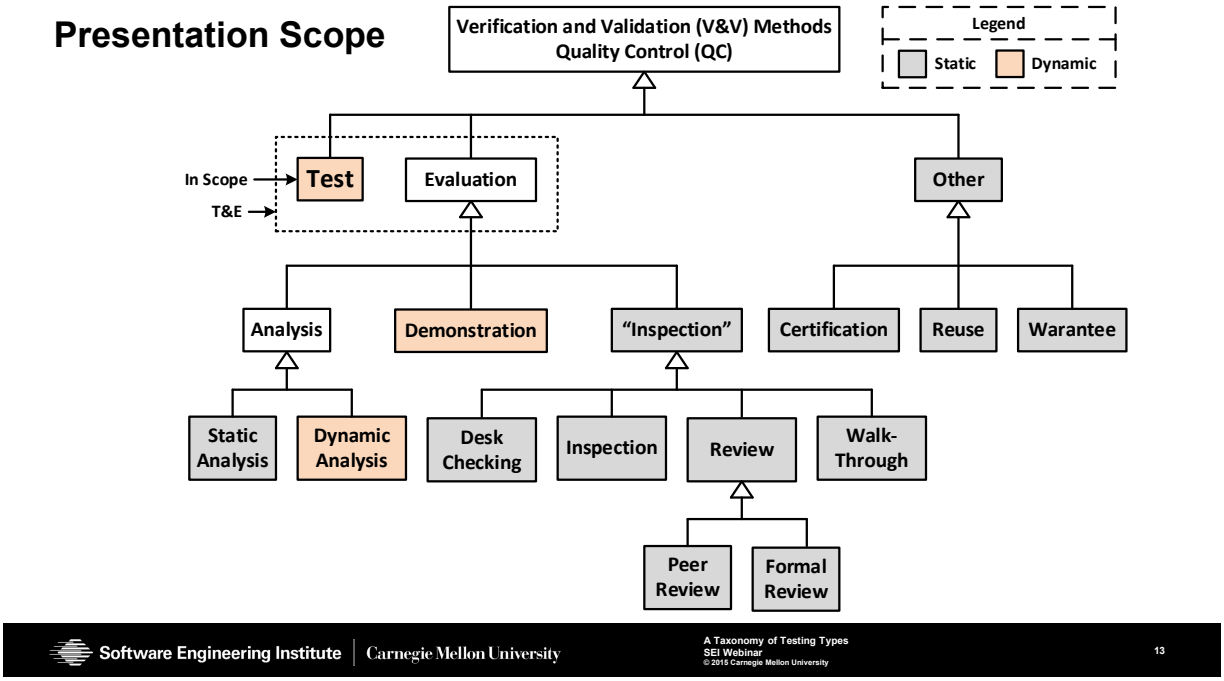
Testing

The ***execution*** of an Object Under Test (OUT) under specific ***preconditions*** with specific ***stimuli*** so that its ***actual behavior*** can be compared with its ***expected or required behavior***

- *Preconditions:* pretest mode, states, stored data, or external conditions
- *Stimuli:*
  - Calls, commands, and messages (control flows)
  - Data inputs (data flows)
  - Trigger events such as state changes and temporal events
- *Actual Behavior:*
  - *During Test:*
    - Calls, commands, and messages (control flows)
    - Data outputs (data flows)
  - *Postconditions*: post-test mode, states, stored data, or external conditions

**011 QA verification/validation. So for the purposes of today, what I'm going to say is you're going to take an object under test, and you're going to execute that object. And you're going to put it, before you execute it, into a certain known state under certain preconditions. You're going stimulate it with certain inputs. And then you're going to take a look at the actual behavior of the object under test and compare it to what you expect the-- what the oracle says the behavior should be.

Presentation Scope



**013 So, what does that mean as far as what I'm going to be covering today? If you look at the slide right now, verification/validation methods, there are a lot of things in that tree. These are also different ways of performing quality control on products. And I'm only going to be covering the box in the upper left that says test. So, I'm not covering all of test and evaluation. In fact, I'm not covering all of V and V. Given that there's two hundred different ways of doing testing, at least, there's obviously a lot more if I add the other ones on.

Types of Testing

Types of Testing

A type of testing is:

- A specific way to perform testing
- A class or subclass of testing
- Much narrower in scope than a testing paradigm

There are relationships between the various types of testing.

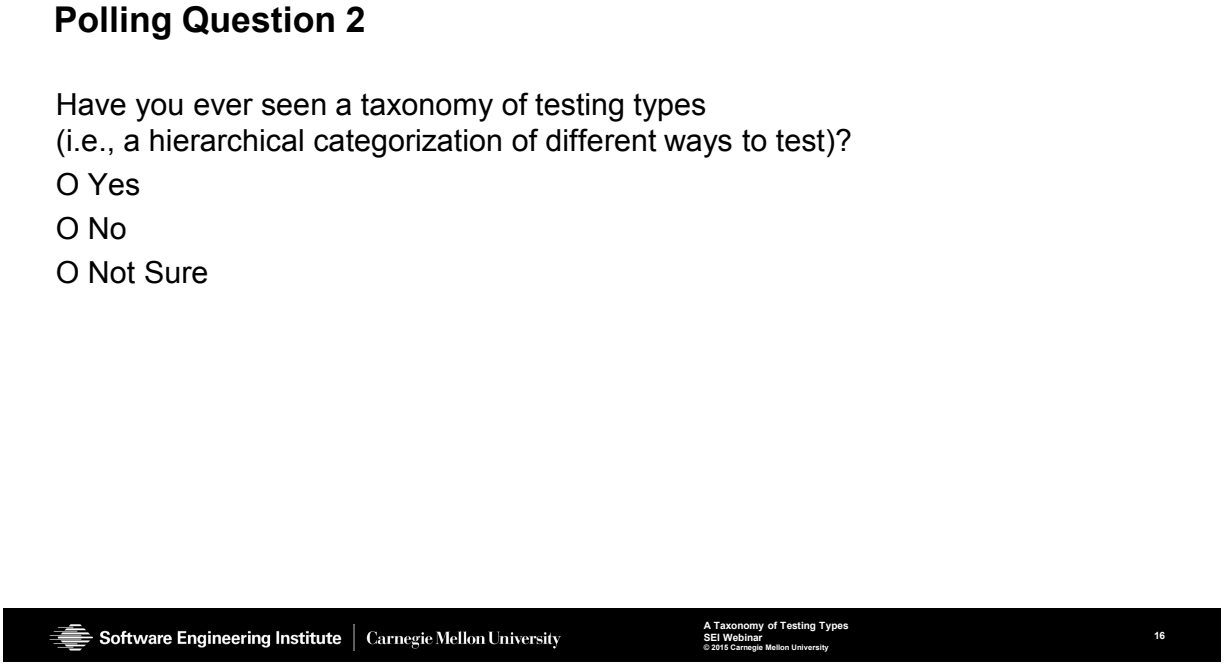
Most testers know:

- A lot about a few types of testing
- A little about some additional types of testing
- Very little about a sizable number of testing types

**015 So, let's get into the actual taxonomy itself, and first discuss what I mean by a type of testing. It's essentially a specific way of performing testing. And you can think of it as a class or a subclass of testing in the OO sense. So, it's relatively specific, certainly more specific, narrower in scope, than a testing paradigm such as Agile testing or DevOps testing or traditional testing.

And all these different types of testing are going to have relationships amongst themselves. So, when I talk about a tree structure, it's really going to be a lot more complex than that. And as I said before, this is going to be a large tree. And most people are going to be intimately familiar with only a relatively small part of it.

Polling Question 2



**016 Shane McGraw: Okay, that leads us to our third polling question, which will appear on your screen now. We'd like to know have you ever seen a taxonomy of testing types, for example a hierarchical categorization of different ways to test. And while we do that, let's see if we've got any questions in the queue. We'll give a couple seconds to vote. We actually have one from Joseph saying at the start, Don, you mentioned over two hundred ways to test. So, he wants to know with so many different types of testing, how do you expect someone to learn these without getting overwhelmed.

Donald Firesmith: That's a great question. It's a little bit like a developer who's learning a pattern language, let's say a design pattern language, or coding idioms, things of

that nature. One never learns them all at once. You don't sit down and just go through them one after the other. So, you have to pick put which ones are most important to your project, to the kind of answers you're trying to get by performing the testing. And it's something that you basically learn over time. And you'll find that people who've been testing for ten, fifteen, twenty years, will know quite a few more different types of testing than someone who's only been in it a year, two, or five years.

So, what you can do, however, with the large taxonomy, is you can use a checklist. You can go through it and ask yourself one by one, is this something relevant to my project, is this something that would be useful to do, or is this just not going to be cost effective and worth the time.

Shane McGraw: Got you. And for our poll results, we've got thirty-five percent saying yes they have seen a taxonomy, fifty-three percent no, twelve percent not sure. Give it back to you.

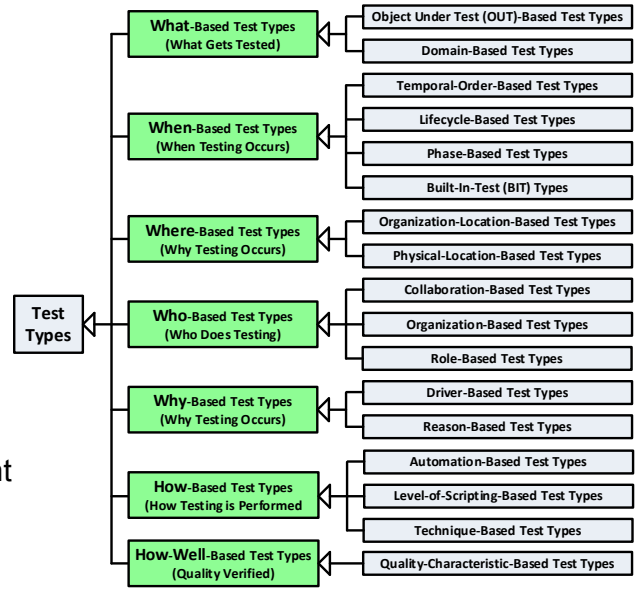
Types of Testing

Types of Testing

16 Categories of Testing Types Answering the 5W+2H Questions:

- What?
- When?
- Where?
- Who?
- Why?
- How?
- How Well?

These supertypes are not disjoint (think multiple inheritance)!



**017 Donald Firesmith: Moving right along, when I first started collecting these different types of testing, I started off with just a list and alphabetized it. But it didn't take very long to realize an alphabetized list gets to be seventy, a hundred, to a hundred and fifty long just doesn't make sense. And so, I initially grouped them together in fourteen different categories that seemed to make sense. And I gave a presentation inside the SEI. And one of my smart colleagues pointed out that one way you could group these was in according to the traditional 5W2Hs of philosophy and writing. In other words, five Ws, you're asking the questions of what, when, where, who and why, and the two Hs, how and how well.

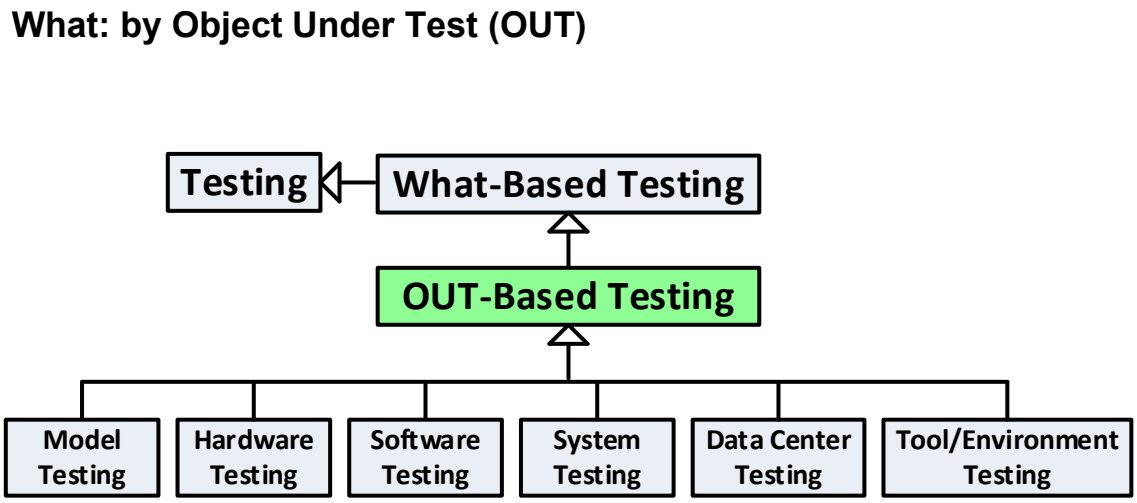
So, what does this mean? The first category at the top level here in green is what is being tested. The second one there is when am I performing those tests. The third one is where are these tests being performed. The fourth one is who's doing the testing. The fifth one is why are we doing the testing in the first place? What kind of questions are we trying to answer? And then the two Hs, how are we doing to be performing the test. These would be a test design type of approach is helping you figure out what your test cases are. And then how well are we trying to find out how well the object under test is actually working. This is where you get into the various quality characteristics and attributes of reliability, safety, security, things like that.

And then for each one of these seven, you'll notice there are one or more subtypes of testing here on the right. And I'm going to be spending ninety percent of the rest of our webinar today going through those one at a time.

Types of Testing

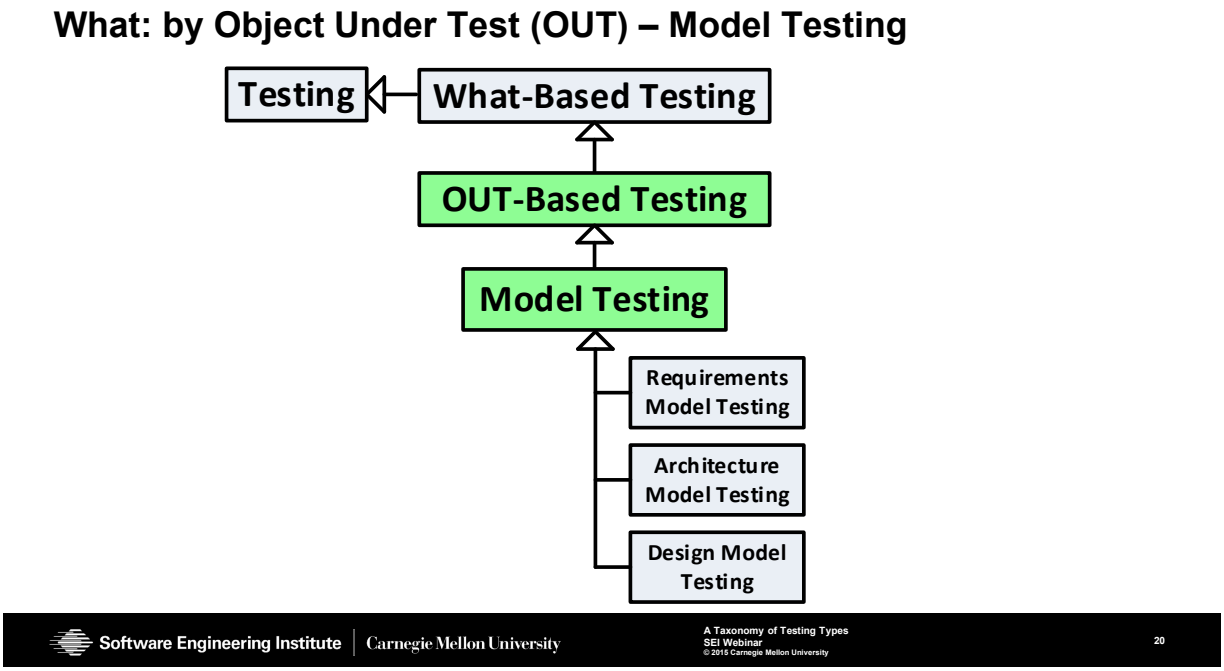
**018 So, moving on--

What: by Object Under Test (OUT)



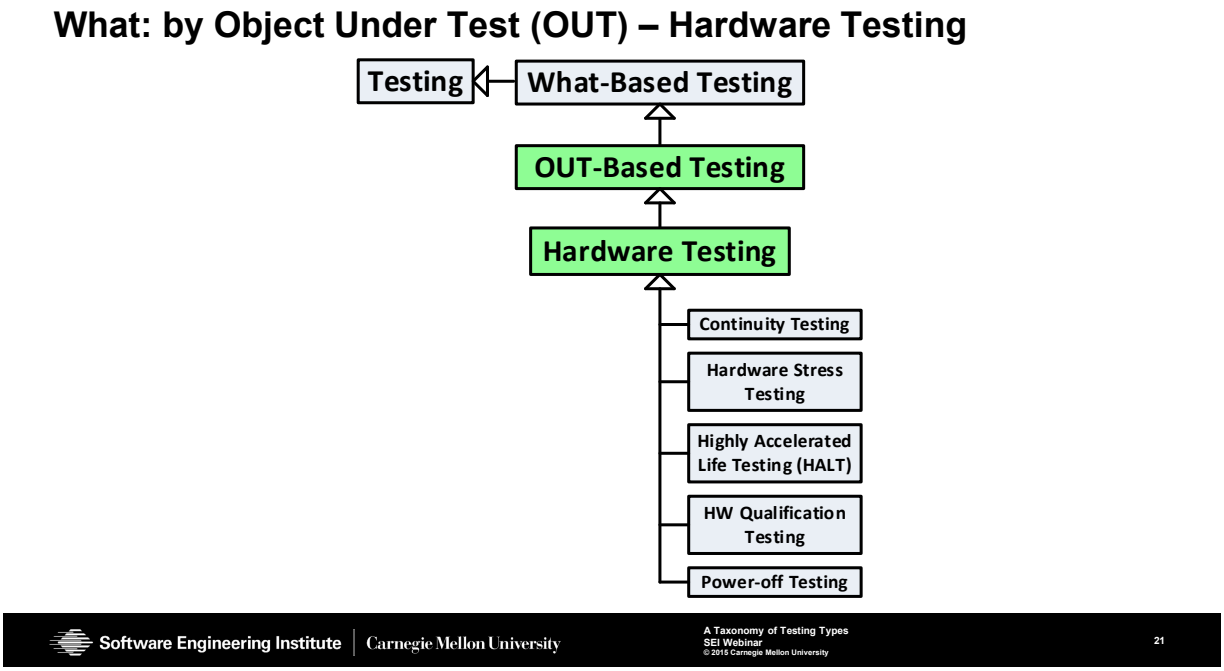
**019 Let's take a look at the what is being tested. You can think of this first as what kind of object is under test. What are you going to test here? People tend to think in our profession, "Well, we're doing software testing and system testing," which are the two in the middle. It turns out that there is also hardware that needs testing as well as the software. If you have executable requirements, architecture, or design models, you can test those before there's even any code to test. We have a data center. You can do some testing on the data center itself. So, you're not looking at a single program. But you're looking at a datacenter that might be supporting multiple ones. And then there are the tools and the environments, whether those are the development tools or test tools and test environments.

What: by Object Under Test (OUT) – Model Testing



**020 So, as I said before, if your model is executable, if you have something like an AEDL, or some kind of modeling language that you could execute, you could execute that model with specific inputs and see if it behaves the way it's supposed to just like you can with software. And you can do this at the requirements, architecture, or design level.

What: by Object Under Test (OUT) – Hardware Testing

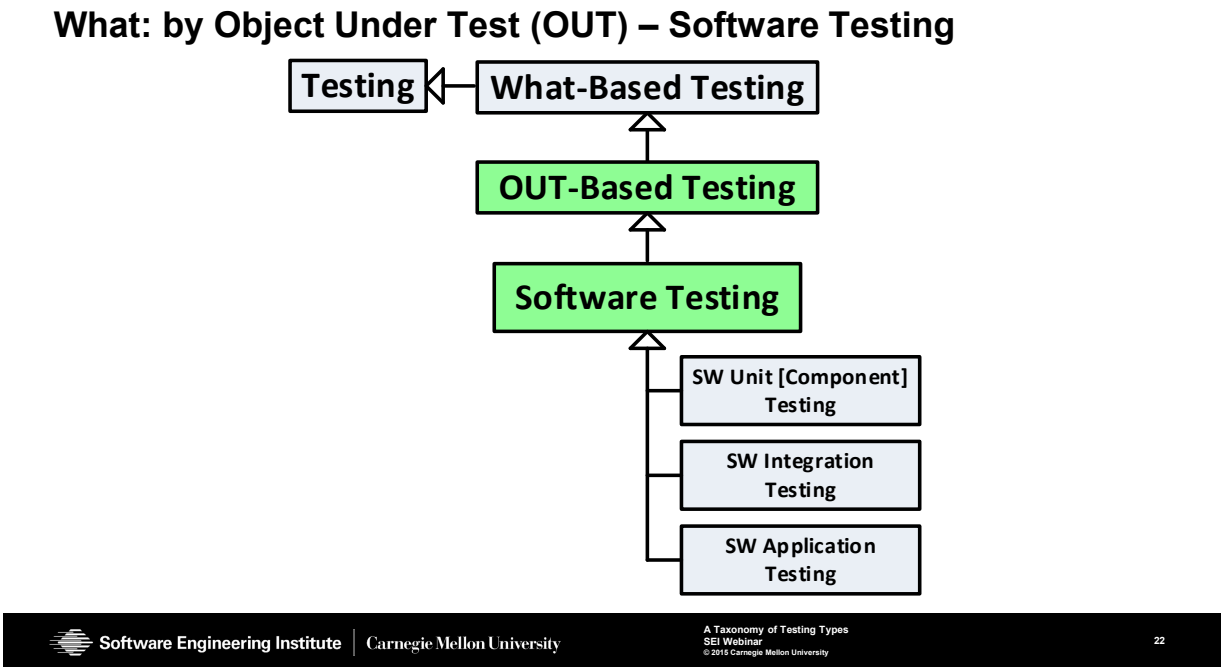


**021 When it comes to hardware, I'm actually going to cover mostly here in this list hardware that is being used to run the software. When you talk about sensors and actuators and things like that, I'll cover that more under systems. And so there's things like continuity testing to make sure that a circuit is continuous across, say ,a circuit board, stress testing to see how the hardware actually behaves when you're vibrating it or heating it up, or whatever. Speaking of which, highly accelerated life testing, you basically put the hardware through its paces to make it go through the same amount of work that it might have taken a year or two years in real time to go through.

Hardware qualification testing, make sure it's good enough to actually use

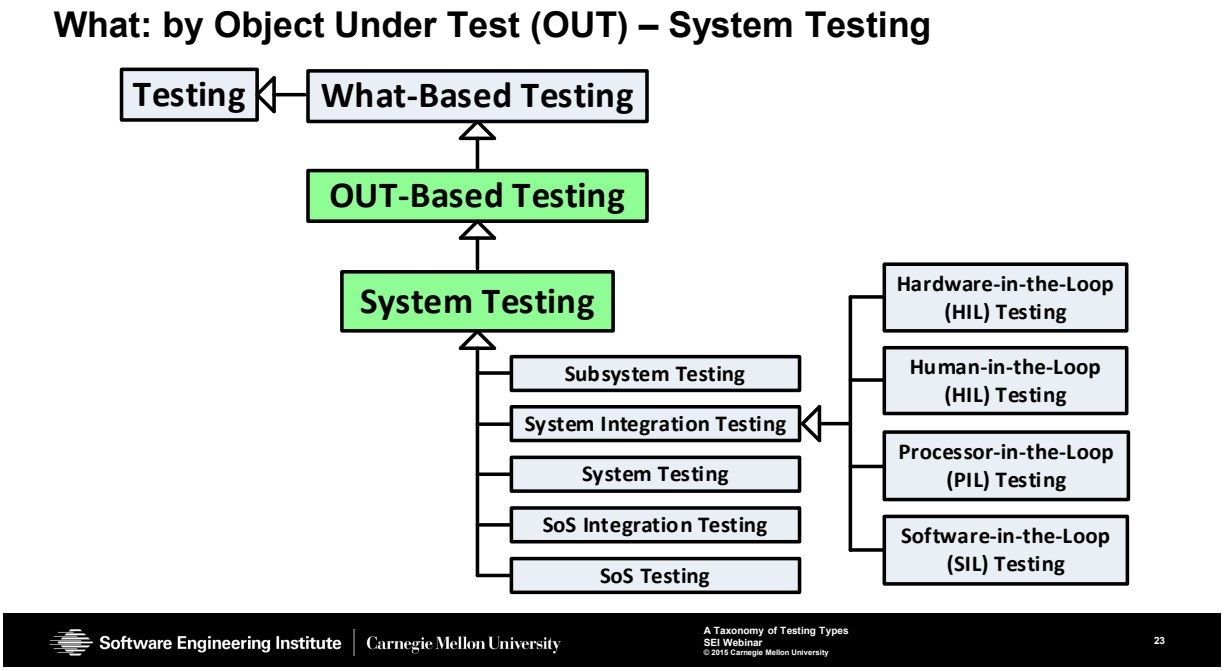
in the first place. Power off testing, when you turn the power off is the power really off?

What: by Object Under Test (OUT) – Software Testing



**022 Okay, when it comes to software testing, these are probably the three most popular kinds of testing that everyone knows about these. There's unit testing. There's integration testing at various levels of integration. And then there's testing the entire application, the entire system.

What: by Object Under Test (OUT) – System Testing

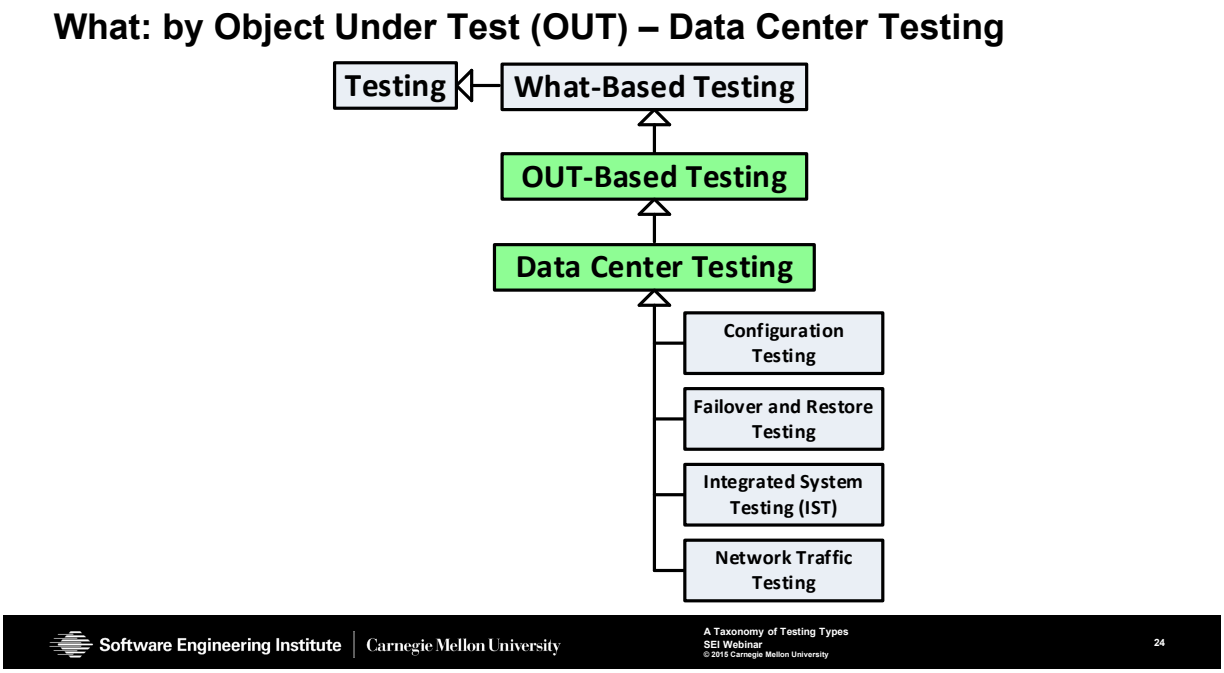


**023 Moving on to system level testing, by system I mean not just software, hardware, data, facilities, manual procedures, personnel, equipment, the whole nine yards here. And we can do subsystem level testing where we're testing the subsystems as sort of a black box. We can do system integration testing. And as you can see on the right hand here, we can do it with actual hardware in the loop, with a human in the loop, with the actual processor, with actual software. And that's to be compared with, say, using modeling and simulation where you're running a simulations instead of the actual hardware because maybe the actual sensor doesn't exist yet.

And then there's testing the entire system. And what's interesting is a

lot of people tend to stop at that level. But in many cases, to get the kind of capabilities that you need, you actually need a system of systems because that's what provides the capability that you want. And so you can do the system of system integration testing to make sure that the systems interoperate properly. And then you can test the entire system of systems as a whole with some kind of black box mission thread kind of approach.

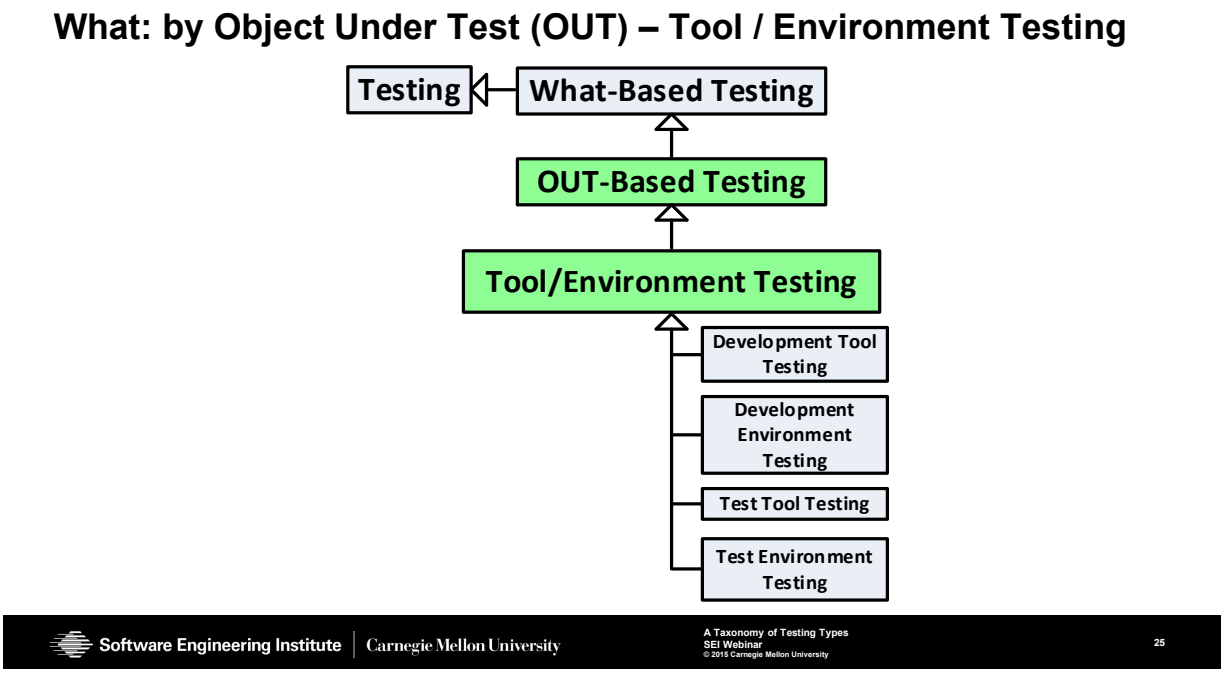
What: by Object Under Test (OUT) – Data Center Testing



**024 When I talked about datacenter testing, you can check to make sure that everything in your datacenter is properly configured. You can actually want to be able to have failover and recovery if one of the servers goes down, for example. Integrated system testing, network traffic testing, there's a bunch here.

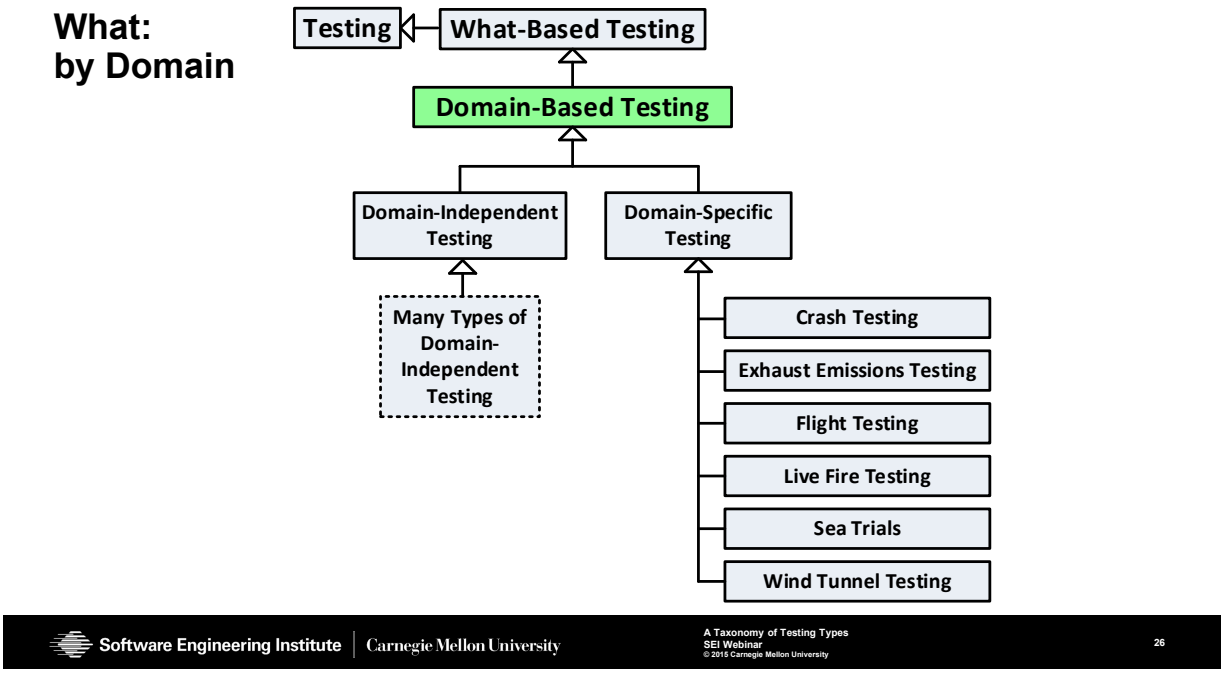
These are just some of the more commonly known ones.

What: by Object Under Test (OUT) – Tool / Environment Testing



**025 And as I said before, if you were using a tool to test, or you're using a tool to develop your software in the first place, how do you know when the object under test, when it fails, that it's actually a problem in the object under test? It could be a problem with your test tools. It could be a problem with your test environments. It could be problems with the development environment. For example, if you're automatically generating code from a model, that tool that does that might have a defect in it.

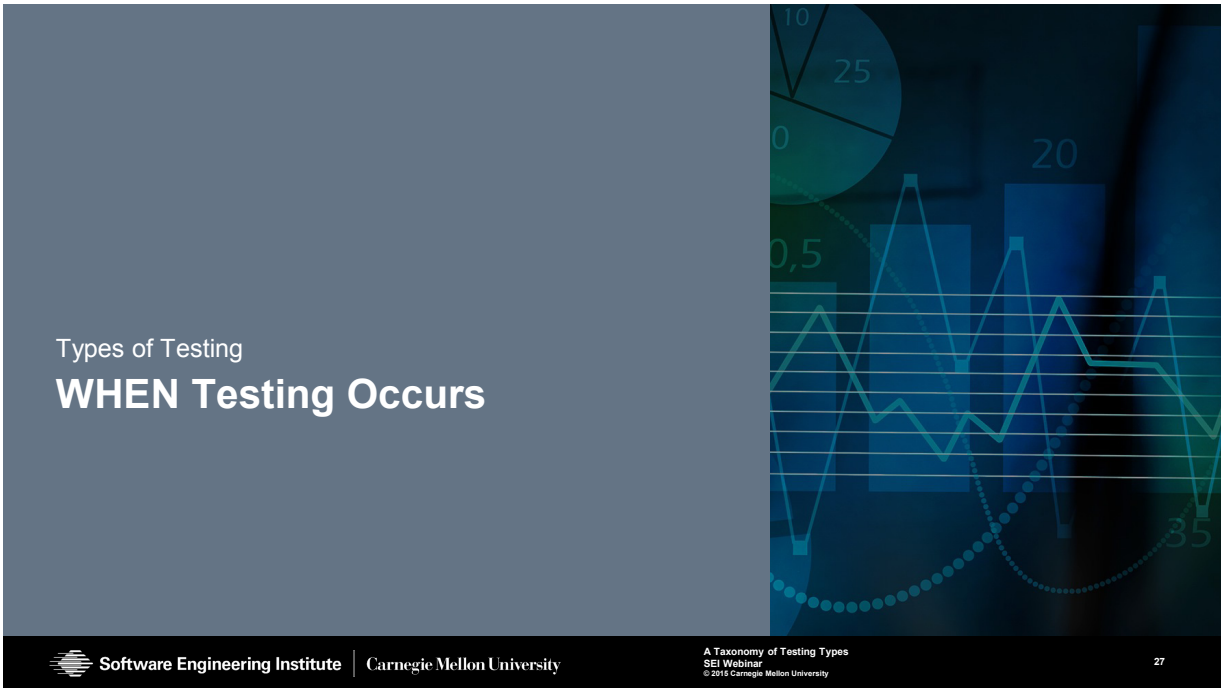
What: by Domain



**026 Okay, let's move on to a different approach. Rather than just the objects under test, let's take a look at what kind of domain we're talking about here. Most of the types of testing that I've mentioned so far are on the left hand side here. They're basically domain independent. You can use those kind of tests for testing basically anything.

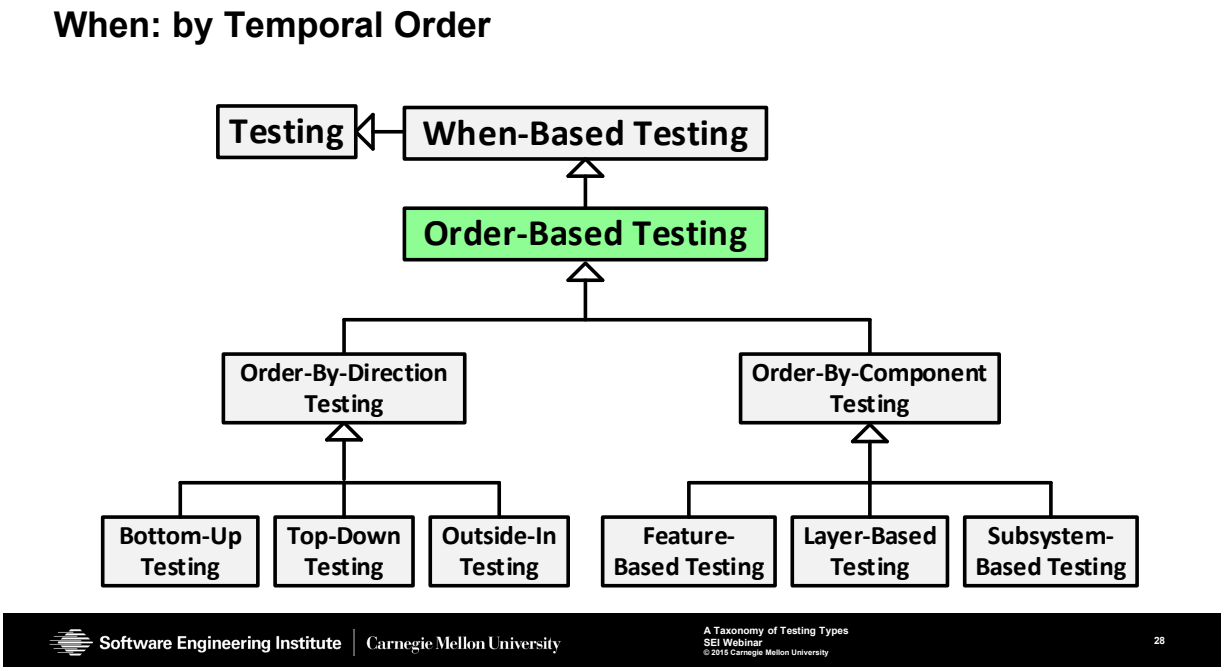
On the other hand, there's a lot of different domain specific types of testing. For example, in the automobile industry, crash testing and exhaust emissions testing are quite common. If you're dealing with aircraft, flight testing is common. If you're dealing with any sort of weapon, live fire testing. Sea trials for boats, wind tunnel testing, there's lots here. Once again, these are just some of the more obvious examples.

Types of Testing



**027 Okay, moving away from what you're testing, let's take a look at when the testing actually occurs.

When: by Temporal Order



**028 In this case, we're going to look at the order in which testing takes place. So, for example, if you look on the left hand side and look down at the bottom, you could start with your entire system. Start at the individual units and do your testing bottom up. So, you test the units. Then you test collections of the units, collections of collections until you get all the way up to the top.

On the other hand, you may very well do just the opposite. And you sometimes see this in Agile testing where you start at the very top. You come up with some behavior that the system's supposed to perform. And you write a test for that at the highest level. You create some piece of code at the very top level that implements that and depends-- everything below is totally stubbed

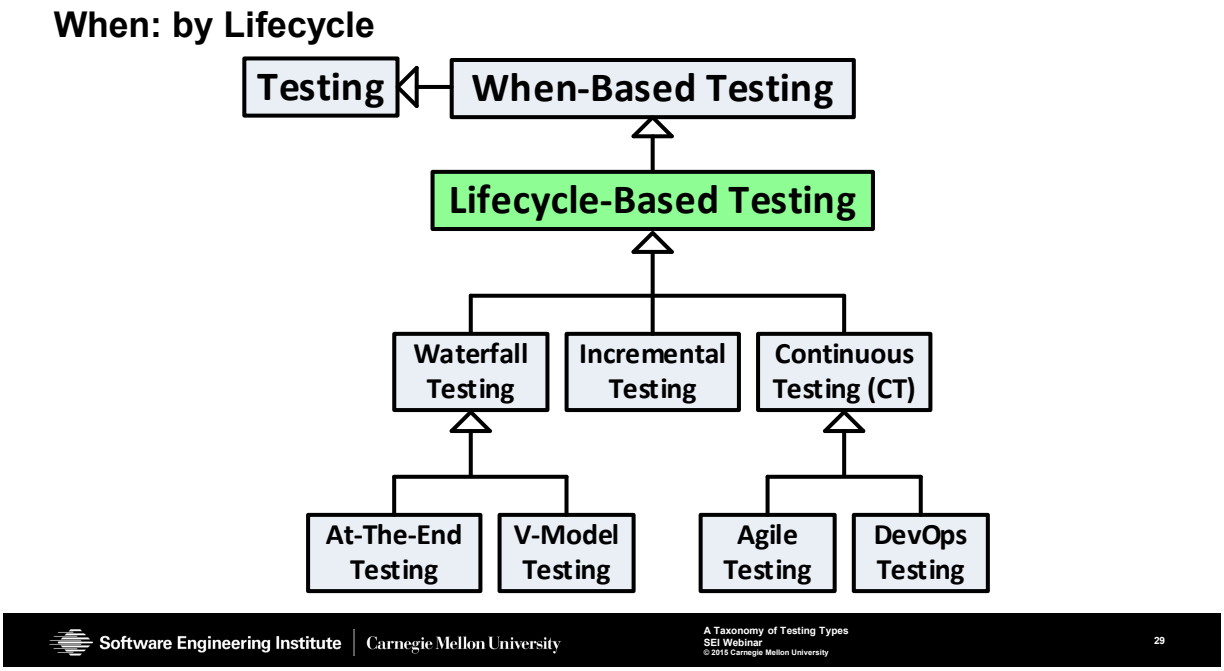
out. And then you just work your way down.

The other thing you might do is sort of a capabilities based testing, where-- or interface testing where you start at various interfaces and create something that interfaces to these external actors. And then you create something that interfaces to those, and so on. So, basically the order in which you develop on the left hand side, the bottom-up, top-down, outside-in, inside-out, the order that you develop is also then the order in which you typically test.

On the other hand, we can look at it as what kind of component we're going to test first. Feature based testing, and you see this also sometimes in Agile programs, you think of a feature that you need. You come up with some user stories based on it. You create that feature, and then you test that feature.

You could do sort of a layer-by-layer approach where you might start at the API, the application layer. You might star at the user layer. You might start lower down database and so on. Or you could do it a subsystem at a time.

When: by Lifecycle



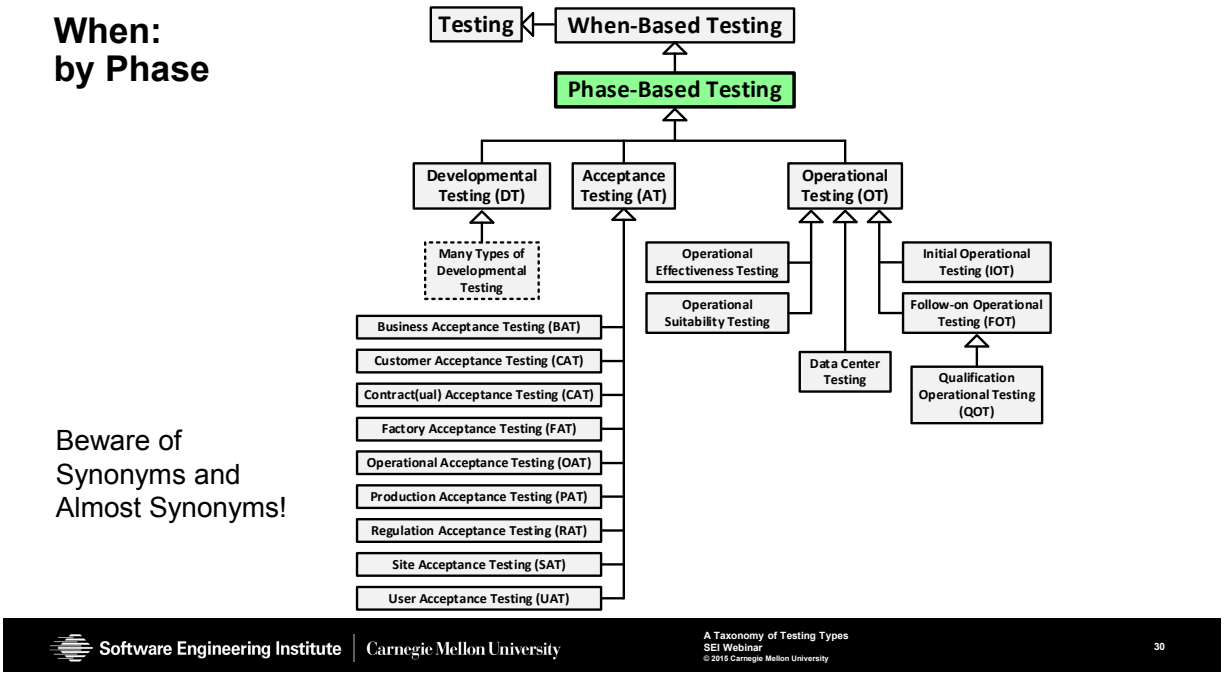
**029 Another way of looking at testing when you're talking about when the testing is being performed is the type of lifecycle that you're dealing with. So, if you're dealing with a classic waterfall lifecycle, you've got basically two traditional approaches. A really obsolete approach that hopefully no one's still doing is you wait until the very end to start doing your testing. Another thing that you can do is you can use the classic V model where you're coming up with your requirements, your architecture, your design. And then you're implementing it, units, subsystems, and systems, and so on, and testing basically in that order.

Now, most people don't do a classic strict waterfall lifecycle anymore. At the very least, they do an incremental lifecycle, which basically

says, "I can't do it all at once. So, I'll do it in small number of relatively midsized chunks." And so, I develop in that order. And I test those chunks as they come into existence, those rather large increments.

Now, if I'm doing either Agile or DevOps, I'm probably going to want more of a continuous testing kind of approach where, as I do things, I immediately test them, either at the same time, or it could even happen beforehand where I'm developing a test before I actually write the code that I test. But it's basically an ongoing testing process. It's not happening at specific times in the lifecycle.

When: by Phase



**030 Another approach that I can do is based more on the phases, if you will. And I'm talking not

requirements, architecture, design, coding, integration, and so on. I'm talking of a higher level. And basically you have developmental testing that's done by the development organization while the system is being developed. That happens first typically.

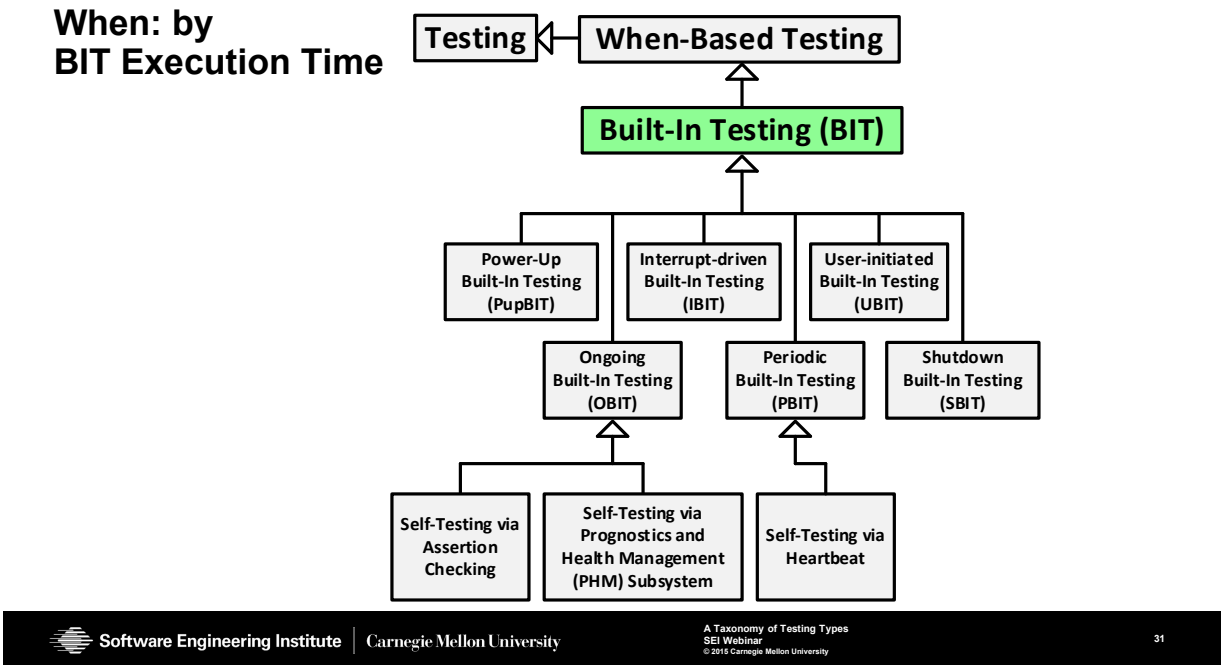
You have operational testing after the system exists and is being placed in operation, or at the very least is more or less ready to be placed in operation. And it's perhaps in the staging environment. But it's in an environment that's operationally very similar to the actual environment.

And then, in the middle, you have acceptance testing. And one thing I guess I'd point out here is that there are a ton of different kinds of acceptance testing. There's a lot of overlap between them. There are-- some of them have the same acronyms. It's very easy to get confused with them when it comes to the different kinds of acceptance tests and getting confused between them.

When you talk about the operational testing, there are several things going on here. If we're talking about large systems, like in the military, it's very important to make sure that our systems are effective, and they are suitable for what we're trying to accomplish. And so, those kinds of things are typically done by operational testers. Another way of looking at it is more of a time thing. Very often you have some initial

operational testing taking place as soon as the thing is ready for your small scale initial development. And then later on, you might have the follow up operational testing once you go into a large scale production.

When: by BIT Execution Time

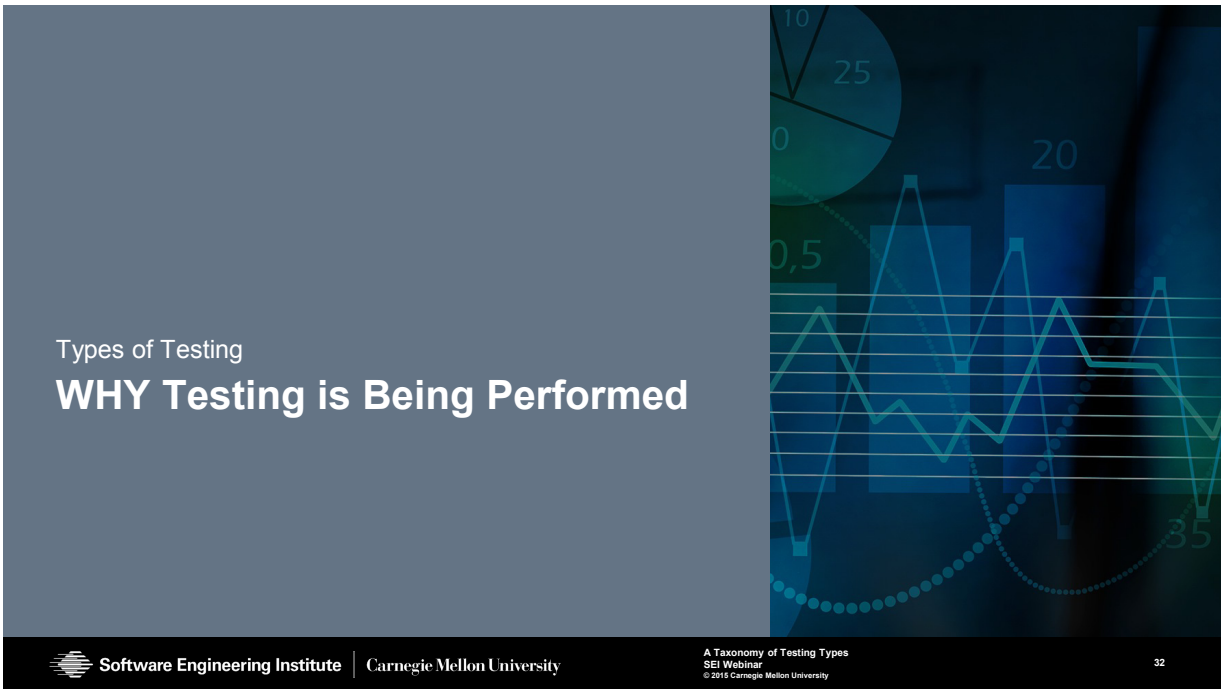


**031 Another way of looking at when the testing occurs has to do with a very strict or a very specific kind of testing, built-in tests. You can perform built-in tests at different times. You can have power up built-in test that automatically happens when you power up the system. You have ongoing built-in testing that the system itself tests itself on an ongoing basis either by checking assertions as it executes, or you might have a health prognostics management system. I got it backwards, prognostics and health management system that basically

oversees what's going on when the system itself is running. And it will be performing tests when that happens.

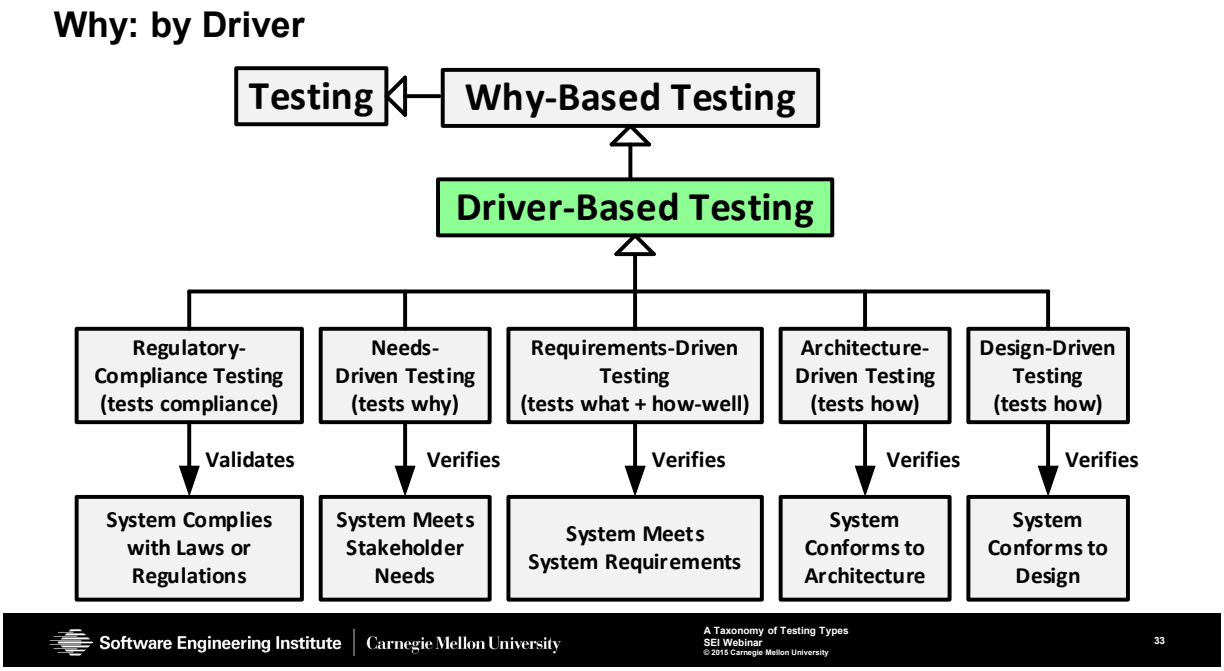
You could have interrupt-driven, built-in tests that when a certain event occurs the testing takes place. You can have periodic built-in tests so every so many milliseconds, or seconds, or minutes a test is run in the background. Heartbeat tests are performed this way. You can have the actual operator or user initiate a built-in test whenever they want. Or when you power down the system you can have built-in tests run there.

Types of Testing



**032 Okay, why are we performing testing is an interesting one here.

Why: by Driver

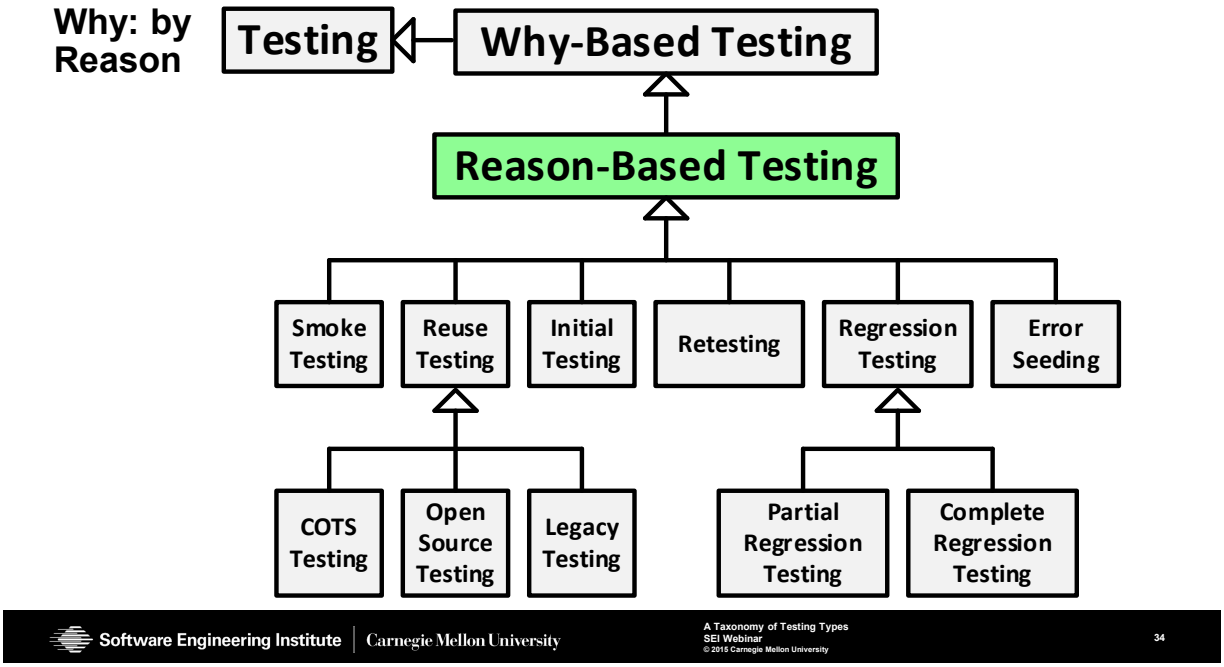


**033 Essentially, if you look at the very bottom of this slide here, you're going to see various reasons why you might want to test. You might want to make sure that the system complies with laws and regulations. And so, regulatory compliance testing is what you'd use for that purpose. If you want to make sure that it meets your stakeholders' needs, then you're going to do some sort of needs driven testing.

On the other hand, you might want to know does the system meet the requirements that have been allocated to it, or the subsystem for that matter. So, it could be requirements driven testing. And there you have to ask yourself, "How many tests per requirement do I need to adequately do that?"

You could have architecture driven testing to make sure that the system conforms to its architecture. A lot of quality oriented testing falls into that category. And then design driven testing, does the system, or really the units and the small pieces of the system, correspond to their individual design. So, white box testing at the unit level, for example, is one of those.

Why: by Reason



**034 You can also have various reasons why you might want to do testing. Smoke testing, for example, is done just to see whether or not you can even turn on the system without it smoking and sparks flying, literally when we're talking system tests, metaphorically when we're talking software testing.

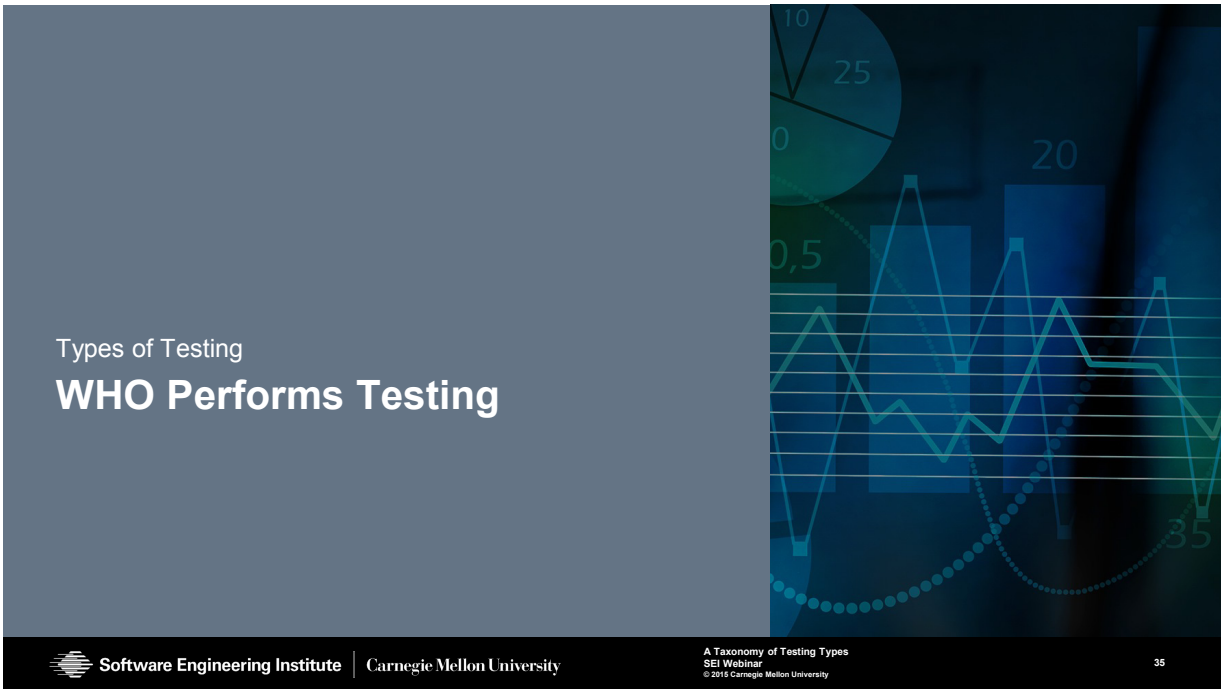
When it comes to testing for reuse, you do COTS based testing to make sure that it does what the vendor claims it does, same thing with open source testing. And legacy, you're basically saying okay does this software that I'm reusing, does it work the way it's supposed to where I have it now.

You can run some tests on some initial software or system that's never been tested before. Once you test it and you find a problem, then you can retest it to make sure that you fixed it. On the other hand, once you fix something, you may make some changes later on. And you want to make sure that you didn't break something in the process. So, you perform regression testing, something that's very heavily automated in Agile and DevOps type testing. And depending on how many regression tests you have, it may or may not be practical to run them all at once. So, if it doesn't take too long to run, you can do complete regression testing. Otherwise, you can do partial regression testing.

And if you want to know how many defects are still in the system, which is something very interesting, most people, when they deliver a system, they say, "We have this many known bugs of severity one or severity two or so on." Well, knowing how many bugs of a specific type that are known in the system, that's a good thing to know.

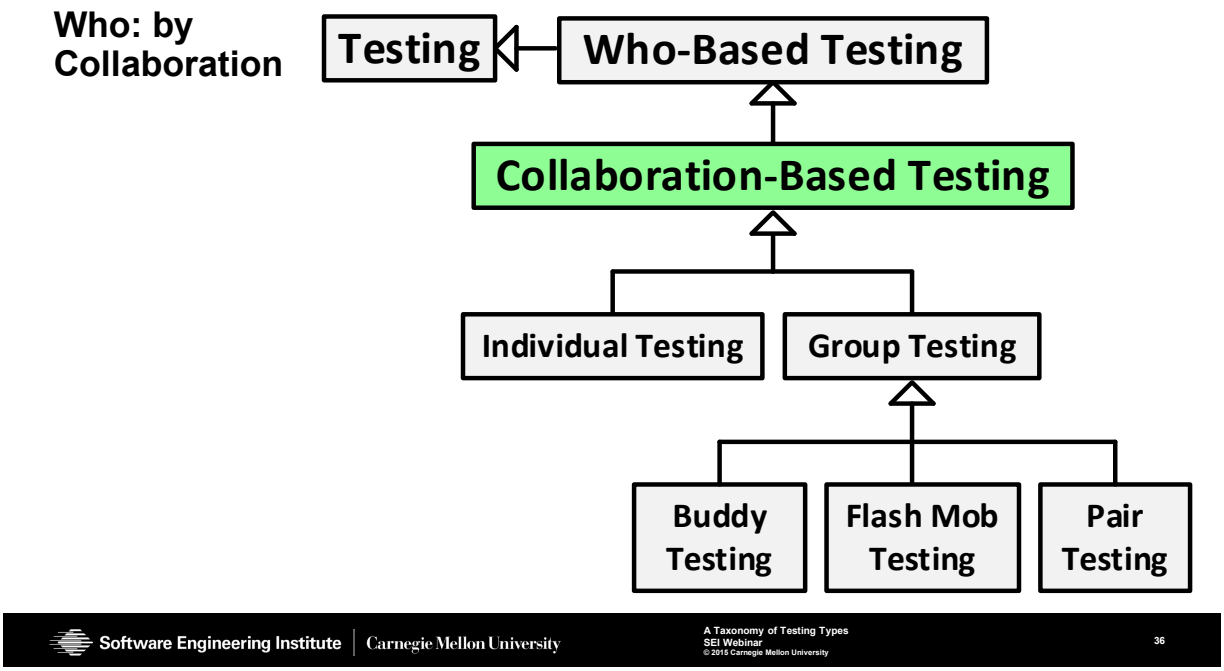
But more interesting is how many bugs are in the system grand total that you may not know anything at all about. Using error seeding can allow you to estimate how many bugs are in the system that haven't been found, that aren't known, that should show up over the next six months, or year, or two years of running.

Types of Testing



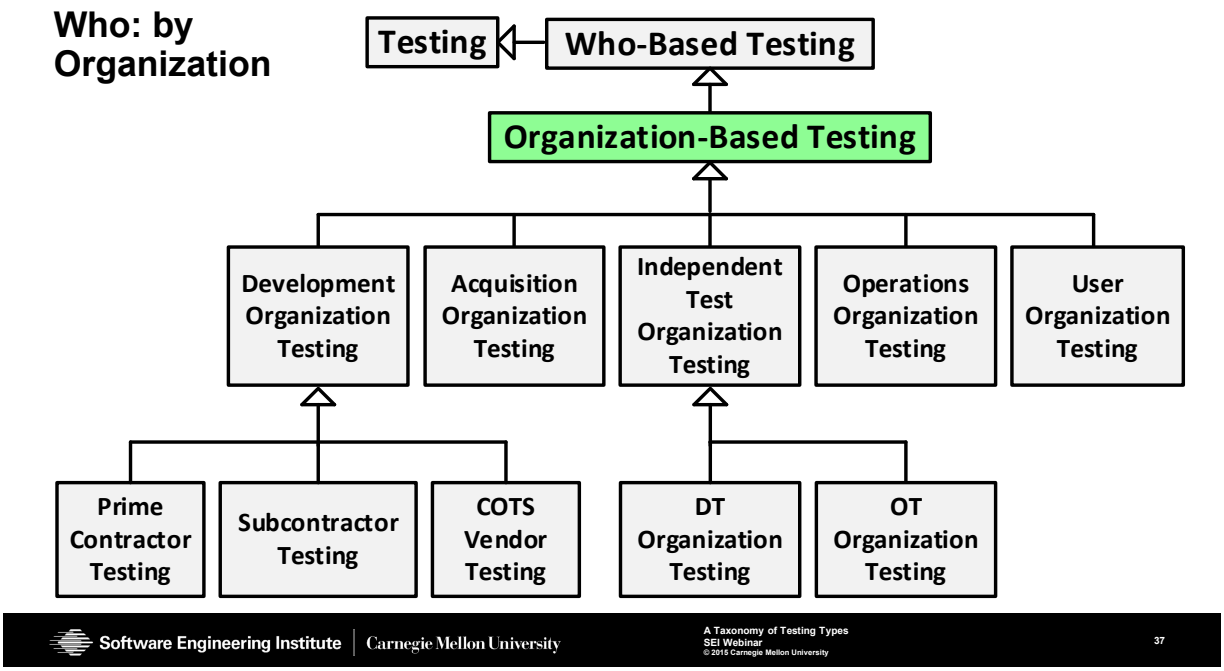
**035 Okay, who's going to perform the testing, a completely new class here.

Who: by Collaboration



**036 You can divide that up based on how much collaboration is going on. You have individuals performing the testing. Or you can have multiple people working together to perform the testing. You can have buddy testing or pair testing is a very common thing. Pair testing, think of that as being very similar to pair programming. Instead of having two people work together on coding, you have two people work together on testing. And you can have things like flash mob testing where you basically run out into the corridor, grab everybody who's free, and bring them in and do some testing right there. That's very common in sort of an alpha usability testing type environment.

Who: by Organization



**037 Organization based testing, what organization is doing the testing? You can have the development organization do the testing, the DT that I mentioned before, the developmental testing. And that testing is typically done by like a prime contractor, a sub-contractor, some vendor who is supplying an individual part of the system.

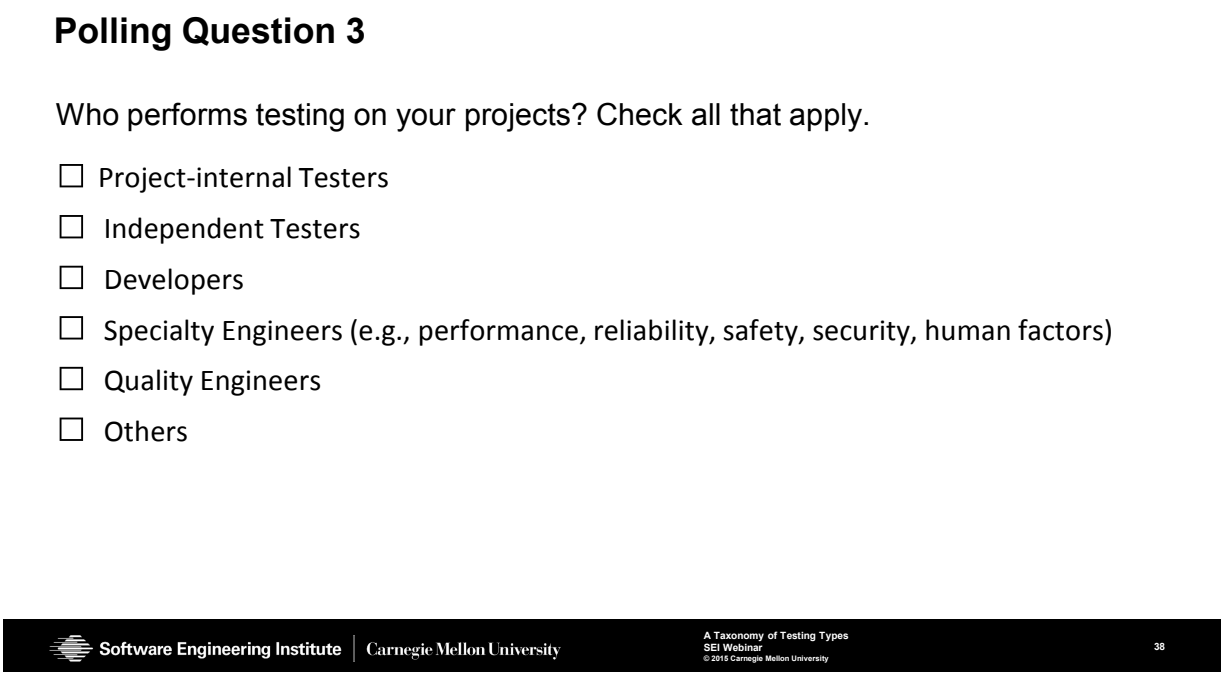
Sometimes, you have acquisition organizations doing the testing. So, for example, I'm working on one project now where the Air Force program office is responsible for doing a certain amount of the testing.

You can have an independent test organization come in and do that. That could either be a developmental

test organization separate from the project. Or it could be an operational test organization. For example, in the Air Force, AFOTEC is an organization that does a lot of that.

You could have an operations organization do the testing so that they're not really a test organization. They're the actual operators. But they are performing some ongoing testing. And we'll talk a little bit about what those might be. Or you can have users, the user organization as opposed to the operations people do the testing.

Polling Question 3



**038 Shane McGraw: Okay, this leads us to our fifth and final polling question. It's going to be posed now. And we'd like to know who performs testing on your projects. So, we'll jump into some questions while we give you a chance to vote. So, we

have a question from Roberta, Don, wanting to know which testing types do you believe that cannot be automated.

Donald Firesmith: There are definitely some testing types that can't be automated. The most obvious ones are human usability testing where you're trying to see how long it takes a human to actually run an operating, perform their job, how many mistakes they make when they're doing this. So, clearly anything that, by definition, needs a human to perform that testing obviously is something that you can't automate.

There are certain kinds of testing that you may eventually automated but you have to do in a manual manner first. For example, exploratory testing is something that really is something that a tester does with-- also error guessing is another kind of testing where a human is making all these choices. And typically, what happens is you're running these kinds of tests manually at the beginning using the user interface typically, or possibly some test tool. But what you haven't done is you haven't automated the actual test itself, at least not yet.

So, there's about five or ten of the kinds that do not make sense to automate. Then there are a whole slew that do make a lot of sense to automate.

Shane McGraw: Got you. Let's get one more in from LeeJo asking are there any studies or research that

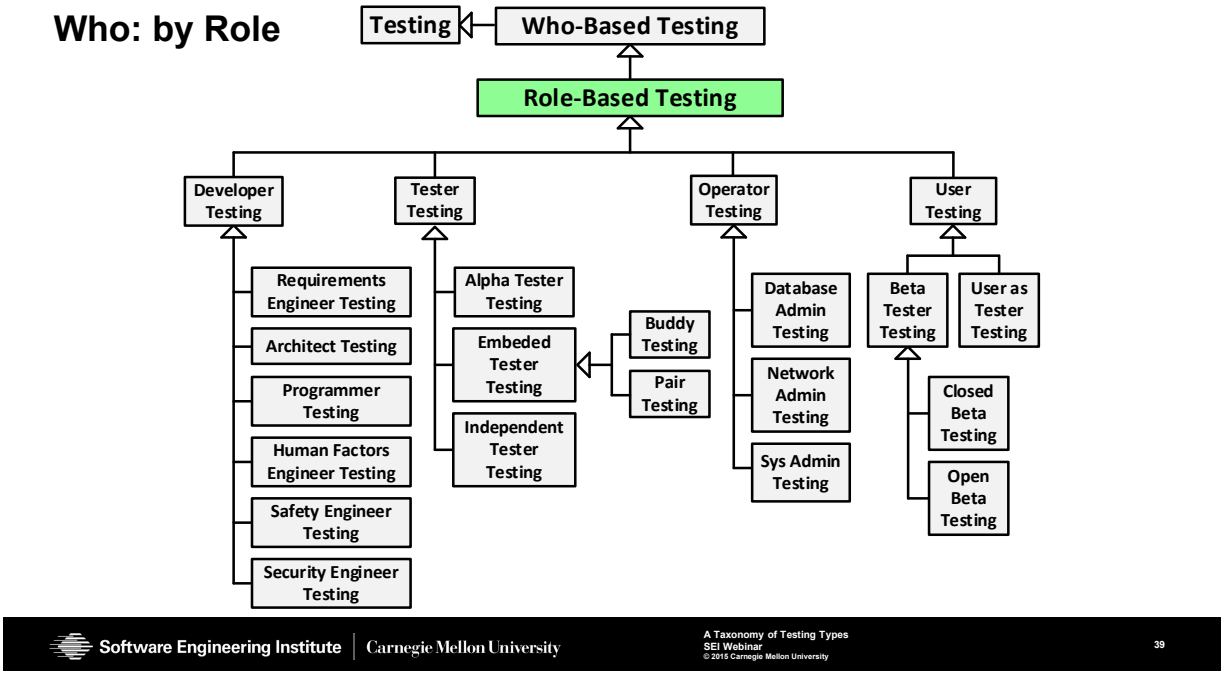
shows what types of testing is prevalent in various industries.

Donald Firesmith: I know there are some interesting studies that talk about how effective different kinds of tests are. Unit testing, for example, finds a certain percentage of the defects. Integration testing finds another percent and so on and so forth, which helps you know that there's a lot of different kinds of tests that have to be done on the same program. I do know that there are some. I can't think of any specifically off the top of my head. I would suggest sending me an email, and I will follow up with you. I do know that there are people out there who are well known for collecting statistics like this. Capers Jones, for example, would be a good person to contact. Google his name, Capers, C-A-P-E-R-S Jones. And he's very good at answering your emails.

Shane McGraw: Okay so, back to the poll. We had thirty-three percent was project internal testers, twenty-six percent independent testers, sixteen percent developers, four percent specialty engineers, fourteen percent quality engineers, and seven percent other, so quite a mix there.

Donald Firesmith: And that is what you'd expect. In fact, you would not expect it to add up to a hundred percent because very often, you have multiple types of people working on the same project doing different types of testing.

Who: by Role



**039 So, moving right along, in fact here we go right here on who's doing the testing. At the highest level, you can think of developers doing testing, testers doing testing, operators doing testing, and users doing testing as sort of very high level of decomposition. If you look at the developers' side over here, the interesting thing that you note is it's not just programmers testing their own code doing unit testing. You have higher level people like requirements engineers and architects and designers who might be test executable models that they produce. You also have a lot of specialty engineering testing being done by specialty engineers, safety, security, reliability, robustness, human factors, people doing usability testing.

When it comes to testers doing the testing, there's a lot of possibilities here. I think the two main ones to think about is you often have like a separate test organization and a tester who's a member of that organization doing the testing. If you're doing an Agile approach, at least for some of the lower level testing, your tester is a member of the development team just like everybody else. And so, that would be a developer who's embedded in the development-- a tester embedded in the development team, not independent of it.

When it comes to operators, it depends on what you're doing, sysadmins, network admins, database admins will be doing some ongoing testing of the databases, the network, the data center to ensure that everything is working the way it's supposed to, that no problems are occurring.

And then when it comes to user testing, certainly beta testing is a very popular one in the commercial world. And you can have either closed or open beta testing depending on whether or not you're invited to be a beta tester, or anybody can be a beta tester.

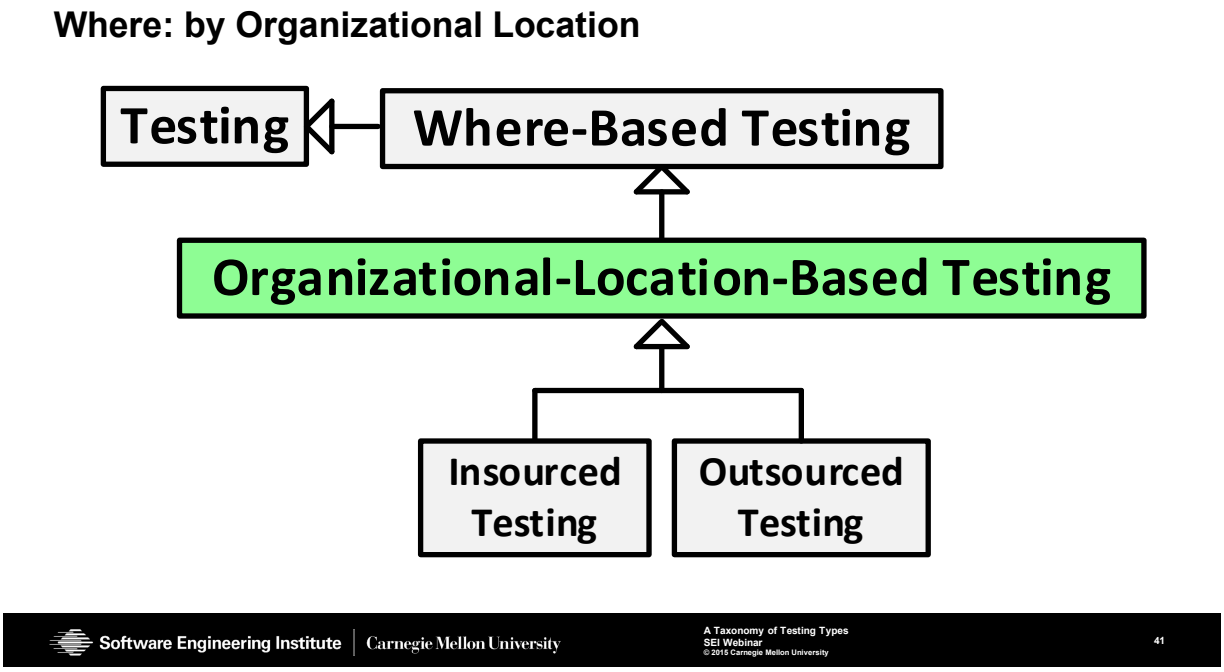
Last thing there, user as tester, not exactly what I would recommend. There are some companies out there how basically figure that the best way to save money is to not have any testers of their own but just make the users the

testers. Not a good way to keep the users, but there are some companies that do that. And sometimes, you suspect some companies are doing it even when they're not.

Types of Testing

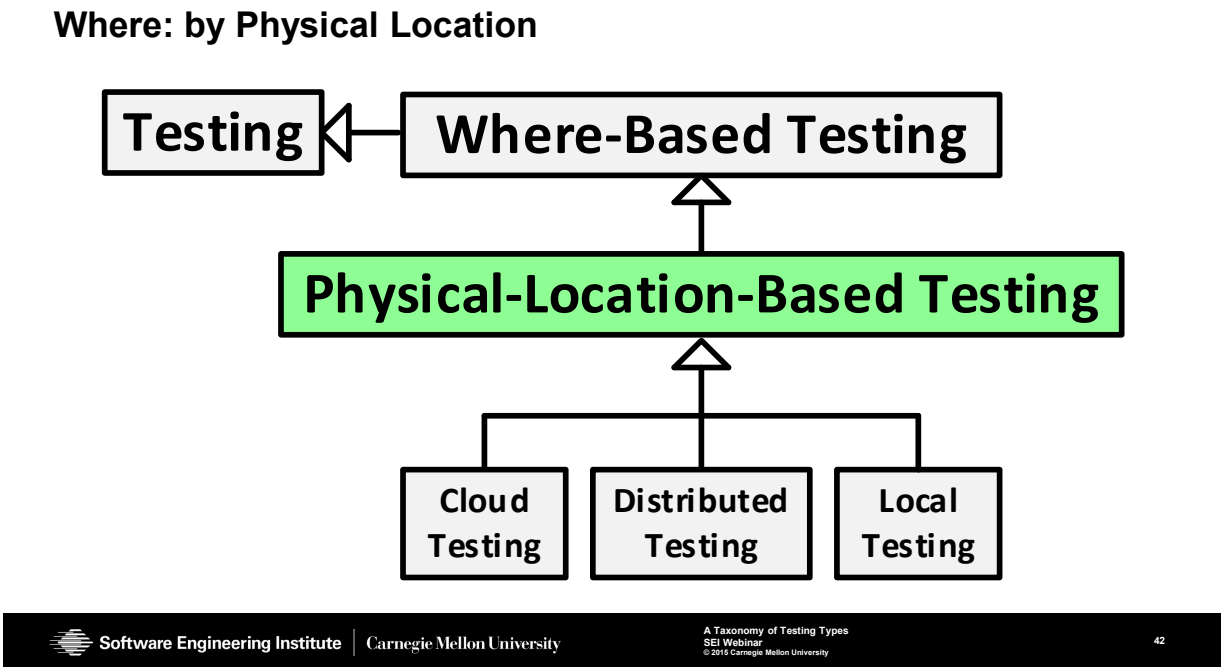
**040 No names being mentioned there, anyway.

Where: by Organizational Location



**041 Where testing is being performed, okay the classic one here is historically, we've done testing in house, in source testing. A lot of people have then moved to do outsource testing, moving your testing to a testing-as-a-service company, maybe in India, China, or wherever. Both of those have their pros and cons. Part of the problem, though, is testing is an actual profession that requires a great deal of experience and training. And essentially, outsourcing it to some other organization because they're the low bidder and the cost of paying an employee in that country is far less isn't necessarily the wisest thing to do. I've got a lot of friends in India, but I've got to say that that's not necessarily the best thing for your company.

Where: by Physical Location



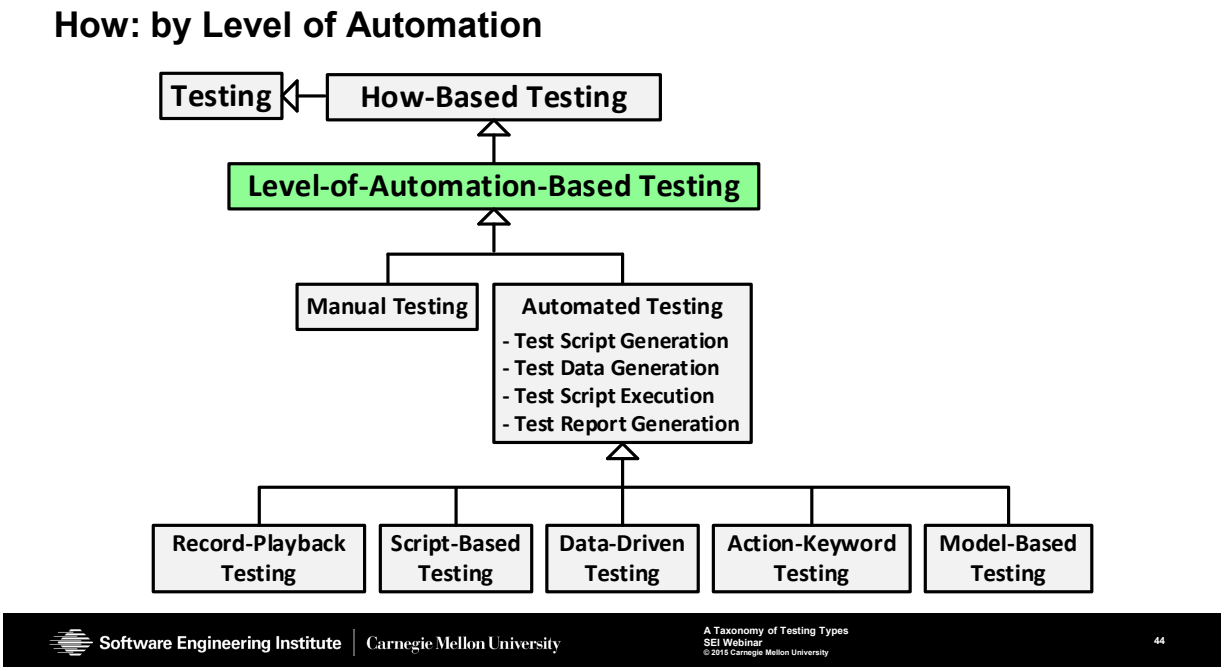
**042 You also can take a look in terms of the physical location of where the testing is taking place. You can have all the testing in one physical location. It might be in your building where the entire program is all being done in the same building. You can have different parts of the system being developed in different cities in different organizations. So, you can have distributive testing.

And then the big popular one these days is cloud testing where instead of having the test environments in your own company, having to pay for them, have the license fees, and just keeping track of all that, there are companies out there that are going to supply these test environments for you. And you can do your testing there. And certain kinds of testing, especially mobile testing, that's an excellent thing to do in most cases.

Types of Testing

**043 Now, how testing is performed. This is one of my favorites here.

How: by Level of Automation



**044 And this is one that I strongly suggest people take a look at, the next few that are how based. We already had the question on automation. Manual testing, basically everything's done manually, although you may have some test tools that help you with part of that. Essentially, a human's typing at a keyboard the whole time.

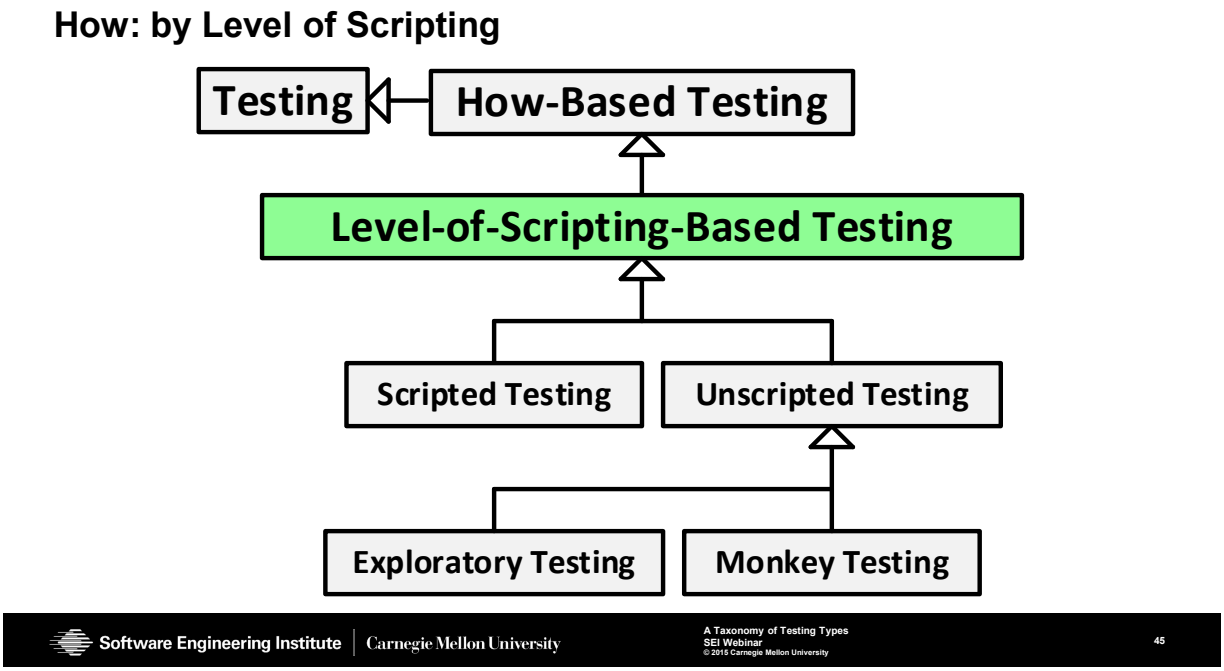
Now, when it comes to the actual automated testing, we can automate the generation of test scripts, we can automate the generation of test data. We can actually execute the test scripts in an automated form. And we can report the testing. Those are the main things that our test tools do for us.

And if you look at the very bottom, you'll see an almost sort of a

historical progression. Initially, a lot of these tools were record and playback worked at the user interface level. Then you have some script based testing tools where you use some kind of scripting language and actually write the code of the script yourself as if you were writing test code. You can then have data driven testing where you are providing data as input to it. You can have action keyword where some of the scripts are basically reused, and you just use names of scripts instead of the actual scripts.

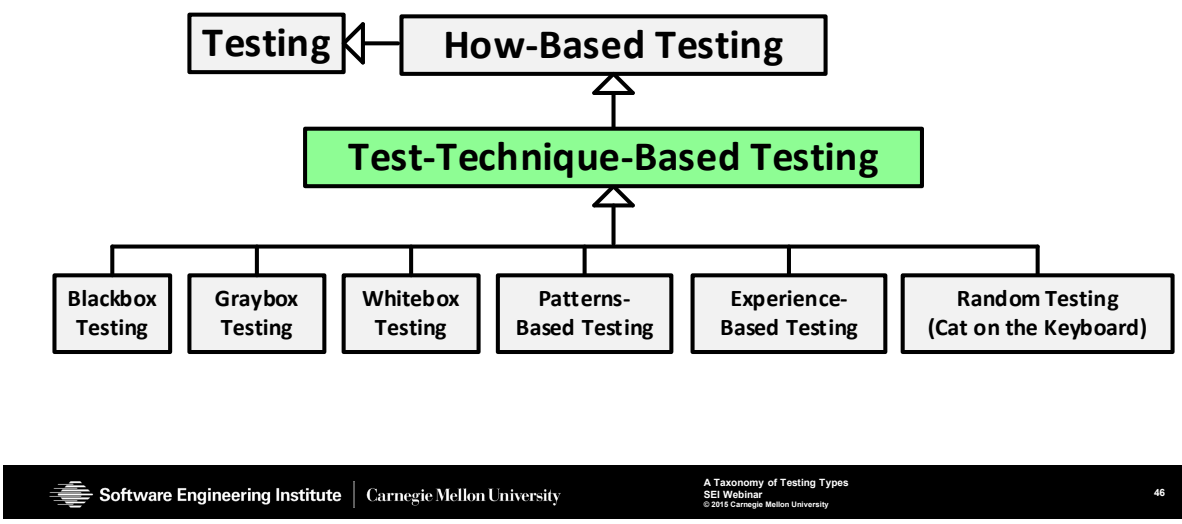
And then the one I find very interesting is if you've got good quality models, you can automatically generate your testing from some of those models. So, model based testing is an idea here that is becoming more and more popular because the test models are easier to maintain than the actual test scripts themselves. And if you're doing incremental, iterative, evolutionary, Agile type lifecycle where you're changing things a lot, it's a lot easier to change and maintain the test model than it is to change the test scripts and the test code.

How: by Level of Scripting



**045 Turning out, here's another one, script based testing versus unscripted testing. A lot of people write down either a very detailed script on how the testing is performed. Some people do a much higher level script leaving a lot more leeway up to the tester on how to run that script. And then some tests are totally unscripted. And the classic example here would be exploratory testing where essentially, as you run manually a test, and you learn from that test, that might drive the next test that you want to perform. And that might drive the test after that. So, instead of planning out your testing upfront, what you're doing is you're basically doing your test planning, you test development on the fly.

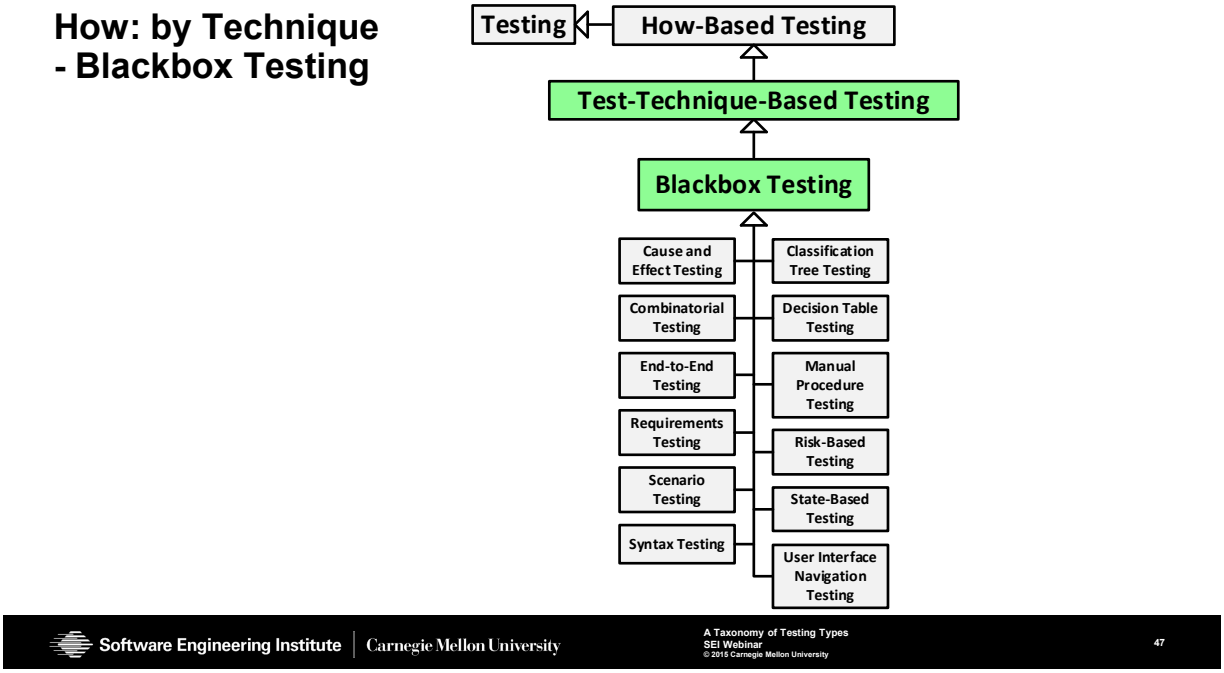
How: by Technique



**046 Test technique based testing, okay this essentially is talking about test design. This is talking about how do I come up with what my test cases are, what are my test case selection criteria, what are my test completion criteria. And these fall into a bunch of different categories. These are the ones where a lot of testers need to learn different kinds of these testing approaches that I'll get to shortly.

In general, these test types fall into these categories. They're either black box testing, gray box testing, or white box testing. They could be pattern based testing. They could be experience based testing, or various kinds of random testing.

How: by Technique - Blackbox Testing



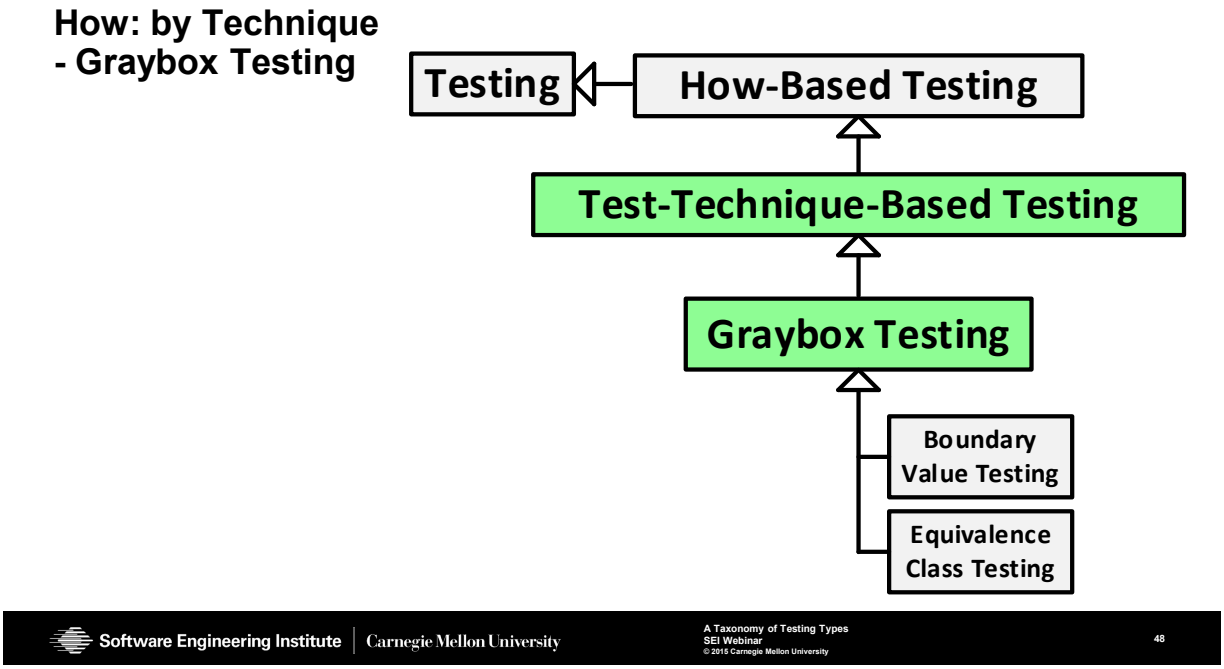
**047 So, taking a look at the black box testing, you can use cause and effect graphing. You can do combinatorial testing or in-way testing or A/B testing to minimize the amount of test cases. You can sort of do end to end, emission threat type testing. You can develop your test cases based on requirements, on scenarios. You can test to make sure that your code matches various syntax requirements. You see this very often in open system architectures where you have to meet certain protocols.

You can use classification trees to come up with equivalence class testing, decision tables. You can try testing your manual procedures to make sure that they work properly. You can do risk based testing, state based testing, user interface

navigation testing for, say, websites. There's a whole slew of these.

What they all have in common is they're basically black box. And so, you're essentially treating the system as a black box. You're only dealing with its interfaces, not its internals. And actually, not just the system, it could be subsystem. It could be a software component. It could be a unit.

How: by Technique - Graybox Testing

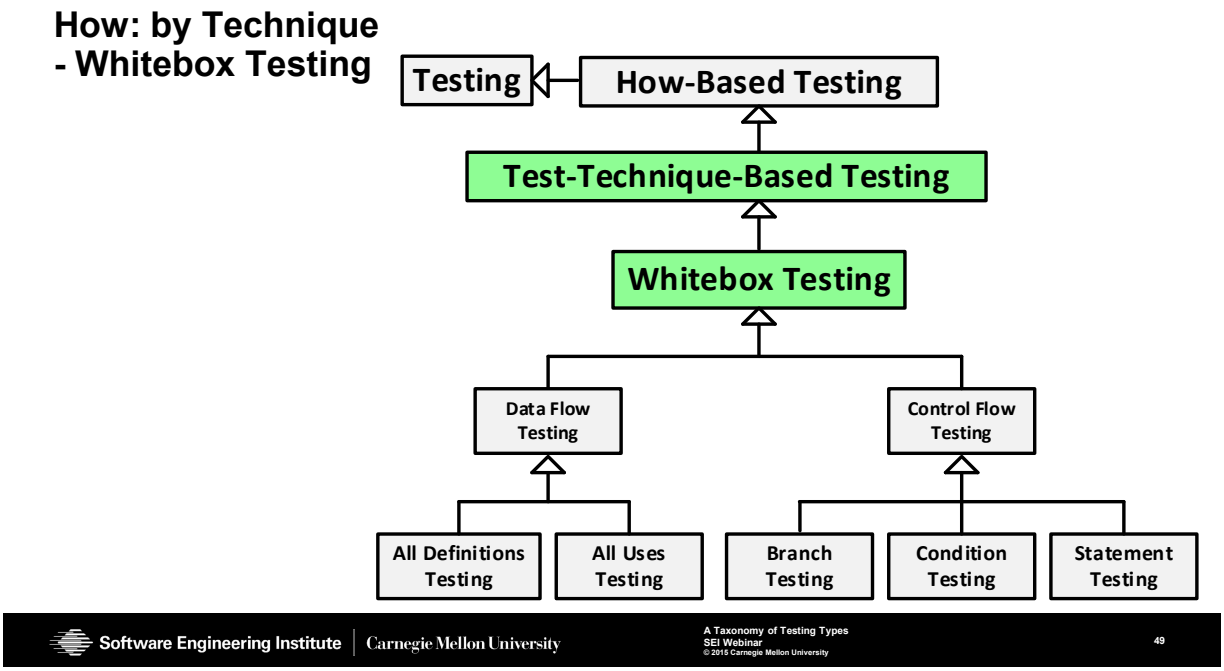


**048 When it comes to gray box testing, equivalence class and boundary value testing tend to fall into this category. They can, depending on how they're used, be done in either white box or black box fashion. Usually, people do both of them as well.

And depending on whether or not your test-- what you're testing, your object under test, is say safety critical or security critical or mission critical, you may do a more complete level of testing than you might do if it's just sort of the run of the mill average code that you're testing.

And so, for example, with equivalence class testing, you might only pick one test case per equivalence class. Boundary value testing, you'd pick that one test case. But you'd also pick a couple test cases just inside the boundary, a couple test cases on the boundary, and a couple test cases outside of the boundary. So, you're ending up with, say, seven test cases instead of one. And that's why boundary value testing is a more complete type of test. And also because you're dealing with the boundaries, that's where the defects tend to hide anyway. And so, you also tend to find a lot of defects with boundary value testing that you don't find with equivalence class testing.

How: by Technique - Whitebox Testing

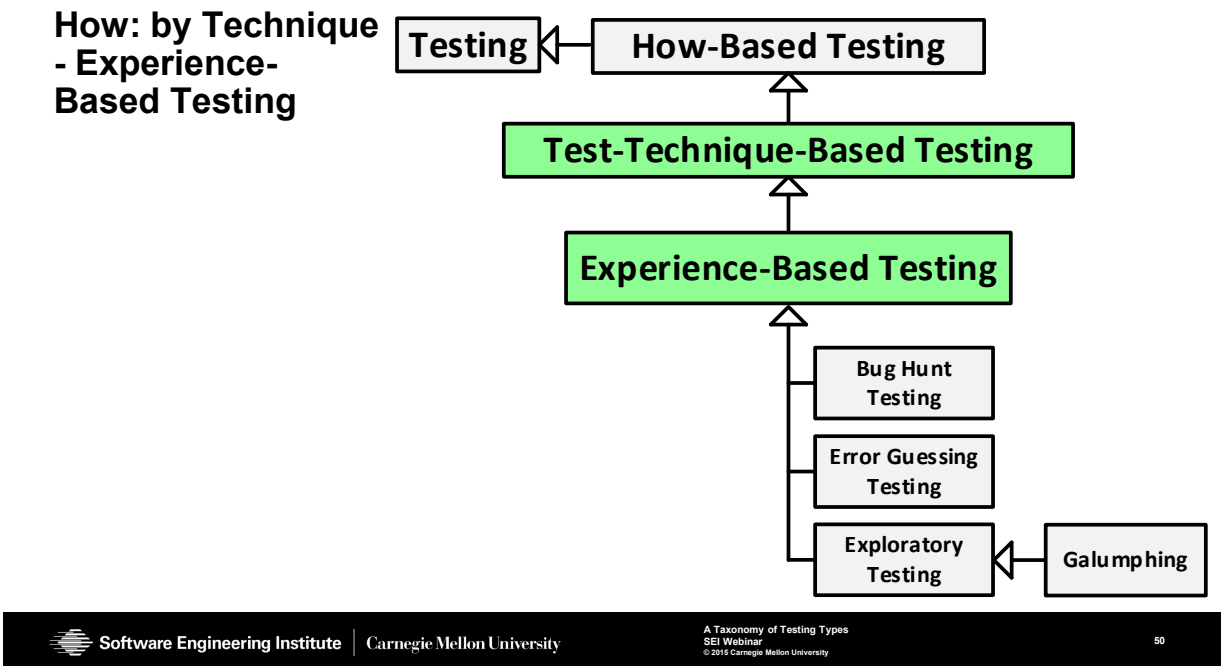


**049 White box testing, we tend to think in terms of control flow testing, so branch, conditions, statement testing. And in fact, very often, we have code coverage requirements. So, we have to have branch coverage or condition coverage or statement coverage. Those go hand in hand with these control flow types of testing.

On the other hand, something that not a lot of people think about is we can have data flow testing as well as the control flow testing. We can take a look and test to make sure that every data item is properly defined, and also make sure that every data items is used properly and take a look at how the data flows through the system. This can also be interesting from a security testing standpoint as you see how the data

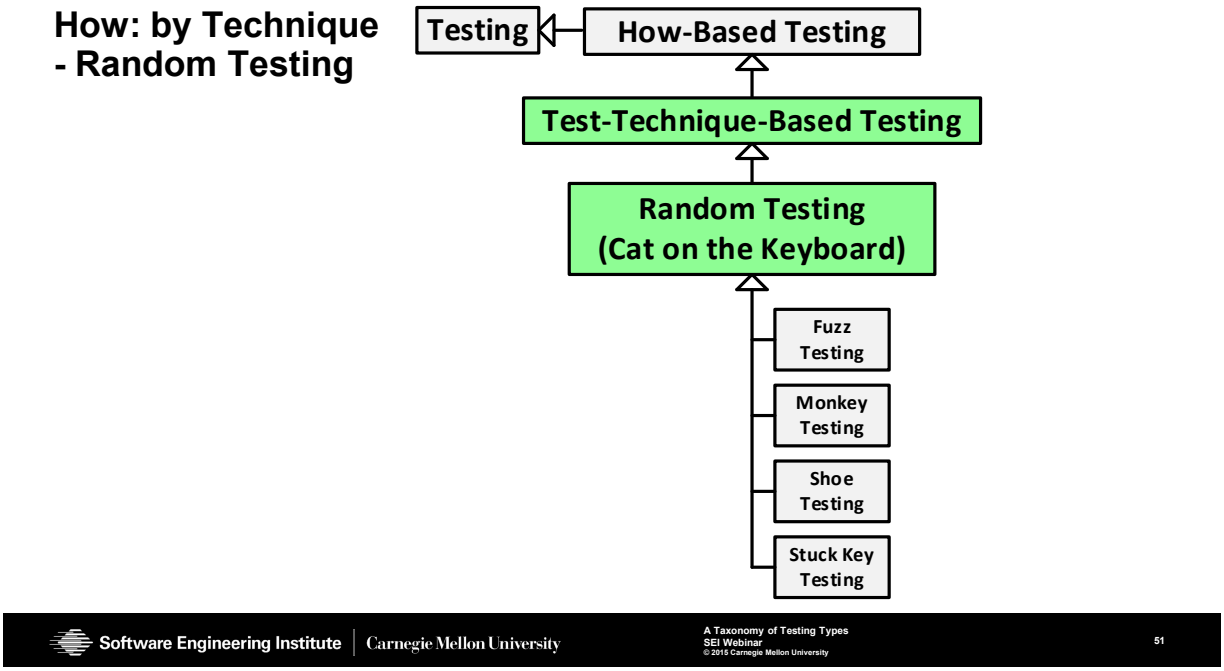
flows into your system, flows across the system, becomes encrypted, and later on becomes decrypted, things of that nature.

How: by Technique - Experience- Based Testing



**050 Experience based testing, these tend to be manual types of testing. I already mentioned exploratory testing, error guessing, essentially you use your past experience thinking where bugs have hidden in the past and try to elicit the same kinds of defects in the future. You can have bug hunts as test events. There's even galumphing, which is kind of an interesting example where you basically are throwing test inputs at the system that shouldn't have an impact, that shouldn't cause the system to do something unusual at all on the off chance that they do.

How: by Technique - Random Testing



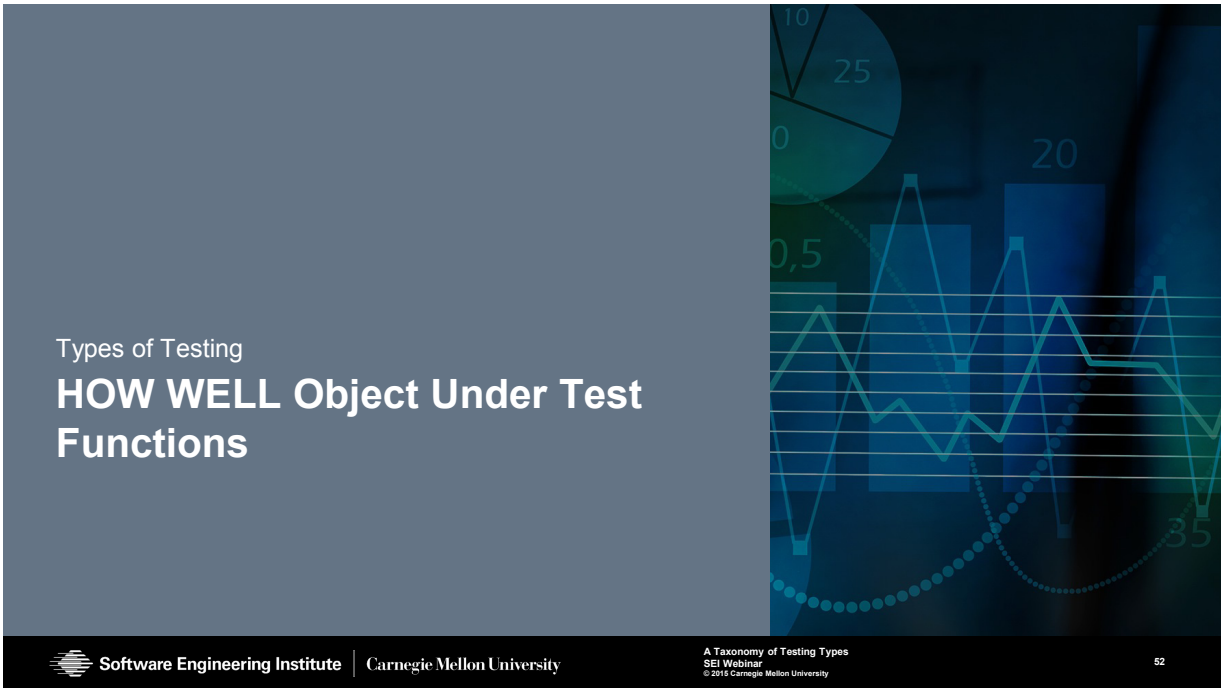
**051 And then, I especially like these. I don't spend a lot of time doing them, but I always do at least some of them. There's fuzz testing and monkey testing. They are often used as synonyms. But they're not actually synonyms. One throws random data at the system to make sure that it handles data properly. The other one throws random sequences of commands, of transactions, of inputs and sees whether or not the systems handle those properly. When the random inputs, whether they're commands or data, are easy to perform, then it's not a bad thing to throw a bunch at them if they're easy to perform and easy to evaluate. The problem, though, is since they're random, a lot of the test cases may very well test exactly the same thing, the same path, and so on. So, you're not being very effective.

And what you really want is you want every-- a good test case is one that find defects. In fact, a really good test case is on that finds lots of defects. And if you're throwing a bunch of equivalent test cases at the system, you're not going to find very many defects per test case.

Shoe testing is an interesting one, sometimes referred to as book testing. Just I forget exactly who the tester was. But essentially, the idea is take your shoe off, set it on the keyboard, and now you're pressing several random keys simultaneously all the time. Does the system handle that kind of input, or does it crash? And we know, in the real world, you could knock your coffee cup onto your keyboard. You could knock a book onto your keyboard. You don't want the system to crash when that happens.

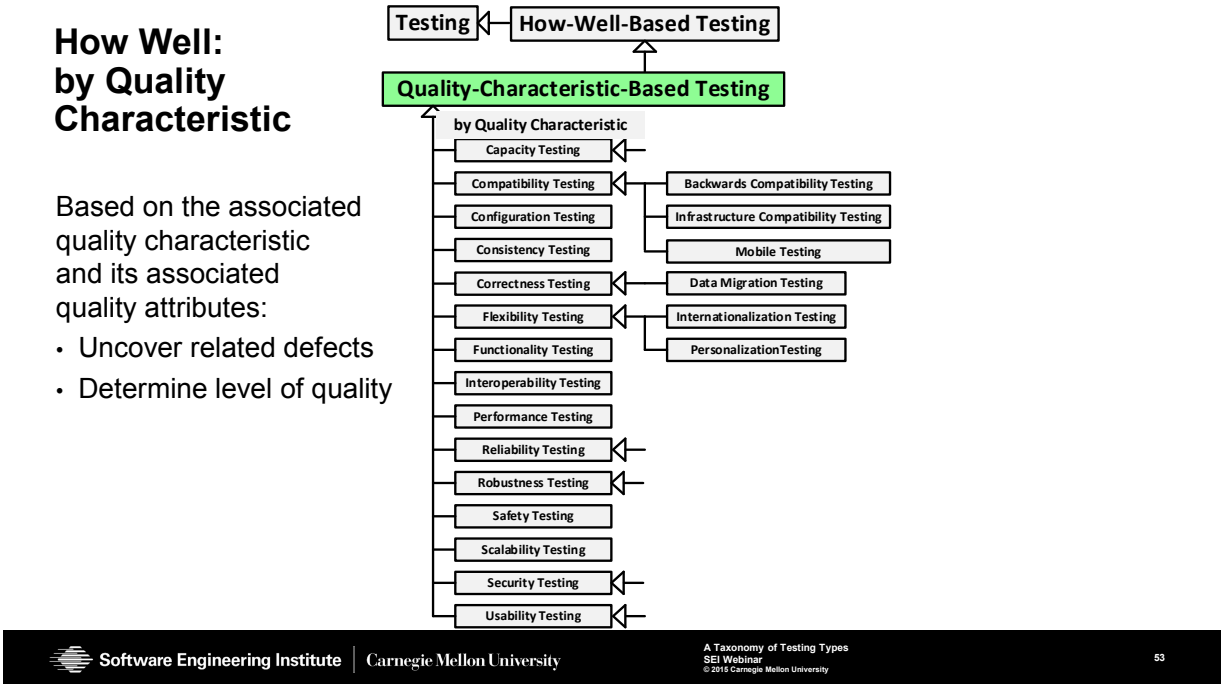
And then there's the stuck key one. We all know the keys on keyboards get stuck. So, just pick a couple keys at random, hold the key down for a while and see whether or not the system goes belly up, random testing.

Types of Testing



**052 Okay, this is the last section that I'm going to be covering over the last eight minutes or so. Here, I'm not talking so much about the tests, per se, but I'm talking about the object under test and how well the object under test has to perform. So, what I'm really talking about here--

How Well: by Quality Characteristic



**053 Is quality. And I'm talking about quality not in terms of number of defects per thousand lines of code. I'm talking about the quality characteristics and their associated quality attributes. So, I'm talking about the -ilities, to pass the safety, security, reliability, robustness, availability, performance, and so on. As you can see from the picture there, there's a lot of them.

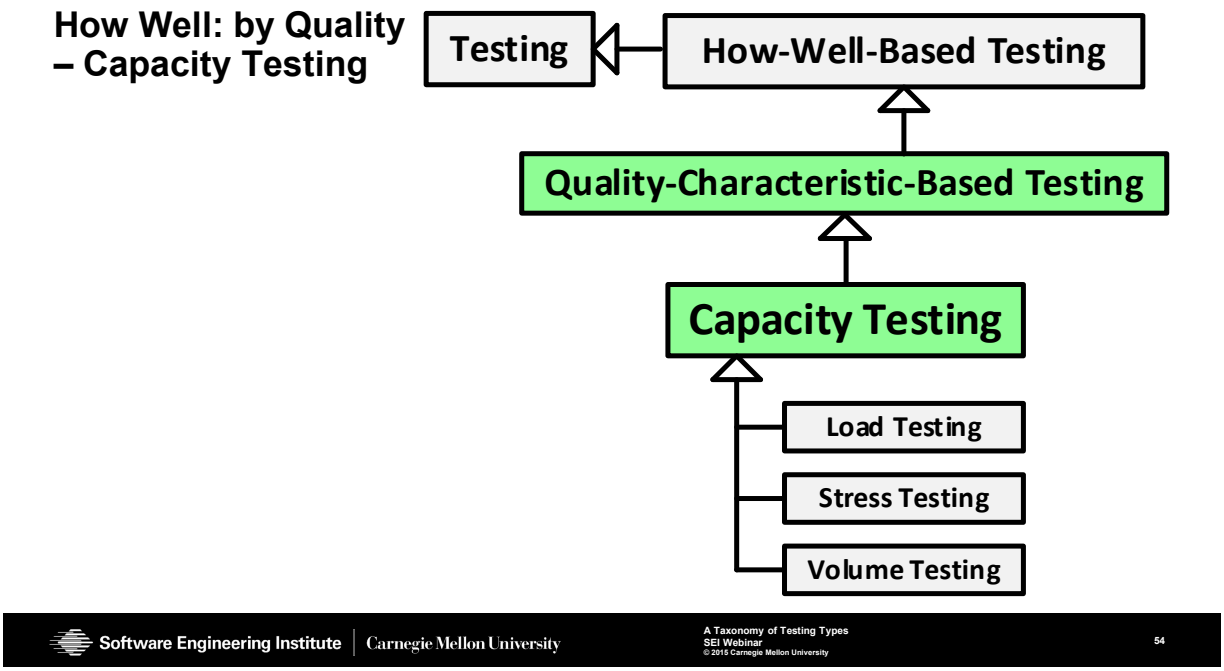
And each one of those-- well, not all of them, but many of them subsets underneath them. So, for example, for compatibility, you could be looking at-- let's see if I can even read that. Oh yeah, backwards compatibility testing to make sure that the new version of your system is backwardly compatible with the old one. You can have infrastructure compatibility testing. In other words,

can I support this application to multiple environments, multiple operating systems, middleware? And does it still work?

Mobile testing is actually the really big one here. You have a zillion mobile devices out there. You've got your smart phones, your tablets, your laptops, and so on. And you have the one interesting thing I would mention here is most people make the mistake of when they're trying to test on a mobile device, they'll buy a brand new mobile device right out of the box. And they'll test on it. Well, in the real world, your users have the mobile device with maybe not just your application on it. They have fifteen or twenty or thirty other applications running on it. And the thing is six months, nine months old and its battery not holding as much of a charge. You get very different behavior when you're testing a mobile device in the real world, a real mobile device, as opposed to a fresh one out of the box.

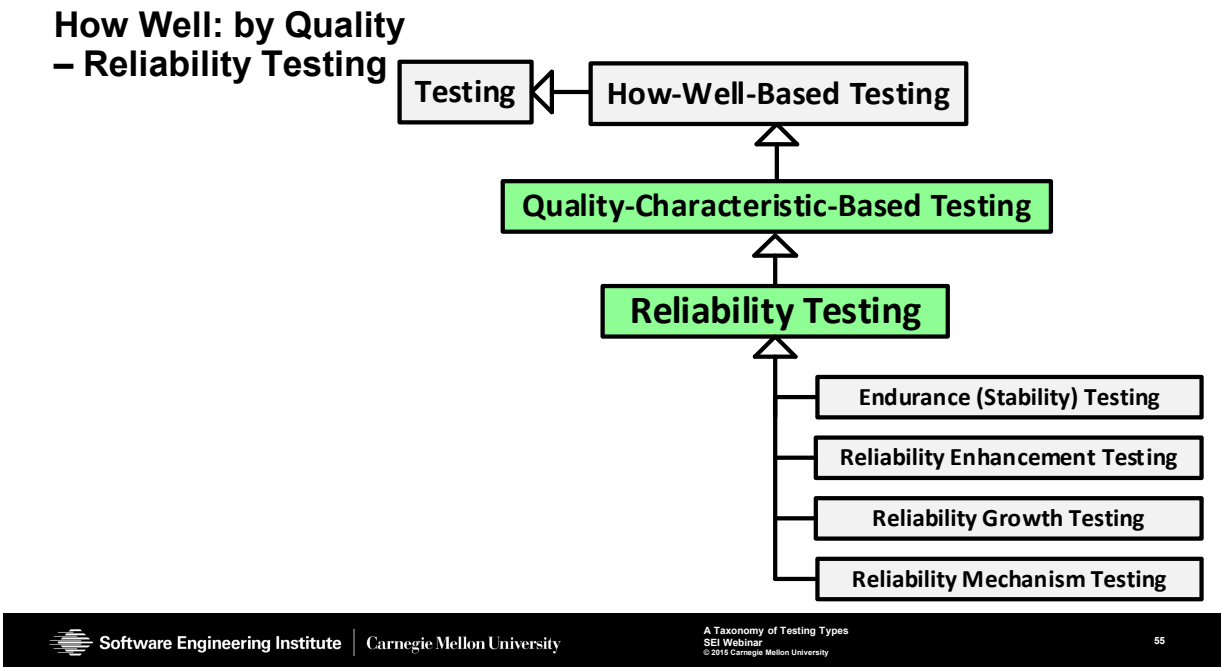
And I think that'll probably do it for this slide.

How Well: by Quality – Capacity Testing



**054 Let's move on to some of the specific ones. When I'm trying to make sure that the system meets its capacity requirements, I can use load testing to make sure that it handles the expected load. I can use stress testing to make sure that it handles above the expected load, sort of the maximum possible load, if you will. And I can do volume testing where I make sure, for example, that it holds really large amounts of data, so big data type testing.

How Well: by Quality – Reliability Testing

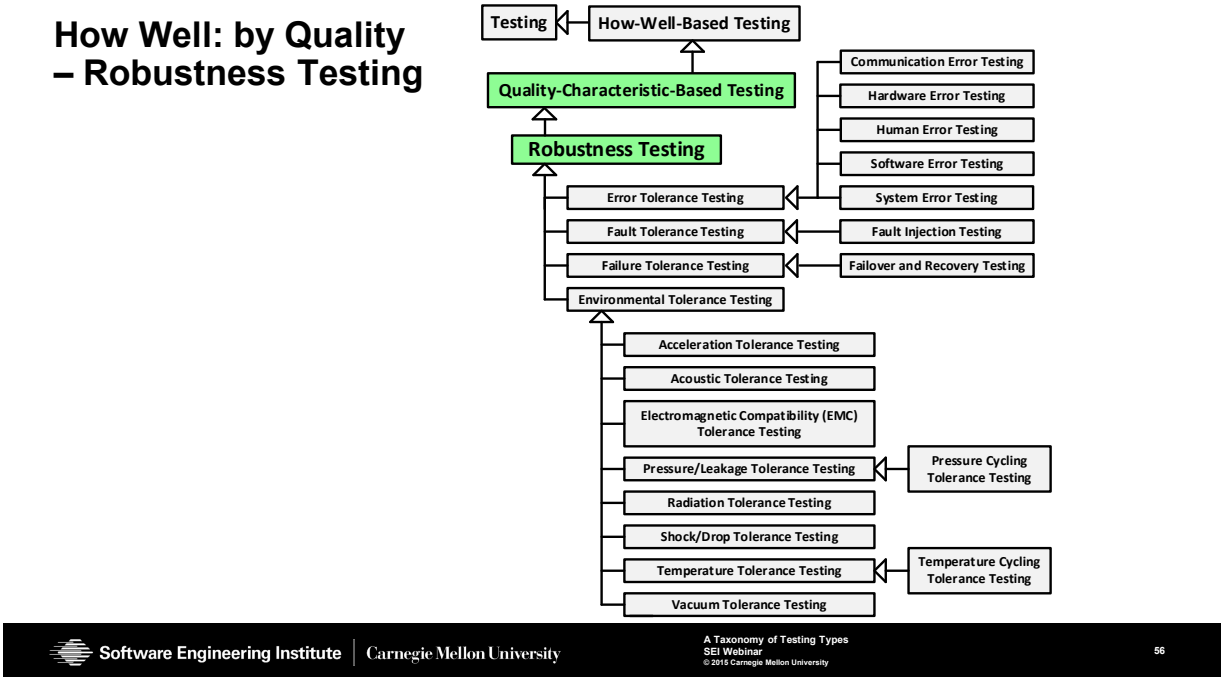


**055 Reliability testing, here there are several different kinds. Endurance testing or stability testing, I have something run for a long time to make sure that we don't have things like memory leakage or something of that nature, signal drift, something that stops it from working after eight or ten or two days, or however many hours.

You can check to see, with reliability enhancement testing, how more reliable each version of the system is than the previous one, very similar to the reliability growth testing to make sure that the reliability is growing as predicted. And there are various architectural mechanisms that you might put into the system to make the system more reliable. So, you can do reliability mechanism testing where you're testing each one of

those reliability mechanisms to make sure that that mechanism is behaving properly.

How Well: by Quality – Robustness Testing



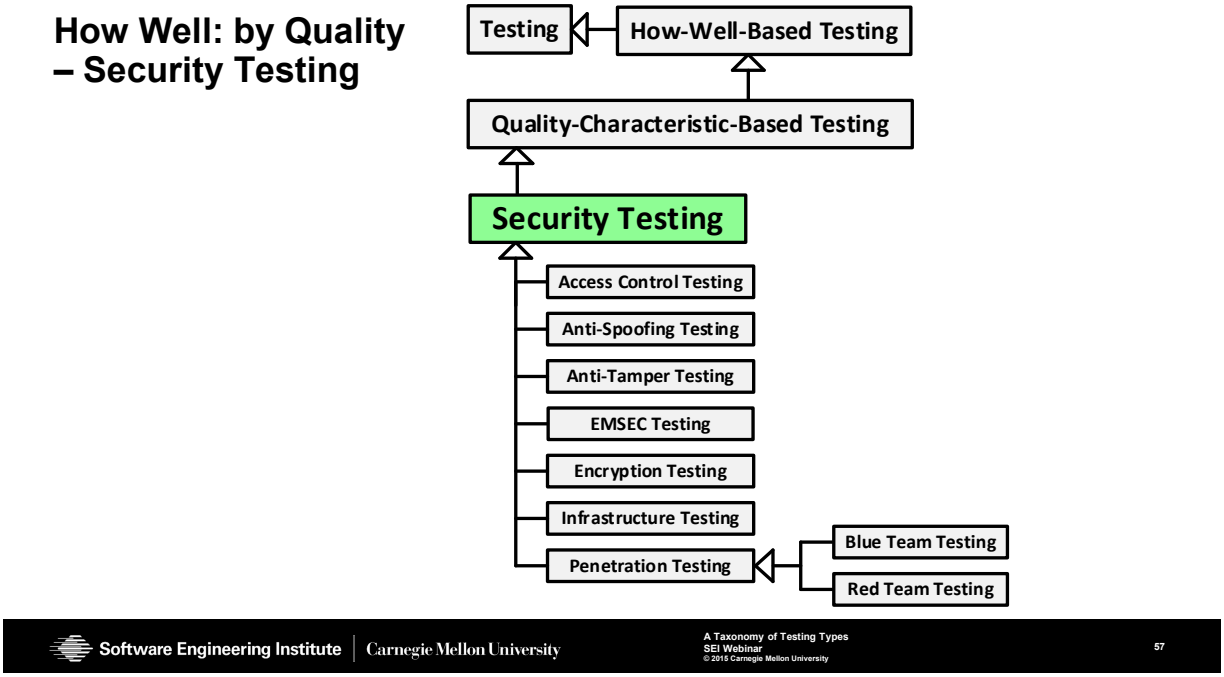
**056 When it comes to robustness, now I'm talking about does the system behave right when something wrong happens, during rainy day situations. So, we're talking here, if you look right under the robustness testing there, there are four boxes. There's error tolerance, in other words, does the system handle bad input, whether it's from a human or a sensor or some outside system. Fault tolerance, does the system handle being in a fault state, having an incorrect mode or state or some value stored inside of it incorrect? Does it handle that properly?

Or does it handle failure properly? If the system actually does fail, does it

do, say, a hot, warm, or cold failover? Does it restore itself to a proper pre-failure state? And so, those are the ones that software people tend to think a lot about, those three and the subtypes up above them.

But if we're dealing with systems, we have to worry not only about those levels of software robustness. We also have to worry about how robust the system is in terms of the actual physical environment that it's in. and so, you have all sorts of things, shock tests, vibrations tests, heat tests, radiation, shock and drop, vacuum testing for satellites. Once again, there's a huge number of these. They're very specific to different kinds of application domains. But definitely, if you're working on a system, as opposed to pure software, you need to worry about those kinds of testing, too.

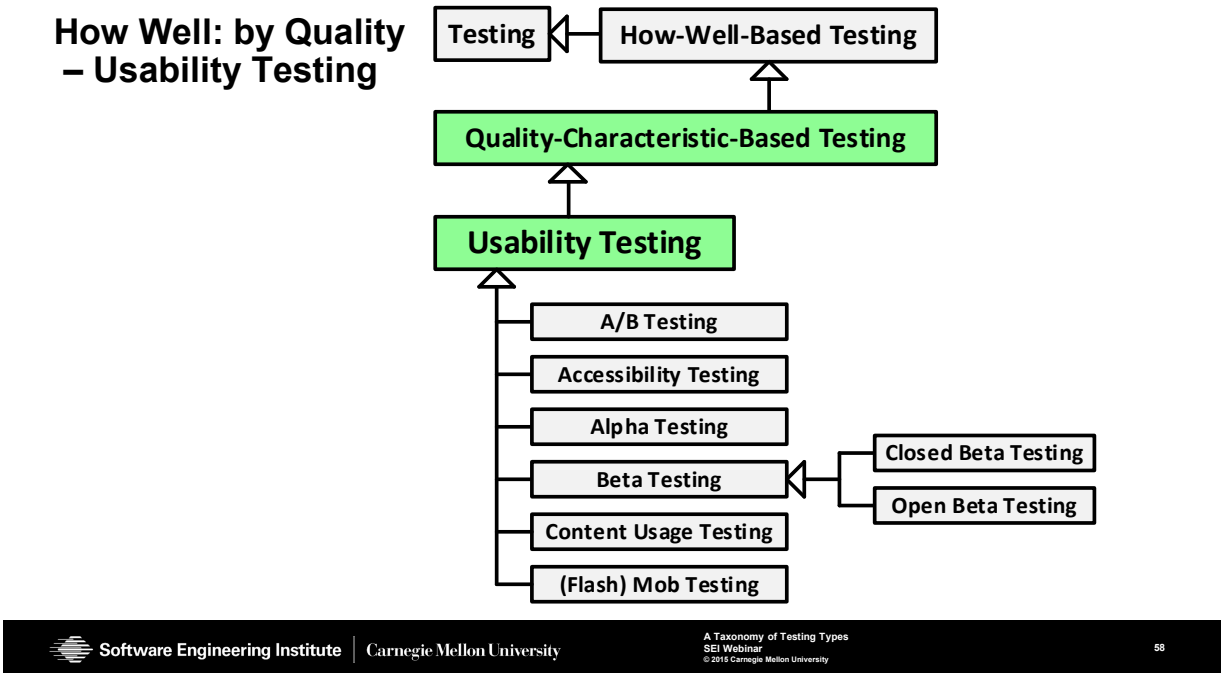
How Well: by Quality – Security Testing



**057 Security testing, there's actually a surprising amount of this. Everyone tends to think in terms of penetration testing when they think of security testing. Well, you may want to be testing access control mechanisms, make sure that they work properly, make sure that the anti-spoofing mechanisms work properly, anti-tamper, which is either tamper resistance or tamper evidence kinds of things. Emission security, if you're dealing with certain things where you're dealing with classified data or any very sensitive data, you have to worry about is your system emitting electromagnetic radiation that someone else can pick up and decrypt and figure out-- analyze and figure out what that secret data is. You can make sure that the encryption is working properly.

There's security infrastructure issues. Interesting thing in penetration testing, there's the blue team testing where you basically are testing the system as the people who are trying to protect the system. And then there's the red team testing where you are the black hat trying to attack the system. And very often, you'll have both teams working together on a penetration test.

How Well: by Quality – Usability Testing



**058 And the last one here, usability testing, here's where we're working on the human factors engineering kinds of things. You have A/B testing where you're essentially taking two versions of the system and having two different groups of users try the two different versions and see which one they like best, or are most effective using, or they make the fewest errors using. In fact,

that generalizes to in-way testing where it's not just two-- A/B is sort of two way. You could have three and more.

You have accessibility testing. So, does your system work well for someone who's got hearing problems or vision problems, things of that nature. Alpha testing, beta testing is different kinds of user oriented testing that I mentioned before. Content usage testing, this is often used for websites to see what parts of the system people are actually using and how much. And then sort of the flash mob testing of just grabbing a whole bunch of people together and bringing them together for a big testing of that.

Conclusion

Conclusion

Most systems require quite a few different types of testing.

Most testers are not aware of the majority of the different types of testing.

- If you are not aware that it exists, then you don't know whether you need it.

These types of testing can be organized into a taxonomy by the 5W + 2H questions.

This taxonomy has several uses:

- Ensure the test strategy is sufficiently complete with no important type of testing overlooked.
- Organize testing types to make them and their relationships more understandable.
- Augment test training materials.
- Help categorize and understand limitations of testing tools.

**060 So, this is our last slide, three minutes left to go. So, actually this is

a software project that might actually end on time. Okay so, what do I want you going away remembering? First of all, most systems require quite a few different kinds of testing, probably more different types of testing than you're used to performing. I'm sure that most of you heard at least a few types of testing that you either weren't aware of yet, or you might have heard of but you didn't know exactly what they meant. These kinds of tests, the nice thing about them is it's not just this big huge amorphous blob of different kinds of testing types. You can actually categorize them, organize them, put them in a structure which makes it easier for you to understand them, think about them, communicate with other people about them, and so on.

And so, essentially what you can do now is you can take what I've presented here. And you can take a look at your testing planning and see whether or not you think it's adequately complete. I don't expect you to include all of these in your test strategies. But I would think it would be useful to take a look at them all and decide whether or not it makes sense. It should be a conscious decision to leave something out. It shouldn't just be you left it out because you weren't aware that it even existed.

And this is something that might help you augment your training materials. What you could do is, just like for example when I was learning pattern

languages for the first time, design pattern languages, back in the old OO days, I'd have one of the patterns books. And every day, I'd pick a new pattern and try to get up to speed on that. That's the kind of thing that you can do with this. You could pick one of these every day or every week. And in your copious free time, go out and Google it and read four, five, or six articles on it, and get up to speed on it.

Okay, and this also helps you deal with test tools. One of the things the test tool vendors like to say is their tool will do everything. Well, one of the things you can now ask is does it handle this kind of testing, does it handle that kind of testing.

Q&A

**061 And you-- the rest of the one minute we have--

Shane McGraw: Yeah, we've got about a minute left. And we are going to try to stop on time. So, we'll get one question for Don. Before we get that question, I just want to invite everyone to the Software Solutions conference. The SEI is hosting a new conference November 116th to 18th. That's going to take place at the Hilton Crystal City in Arlington, Virginia. So, if you like talks like today, Don's on testing, Don will actually be there giving a talk on testing as well. There will be talks on legacy systems, Agile and government complexity, high shore and software acquisition, cloud computing, DevOps. It'll be a great conference. So, we look for as many people as possible to attend. So, we'll send out an email tomorrow with information on that conferences, also a registration code.

So, let's get to some questions for Don. We'll take on here. And then we'll wrap it up, Don. So, again, a lot of comments too about where to find a recording or the PowerPoint slides. The slides are available now before you log off in the files tab. You can walk away with a PDF copy of them, and fill out the survey upon exiting. Everything is being archived. That'll come out in the email probably tomorrow.

Donald Firesmith: Actually, on a similar note, I'm in the process right now of producing a wiki that has all of this taxonomy in it including definitions and applicability and recommendations and so on. It will

probably take a couple months before I have that wiki up and available for people to access on the SEI website. I'm doing a similar one with my testing pitfalls. So, keep looking at our website. You'll see things.

Shane McGraw: And folks, we apologize, there are a lot of questions we're not getting to. So, feel free to reply to the email you get tomorrow, contact Don directly. I'm sure he wouldn't mind. But let's ask one more from Dewheat asking will this testing approach you're discussing apply almost equally to maintenance efforts.

Donald Firesmith: Very good point. By maintenance, I'm assuming what you mean is not major new increments, but bug fixes and small little issues. Different test types will make more sense under those circumstances than they would when you're developing the whole thing. So, for example, if I'm making a small change, I want to run my regression tests on that change at the very least. If I'm adding new things, I'm going to want new unit tests on it. I may want some integration tests and so on.

A lot of it depends also on how much has been automated. If you have tons of it automated already, then you've got-- it makes a lot of sense to reuse that when you're making changes during maintenance. If you don't have an awful lot, then often you don't have the time to generate

a lot of new stuff. So, you're stuck more concentrating on just those changes that you've made. But once again, realize that if I have a defect in the software change that I've made. It may show up locally. But it may have ramifications and show up someplace else. And that's why it's very important to do a lot of local testing, unit testing, upfront because there it's easier to localize where the problem is.

Shane McGraw: Don, great presentation. You are wealth of testing knowledge. Again folks, that's all the time we have today. Sorry for the questions we didn't get in the queue. So, feel free to continue the conversation after the event and you get the email tomorrow. And again, thanks for your participation today. Have a great day.

Carnegie Mellon University

Carnegie Mellon University

This video and all related information and materials (“materials”) are owned by Carnegie Mellon University. These materials are provided on an “as-is” “as available” basis without any warranties and solely for your personal viewing and use.

You agree that Carnegie Mellon is not liable with respect to any materials received by you as a result of viewing the video, or using referenced websites, and/or for any consequences or the use by you of such materials.

By viewing, downloading, and/or using this video and related materials, you agree that you have read and agree to our terms of use (www.sei.cmu.edu/legal/).

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Copyright 2015 Carnegie Mellon University

Copyright 2015 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0002727