**Nicola Tavares:** Welcome to the ITS Standards Training.

**Ken Leonard**: ITS Standards can make your life easier. Your procurements will go more smoothly and you'll encourage competition, but only if you know how to write them into your specifications and test them. This module is one in a series that covers practical applications for acquiring and testing standards-based ITS systems. I'm Ken Leonard, the director of the U.S. Department of Transportation's Intelligent Transportation System's Joint Program Office. Welcome to our ITS Standards training program. We're pleased to be working with our partner, the Institute of Transportation Engineers, to deliver this approach to training that combines web-based modules with instructor interaction to bring the latest in ITS learning to busy professionals like yourself. This combined approach allows interested professionals to schedule training at your convenience without the need to travel. After you complete this training we'd hope that you'll tell your colleagues and customers about the latest ITS Standards and encourage them to take advantage of these training modules as well as archived webinars. ITS Standards training is one of the first offerings of our updated professional capacity training program. Through the PCB program we prepare professionals to adopt proven and emerging ITS technologies that will make surface transportations safer, smarter and greener. You can find information on additional modules and training programs on our website at www.pcb.its.dot.gov. Please help us make even more improvements to our training modules through the evaluation process. We look forward to hearing your comments and thank you again for participating and we hope you find this module helpful.

**Nicola Tavares:** Throughout the presentation this activity slide will appear indicating there is a multiple choice pop quiz following this slide. The presentation lecture will pause at each quiz section to allow you to use your computer mouse to select your answer. Selecting the "Submit" button will record your answer and the "Clear" button will remove your answer if you wish to select another answer. You'll receive instant feedback on your answer choice. Please help us to make even more improvements to our training modules by completing the post-course feedback form. This module is T315 Applying Your Test Plan to the NTCIP 1202 ASC Standard. Your instructor Ken Vaughn received his bachelor's degree from Tulane University and his master's degree from Texas AMU. He coordinated traffic signals in Los Angeles County in the early 90s and he has graduate-research studied the performance of actuated signals in coordinated systems. In 1994 he became involved in the ITS Standards and was a founding member of the NTCIP joint committee in 1995. Ken Vaughn is also president of Trevilon, LLC.

**Ken Vaughn:** Hi, this is Ken Vaughn and we are gonna be talking about how to apply your test plan to actuated traffic signals controllers today. Our target audience for this

course are engineering staff. These are the people that will be procuring [ph?] the system, be responsible for its delivery. Operations and maintenance staff, of course, will have to operate and maintain the equipment and from central system and also out in the field. Systems integrators as well 'cause, of course, they will be procuring equipment and making sure that that equipment meets specifications before they integrate it into their system. Device manufacturers, of course, they will have to make sure that their devices meet the specifications and the testing process. And, finally, testing contractors who will perform the tests. And, of course, "testing contractors" that can also mean just internal staff. It doesn't necessarily have to be an external contractor. Some of the prerequisites for this course include T101, Introduction to ITS Standards Testing and T201, How to Write a Test Plan, followed by T202, Overview of Test Design Specifications, Test Cases and Test Procedures. Those three courses provide the background of testing, how to write test plans, how to document things and just the purpose of testing, the stages of testing, phases of testing. We'll reference some of those ideas during this course and it'll be useful for you to have that understanding before you take this course. In addition, C101, communication protocols and their uses in ITS applications, defines the lower layer of protocols, SNMP and such, that will be used during testing. Once again, it will be very useful for you to understand those issues before you take this particular module. And, of course, A315a and A315b, the user needs is A Section requirements for the B Section for actuated traffic signal controllers based on NTCIP 1202 standard. Of course, we're testing two requirements and those requirements are based on user needs. So understanding those two modules before taking this course will also be useful. All of those courses link back-to-back and lead up to this course, which is "Applying Your Test Plan to the NTCIP 1202 ASC Standard". This is the last in this particular course series. The learning objectives for this course include recognizing the importance of testing ASCs, applying the rules and developing an ASC test plan and listing the rules for developing test case specifications and procedures, and we'll apply those to that section as well. And then, finally, in Learning Objective Number Four, "Developing a Sample Test Case: Specifications and Procedures", followed by "Understanding Test Results for NTCIP 1202".

**Ken Vaughn**: So we'll get right into it with Learning Objective Number 1: "Recognizing the Importance of Testing ASCs". We'll discuss several points within in this section: the safety implications of signals controllers, the complexity issues related to signals, functional versus communications testing, the importance or repeatability within testing and, finally, budgeting for testing.

**Ken Vaughn:** So one of the things we'll talk about are the safety implications. The ASC is relatively unique for NTCIP in that it directly controls conflicting traffic streams. So not only does it control traffic, but also conflicting streams compared to, say, a ramp meter that only controls parallel streams. ASCs control conflicting streams. Now some of the safety will never result in an unsafe operation, presumably because you have a conflict monitor

that provides a base level of safety. However, improper timing can still result in less safe operation ‘cause poor timing can increase crash rates. For example, you may have yellow times that are too short or you may just have very short green times that frustrate drivers. So improper implementation of NTCIP can result in undesired timing that may result in some safety implications. Once again, not resulting in unsafe operation, but perhaps less safe than ideal. The other thing is, they have complex device, even quite a bit more complex than the other NTCIP devices, in fact. The complex logic increases the potential for errors, which is why testing is important. There’s also quite a few very complex features within the ASC and these features a lot of times continually interact with each other further complicating the device. So it really is important to test the ASC even more than a lot of other devices. Also, time sensitivity complicates interactions. Regardless of what’s happening in the background, that signal controller has to make sure that it’s properly timing the signal indications on the street. So those have to be very, very time sensitive, and that also complicates the design of the ASC and makes sure that we need to test that to make sure it’s operating properly. The way you do this is to perform your testing in steps: You isolate the features and then you test them. And then once you’re confident that the isolated features are working properly, then you can combine features together and make sure that they all operate properly together. So, combine proven test features and test again.

**Ken Vaughn**: Now there’s also an issue related to functional versus communications testing. NTCIP defines, in fact, both your communications, i.e., insuring the yellow change interval is set to the desired value from the controller, and the functional aspects, i.e., making sure that yellow change interval is timed and displayed appropriately on the signal head. Now, technically, NTCIP Standard defines both of these, some of which by reference to NEMA TS2, but both of them are defined within the NTCIP document. However, it may be appropriate to separate out these tests to simply make sure initially that when you set the yellow change interval that it changes in the database, then have a separate set of tests making sure that timing is appropriate according to the database. So that is left up to you to decide, but there are those two issues and, of course, for the controller to work properly you have to have both of those working properly.

**Ken Vaughn:** The other aspect of testing is repeatability. It’s very critical, particularly with a device this complex, that you develop proper documentation so you can ensure repeatable tests. Unexpected operation may occur to all sorts of reasons, implementation issues, actually, the device not performing correctly. Tester issues, in other words, the person testing performs a step incorrectly. Or there may be actually an error in the test procedure he’s following and that may be an issue. A problem may also be in the specification. You may have not required an item that is required for a particular feature you’re testing. Another possibility is the standard itself is ambiguous. So all of these areas could identify problems and if you document your test precisely, then it will help you make sure you understand where the error is, whether you read through, you test it, a problem

occurs. Typically, you can very quickly identify the source of a problem in one of these five areas.

**Ken Vaughn:** Another issue to be concerned about is the budgeting. A lot of people end up not considering budgeting for this testing. You need to understand that testing is designed to reveal problems before you reach the field. And when you think about it in that sense, you should really almost be expecting the device will fail the first round. You may have to perform multiple rounds of testing in order to get it working properly. Even mainstream computer applications experience bugs. So you should not be surprised to find some issues within the delivered equipment. And these errors should be corrected before deployment. It's a lot better to identify them before you deploy them, get them fixed, and then deploy rather than having to retrofit your system.

**Ken Vaughn:** Finally, devices should be retested after any software update and making sure that change in software did not create its own problems. Full testing is likely to require multiple rounds of testing because of that. So you go through a round of tests, you identify some problems, you send it back to the manufacturer, they send back a fix, you have to re-test it, and maybe that goes on two or three times. All that has budget impacts and schedule impacts on your project. All this needs to be considered up front.

**Ken Vaughn:** Well, that brings us to our first activity of the day and the question is "Why is it important to test ASCs?" Option A is implementation errors might result in conflicting green displays. Option B: Implementation errors might decrease traveler safety. Option C: Testing is not important for ASCs. And Option D: it gives peace of mind for a trivial cost. So go ahead and make your selection and we'll discuss those in a second.

**Ken Vaughn:** The correct answer, of course, is Option B: An error can result in an undesired configuration and decrease safety. Now it does not decrease safety so much you present conflicting green displays because the conflict monitor will insure that conflicting displays are never shown. There's also lots of safety built in within the NTCIP, but even if there's an error in there, the conflict monitor will prevent those conflicting displays. Option C is not correct because testing is especially important for ASCs due to their abilities to directly control traffic operations. And, finally, Option D is incorrect because testing will require significant effort, significant costs, but it is very important to do nonetheless.

**Ken Vaughn:** Well, that completes Learning Objective Number 1, "Recognizing the Importance of Testing ASCs". The safety implications-- we talked about complexity issues, functional versus communications testing, repeatability and budgeting were all issues we talked about in that particular section.

**Ken Vaughn:** We'll now get into developing a sample test plan and really kind of putting what we've learned into practice. We'll identify some of the requirements for testing each phase. We'll identify test methodology and which ones are appropriate to NTCIP. We'll describe the requirements to test case traceability matrix and plan the logistics of testing. We'll also estimate level of effort for testing, evaluate the risks, and then look at large project close-out. Test documentation is defined in IEEE 829. It defines how you go about writing your test documents, the rough outlines for these documents. The first document is a test plan and that's discussed in detail in module T201, which was one of the prerequisites for this course. The sample test plan does identify the requirements that are to be tested and, of course, we talked about these requirements in module A315b and that discussed how to define those requirements and also develop a list of sample requirements. You can see the Participant Student Supplement for a list of those requirements that we developed. Every requirement should be tested during at least one test phase, using at least one method, and by at least one party. Of course, you can test requirement multiple times, but every requirement should be tested. Now that phase, it may be early on in the project, during acceptance, or it can be late, during actual implementation, site integration. One test method, we'll talk about that-- by at least one party. That can be the manufacturer, it can be a third party tester, it can be the agency itself, but someone needs to test those. The extent of agency testing is a risk management issue. So a lot of times agencies will require the manufacturer to do the bulk of the testing, but the agency should still do at least some of the testing; how much is a risk management issue.

**Ken Vaughn:** Identifying the test phase: Well, we talked about the various test phases in previous modules. For NTCIP, we're mainly focused on factory acceptance, incoming device and site acceptance. Perhaps some devices, if you're testing a new standard and not this particular standard, but a new standard, you may want to test during prototype or design approval stages. But, generally, for signal controllers, you probably are focused on factory acceptance before any product is delivered to you or incoming device, so that every device gets tested as it arrives or site acceptance right before you make it active. Burn-in testing is not very commonly used with NTCIP on the simple basis that once the software's been designed and deployed within the system it's not gonna change. So factory acceptance, incoming device and site acceptance are the three main areas where NTCIP testing is performed. But even within those three areas, the testing is often divided to separate out things like NTCIP testing, hardware testing, environmental testing, all these other different aspects. And so within each one of those phases you may have different sub-tests.

**Ken Vaughn:** Once again, what-- there are four different types of test methodology that we discussed in previous modules. For NTCIP we're mainly focused on demonstration and formal testing with a particular focus on formal testing. And when you test you need to consider two basic scenarios: positive tests and negative tests. Positive tests being

those tests that make sure the device functions the way you expect it to. Negative tests are designed to make sure the device rejects requests that are invalid. So if I want to set my yellow change interval to zero seconds, that perhaps is probably not allowed by your specification. The device should come back with an error message, properly.

**Ken Vaughn:** So we'll look at the requirements to Test Case Traceability Matrix a little bit now. Once you've identified all of the requirements that we discussed in a previous module, then you start looking at each one of those requirements and identifying each specific test cases you want to perform. And then you start mapping those one against another. And oftentimes when you are considering both a positive test with a negative test, you'll come up with multiple test cases for a single requirement. So if you just look at, say, a requirement 2.2.1.1 on this slide, "Configure a Timing Pattern", we'll see that we have two test cases: "Configure a Valid Timing Pattern" and "Incorrectly Configure a Timing Pattern". You'd also develop other test cases, such as configuring an invalid timing pattern. So there's all sorts of scenarios you can dream up. You'll also see that as you go down, some of these in the Test Case Traceability Matrix may say, "See manufacturer acceptance test plan," because that task may have been left to manufacturer, not a part of the particular test plan that you're writing the specification for. For that manufacturer test plan, they would need to specify exactly which tests are being performed there.

**Ken Vaughn:** Another thing that goes into your test plan is the overall design of your test environment. Now this is a standard figure from NTCIP 8007, which is that test documentation standard. And it basically follows in order for signal controllers. The device under test would be typically your signal controller. The communications cloud, hopefully you keep it as simple as possible for your implementation. But somehow you have to connect that signal controller to your test application, which is at the bottom of the screen. You'll also notice that connected to that communications cloud is a data analyzer. That data analyzer is a third party, basically, that just monitors the communications that are occurring between the test application and the device under test. So those are the three major components to the testing along with communications between those three components. And, of course, a test environment will also have to consider where are you physically, what sort of protection from the elements you need and other issues as well--power requirements, etcetera, which is the logistics. Where will a test be performed? Are you gonna be on site or in a laboratory-type environment? If you're on site, how do you consider safety issues? Do you have to be wearing a vest, hard hats, other issues? Who's responsible for what? Who's responsible for providing power? How is that gonna be done? What tools might we need? Tables, protection from the elements, local assistance for remote testing. So, for example, with the amazing technology we have today we can do a lot of this testing remotely across the Internet. That complicates some things, but it makes less travel and other things on some of the people. But if you do that, oftentimes you still have to have someone locally to the device to do certain things to the device for certain test cases. So you may need some local assistance there.

**Ken Vaughn:** And also you have to consider what happens if testing is suspended. For example, as it happens, I was actually testing a device on 9-11. And, of course, when 9-11 occurred certain things went into mode and we had to suspend testing for a while. So you never know what's gonna happen. As it turns out in our case, we weren't testing in one of the actual affected areas, but if we were and we were testing a live sign on the street, then the obviously we would have to stop testing for a prolonged period. We may actually have to put equipment back into operation for the event. So all of that has to be considered if you're gonna be suspending testing, particularly if you're in a live environment.

**Ken Vaughn:** Developing a sample test plan, the other thing you have to do is estimate your level of effort. So estimate the effort, schedule and budget for preparing the actual test plan and test cases and procedures, all of your test documentation along with performing multiple rounds of testing, as we discussed. And that's investigating problems, preparing the test result documentation-- all of that needs to be considered for multiple rounds of testing. So it can be quite extensive. The other thing you need to do within your test plan is evaluate risks. What happens if the device does not pass? At what point do we just kind of throw up our hands and say, "Enough's enough. We're not gonna test anymore"? Also, if you go through multiple rounds of testing and your project is being delayed, are there impacts to the financial reward for the vendor? Is there a cost that he's paying for you to have to test so much? So that's another big, important factor. Who pays for the additional rounds of testing? And do external factors control the schedule? Are you planning-- is this project part of an effort to prepare for a major event? You know, is it the Olympics or something where start dates are certain and you have to have your equipment up and running by that date? That all impacts decisions you make on whether or not to accept a device at a particular time. All of those issues need to be discussed in your test plan up front so that if the situation occurs when you're running out of time, you know the plan you're gonna follow. Finally, what if it's unsatisfactory, you can't accept it? How do you return it? How do you get out of the contract? All of those things need to be addressed within your test plan. And, finally, how are disputes resolved, particularly if the problem is with a specification or the standard? How do you resolve those disputes so that you are still contractually proper and still get what you need?

**Ken Vaughn:** Also, we'll talk about the impact of a failure. So the testing you can very often-- will find an anomaly in your device that means it's not meeting requirements. If it's not meeting the requirement, how does that impact the overall operation of the device? As we discussed, a requirement can be traced to a lot of different user needs. So the traceability tables, while they take time to prepare, once you get to the testing phase with the project they become very, very useful, because you go directly from the test case back to the individual requirement, identify the exact text of that requirement, identify whether or not this is a true problem or there's an error with the way it was tested or something else. If it is a true problem, then you can trace that requirement all the way

back to user need. And that will give you an indication of the real problem of all the things that may be affected. So, for example, if the error is in the setting the time, that's gonna affect your ability to properly control the selection of timing pattern. Because that requirement is traced to that user need. It probably is also traced to several other user needs. And when you're going through your whole list, through the magic of the search engine with a text document, you should be able to quickly identify the true impact any requirement has on your project, and if that requirement fails, what that means to your overall project.

**Ken Vaughn:** Well, that leaves us to the whole close-out concept, because if you know what your risks are, you know how to identify a failure, how-- the impacts that has on your project, then with all that in consideration, you can have a plan for close-out, making sure that all of your key risk items are going to be addressed before you accept the device and the impacts of accepting any failure there.

**Ken Vaughn:** That brings us to our next activity, which is the question of "Which of the following statements is not true?" In this case, we're looking for the false statement. Option A: Every requirement should be tested. B: Some testing may be performed by the manufacturer. Option C: You should only need to perform your test plan once. Option D: Traceability tables can help you to access the impact of a test failure. So go ahead and make your selection. We'll talk about that in a second.

**Ken Vaughn:** So to review the answers, the false answer, which is the correct answer, is Option C: Testing will often reveal problems; these should be fixed and the device retested. The true statements: Every requirement should be tested. That's one of the key things. Now it can be tested by other agencies or it can be tested by the manufacturer, third parties, whoever. But every requirement should be tested otherwise, why is it a requirement? Option B: Some testing may be performed by the manufacturer, which is true, or a third party or the agency itself. Option D: Traceability tables can help you assess the impact of test failure. Traceability tables allow you to identify the user needs that will not be completely fulfilled if that requirement has failed.

**Ken Vaughn:** Well, that completes Learning Objective Number 2. We did talk about developing a sample test plan, discussed assigning requirements of test phases, identified test methodology requirements to test case traceability matrix, discussed level of effort, and considered risks. And the actual sample test plan is provided in your Participant Student Supplement.

**Ken Vaughn:** That brings us to Learning Objective Number 3. Now we're gonna talk about the rules for developing a test case, specifications and procedures. We'll review the

guidance from IEEE 829, and also information in NTCIP 8007 and we'll apply that guidance to simple sample dialog.

**Ken Vaughn:** IEEE 829 defines other test documentation in addition to the test plan. It includes test design specification, test case procedure specification, and-- sorry, test case specification, and test procedure specification. What's interesting is the standard also says that each organization using the standard will need to specify the specific documents required for a particular test phase. So the standard does not try to dictate that you have to develop all three of these documents separately, but rather when you're developing your tests and your test plan, you need to identify which documents you are going to develop. And for NTCIP, as refined in NTCIP 8007, we generally group these three documents together in a single specification. So 8007 describes how they can be combined together. And basically what happens is you develop a test case identifier, couple it with a purpose, the inputs to the test case, pass-fail criteria, and then the procedure steps. The steps can reference other often used procedures that also will identify the expected outputs for each step and the features that are tested in that particular step.

**Ken Vaughn:** So the best way to explain this is go ahead and then provide a very simple example. We'll do that with set time dialog. Set time dialog starts out with a SET command from the management station, going to ASC, and the ASC provides a response. The test case will identify as a set time. So we'll develop our specification, we'll-- this is a structure tabled that is defined in NCTIP 8007 where you can have a test case with a number, a title, set time and a description. This test case verifies that the ASC properly tracks time. It advances the clock by user-defined amount, waits a few seconds, retrieves the time and verifies it, and indicates an appropriate value. We'll note that this test will advance the ASC clock by time-offset seconds. Any time that the test case alters the state of the device you should highlight that in your description because that's very important, because the tendency of test plans-- what they typically should be designed to always reset the device back to its normal operation. If you change anything, that needs to be highlighted. So we've bold-faced that in this particular case. Now in a real set-time test case, we would add in the extra steps to reach-- to change the time back to where it was, in this case, just to keep things simple, we're not gonna worry about that. The test specification will also include variables. The variable is any parameter that the tester himself may change, may set, but may change between one performance of the test and another performance of the test. So it just makes your test case a little bit more rigorous. The developer can't hardcode their implementation to pass a particular test because you're gonna vary these parameters. So in this case the time offset as defined in the test plan. So in the test plan itself we will define the value for time offset, which is the amount we're gonna advance the clock. And then finally the pass-fail criteria: The device at our test shall pass every verification step included within the test case to pass the test case. And, in fact, this is kind of boiler plate text that that tend to be your pass-fail criteria for the

vast number of test cases you'll develop for NTCIP. Note that if the requirement is from an SEP standard, you may be able to reference the standard test case as well. So if you remember back in the A315b module where we talked about developing requirements, one of your sources for requirements are other standards. We don't want to reinvent the wheel. Something like set time is a very, very typical concept. So we're not actually gonna suggest that you develop new test cases. We're only using this as a simple example that everyone can understand. But if you actually look at where we pulled the set time requirement from, from the DMS standard, you could very easily go to the DMS standard and use that set time test case. But to provide a very simple example, we'll run through how that's defined here. The first thing you do is you specify your procedures. Data exchanges should always follow defined dialogs. These are the dialogs you have to find in your standard and we discussed those in module A315b. Those are the things that the device is required to respond to and they should always respond properly. Generally speaking, you should return the device to its original state. If you don't, that needs to be highlighted within your test case. The verification steps, you should cite the relevant requirement. So at the specific step you need to identify if that step fails, which requirement is gonna be impacted. Once again, this becomes very important when you're performing the tests, you have a problem, you can immediately reference the precise line in your requirement where-- that's causing an issue. And by doing that, you can resolve a lot of these issues in real time. While you're testing you can identify the fact that, oh, maybe this wording of this requirement is perhaps ambiguous and therefore I have to change something. Or perhaps you say, "No, this requirement is very, very clear and the manufacturer's at fault," or "maybe I did my testing wrong," or something. But being able to immediately go back while you're testing to identify the requirement it becomes very important because if it is something you'd change on the fly, say if it's a tester error, then that can be done immediately rather than waiting when you go home and three days later you figured it out and you have to review everything. So being able to do it live on the fly is very, very important. Also, understand that a test case typically test multiple requirements. So by tracking the specific steps of the procedure to a specific requirement becomes very useful. NTCIP 8007 precisely defines its standardized test types. That's a huge benefit. So if you think, for example, of the SET operation, there's actually nine specific verification steps, or checks, related to the SNMP response packet. So what happens is within our test procedure, we developed a system where all we have to do is say that we will set a particular value and automatically built within that statement are nine specific verification steps that make sure that the response comes back properly, is formatted properly, references the right request ID, contains the right information, etcetera, etcetera. So all of that's done by properly using these defined terms that are defined in NTCIP 8007.

**Ken Vaughn:** So putting this into practice. Underneath that header that we talked about before that defined the title of the test case, the description, the variables and the pass-fail criteria, you would then list out all of your steps individually. So in this case, step one is a

configure step. You'll notice "CONFIGURE" is in all capitals. That indicates that it's a defined term from NTCIP 8007. And the configuration step is determining the number of seconds to advance the clock in the ASC. And we're gonna record that information as time offsets. So that was that variable that we had defined in our test plan. The very first thing we do is record that within our test case, in our implementation. And you'll notice that "RECORD" is also in all caps indicating that's a defined term in 8007 as well. Step 2, we get the following objects. Once again, "GET" is a defined term that has eight specific verifications steps. We're gonna get the following objects, global time dot zero, and that has a possible result: pass or fail. You'll notice Step 1 does not have a result pass or fail. There is not verification there, but the GET operation does include pass-fail characteristics. And because of that, we give a precise clause reference to where that requirement exists that we're testing for being able to set global time. We then go down to Step 3 and just record the response value. You'll notice that we don't have a step explicitly for making sure that the device responds. That's all built into that GET operation, one of those eight checks that are defined. So Step 3 is recording your response value. Step 4, we set the time to our new value, now incrementing our time by time offset. Once again, this is a built-in procedure that you can verify. It has a column for the results for pass-fail. You then wait fifteen seconds and then get the time again. Finally, you make sure that the new time has been incremented by not only the time offset but also the fifteen-second delay that we had. And that would be your entire test procedure. Once again, at the end of this process we've increased the time value within the device by fifteen seconds-- or, sorry, by the time offset seconds. A normal test procedure would go ahead and remove that offset to set the device back in its normal position.

**Ken Vaughn:** So that brings us to our next activity, a question on "Where can you find definitions for terms that can be used in NTCIP test steps?" Option A: IEEE 829, Option B: ISO9001, Option C: NTCIP 8007, and Option D: Student Supplement. Once again, where can you find the terms that you use in your NTCIP test steps, the terms that you capitalize? So after a few seconds will let you respond and go ahead and respond now and we'll review the answers in a few seconds.

**Ken Vaughn:** So the correct answer is NTCIP 8007. It does define a number of terms that can be used in test steps for testing. IEEE 829 defines sample outlines for test documentation but it does not define test steps for NTCIP. ISO 9001 deals with quality management, not directly with NTCIP testing. And Student Supplement does provide a lot of useful information, including sample test procedures, but it does not define the test terms.

**Ken Vaughn:** So that concludes Learning Objective Number 3. We reviewed IEEE 829 and NTCIP 8007, and we applied that logic to a simple dialog. We're now gonna take it to the next step in Learning Objective Number 4, developing a range of sample test case

specifications and procedures for NTCIP 1202. We'll look at a variety of different cases, database transactions, consistency checks, the STMP or Simple Transportation Management Protocol, load testing, and then we'll discuss some other key items or requirements to consider as well.

**Ken Vaughn:** Now these are some of the more complex features of ASC, but because they're complex they're very important to test and that's why we're emphasizing them. The first one is database transactions. Now ASCs are a required mandatory feature of 1202, so they support this transaction mode. This allows for complex data transfers. A lot of the data, as we've mentioned within the device of ASC are inter-related. So if you change one parameter, it affects another parameter. And at times, the number of parameters that are being affected simultaneously become quite large and perhaps you can't download all of this in a single message. Because of that we've defined the transaction feature with an ASC that allows you to say, "Start the transaction, now buffer all of my downloaded parameters and then commit those parameters into my database." So it allows complex data transfers. At the completion of that process you perform a variety of consistency checks. The individual steps of downloading portions of database, you can't perform complete consistency checks because you don't have a complete set of data. All this data is inter-related. You have to wait till the final end to perform those consistency checks. If at that final step, there's a problem with consistency checks, then the device is going to reject the entire transaction. So you start a transaction, download your data, complete your transaction. The device verifies things, and if anything's in error, it rejects everything that's been sent down so far. So we need a test that this mode tests correctly for all object categories under all conditions. Obviously, a complex feature like this there's gonna be a lot of different types of test you can perform. So what is interesting is NTCIP Annex A categorizes every object into one of four particular categories. It's either a control object-- and control objects can be set at any time without delay even if you're in transaction mode. So if you think you're in the process of downloading a new timing plan to your signal controller and all of a sudden there's an incident on your freeway-- you're getting traffic re-diverted onto your surface streets-- you need to change your timing pattern for this particular signal. Because that timing pattern command is a control object, you can send that at any time, even during the middle of a transaction operation and that is immediately implemented.

**Ken Vaughn:** Other objects are called parameter objects. Parameter objects can be set in normal mode, but are buffered if you're in transaction mode. So these objects are maybe the green time for a signal phase. You're not really changing anything other than I just want to lengthen or shorten my green time for a signal phase. You should be able to do that or at your minimum green or you should be able to do that without affecting the entire device. So that's perhaps a parameter you can set individually without having to go through the entire transaction mode process. Other parameters, called P2 objects, can only be set when in transaction mode. So, for example, if you're talking about-- you're

setting up your rings, which phase is in which ring, then that is gonna affect all of your timing parameters. And because of that, that has to be done in transaction mode. So if you try to set that parameter outside of transaction mode, your device will return an error. Or it should do, anyway.

**Ken Vaughn:** Finally, there's some objects that are called status objects. They are read-only type objects. You can never set them, but at any time you should be able to retrieve them and get the current value. So which phases are currently timing-type of information? Any object can be retrieved in time, including status objects.

**Ken Vaughn:** So before we develop a full-blown test case, we need to understand what we're gonna develop to. So even though this module doesn't focus on user needs and requirements, we need to understand why we're developing the test case. So we'll present user needs and requirements that have been discussed previously. So sample user need. Agencies need to be able to fully configure an ASC remotely. Fully configuring an ASC requires setting a large number of inter-related parameters that may not fit to a single request and they require considerable time to validate. However, the configuration must be implemented simultaneously due to the interrelationships among the data.

**Ken Vaughn:** Well, that's a user need, a very high level. Then we get down to the requirement, which may be something like "Until all inter-related database parameters have been downloaded the ASC shall not implement new values for any database parameter, but will process operations on non-database parameters normally." So while we just discussed, we're just kind of capturing, emphasizing the concept that all this needs to be documented in proper user needs, proper requirements. These are not defined by itself, but you should define them for your project.

**Ken Vaughn:** And then you get to the dialog. Once again another concept that we discussed in A315b and the dialog for this particular case is rather complex. It starts off with the device we walked about; it sets up a transaction. So that's what the first couple of steps are. First, you check to make sure that you're in the normal mode of transaction. If you are, then you can go and set the operation to transaction mode. And you start your transaction. Once that's been done, in any order, I should be able to set writable objects to valid values. The control objects are implemented immediately. Other objects are buffered. And then my-- I should also be able to get any object at any time. That dialog continues once you're done with all of that, then you set our transaction mode to the "Verify" state. That's what causes the device to go out and do its consistency checks. During that period which may take some time, the management station should be polling the device to see if it's done. Everything is driven by the management station. Management station is now waiting on the device. The only way it has right now to identify

when the device is done is by polling it. So it repeats that poll of getting its state until it's done. Once it is done, then the management station can go out and get db-verifiedStatus and db-verifiedError, get these two parameters to see whether or not the timing parameters are valid. And then it can set it back to normal. Once it's set back to normal that is when the buffer goes to-- is either discarded if there were errors or it's implemented if it was successful.

**Ken Vaughn:** So with that sort of complex design, there are multiple test cases we could develop. We could download a database configuration and that's the positive test that we've talked about. Or we could download the database configuration with syntax errors. Or we could download a database configuration with consistency errors. Or download a database configuration with other valid requests. So you'll notice that, you know, the first one listed and the fourth one listed are both. They're positive tests. Whereas the second and third items listed are negative tests. So there maybe multiple positive sort of tests you can do on a requirement and there may be very often are multiple negative tests. You may also want to download a valid database configuration with other invalid requests and see what happens. Download a database configuration with commands mixed with database objects or cancel a database download. So all of those are possibilities you may want to perform as test cases. A sample test case, so this once again is using that same structure that we talked about before. This test case verifies if the ASC properly buffers database parameters and so all inter-related parameters are downloaded. It initializes, starts the database transaction, downloads a number of parameters, retrieves the parameters, verifies that they were unchanged, completes the transaction, retrieves the parameters and verifies that they have been updated. So it modifies the database configuration because of that, we highlight that statement. Now you'll notice there's lots of different steps even within the high-level description so you're expecting a longer test case here. Plus, this is the database. There are a lot of parameters, so we've only listed out a few of the variables here, but there'd be a long list of variables that we have to configure. And then there are standard pass-fail criteria. That brings us to the test case. You'll notice that it starts out with configure steps and we just have ended up skipping from one to a hundred on configure steps indicating that there're a lot of different parameters or variables for this particular test case. Once Step 100, "GET the following objects(s): phaseStartup, phaseStartup.2," phaseStartup, etcetera-- and that's gonna be a GET operation, so that's pass-fail-- we get this so that we know the original state of the device that we can compare to later. So once we get that data, then we do a number of steps to record each parameter that we retrieved. "RECORD phaseStartup.1 as" the original phase startup one. And we go down the list and record all the information we retrieved. At Step 200, we get into "GET the following object(s): dbCreateTransaction". So this is now starting our normal dialog for setting up transaction mode. You first get your transaction mode verified that it's in the normal state, and that's a set-up step. So that means there's not actually-- even though this is a verification step, failure of the step does not result in the failure of the test case because it's defined as a set-up step. And if this

step fails, what it means is the device was not in the right configuration when I started my test case. So what I need to do is re-set my device, get it in the proper initial state and then start the test over again.

**Ken Vaughn:** Step 202, “SET the following object(s)” dbCreateTransaction to the transaction state. And then once I’ve done that, Step 203, now I can start setting-- or downloading my parameters, all the variables that I have in the device. That takes us to Step 300 where I then can get all that information I just downloaded. Except here, because I’ve not committed this information to the database yet, I should be getting the original values, not the updated values. Because I should be getting the information directly from the database, not from the buffer. So we go through a number of steps in the 300 series that verifies that the particular parameter is equal to the original value, not the new value. At Step 400, we go ahead and set the transaction mode to verify and then we set it-- or then we get the transaction mode. If it’s done in Step 402, then we can go on to 403, otherwise we go back to Step 401 and we go into a loop until it’s done. And smart logic would make sure you don’t exceed more than about twenty loops there or something. So you don’t go into an infinite loop, but eventually you get to Step 403. Step 403 would-- you get your Verifystatus and VerifyError, make sure in Step 404 that you’re “doneWithNoError”. So presumably everything we’ve downloaded is proper information. You should get “doneWithNoEror”. Then once that happens, Step 405, we can set the mode to normal and then once again get all of our objects in Step 406 and verify in Steps 400 remaining that the return values equal the new values that we just set in the device. And that completes the procedure. So you can see through having a lengthy procedure that’s probably not something you want to do step by step by step, manually. It is pretty extensive. It’s a lot better to be able to write the script and implement that in an automated fashion. So manual testing become very, very time-consuming and very much prone to error whereas if you can write a script, you can get that script right and then you can implement that multiple times, make sure the device is well-tested.

**Ken Vaughn:** With that, at the end of that database process, we talked about this consistency check being performed. Well, that itself is a complex concept. In fact, there are forty specific consistency checks that are performed every time you do a transaction operation. The transaction mode at the end of this consistency checks make sure that the phases-- the phase order are serviceable without conflict, make sure the phase references are valid in each context, so that Phase 1 in a traditional eight-ring controller-- Phase 1 is not designed to time concurrently with Phase 2 because that would be a direct conflict. So all sorts of issues like that are addressed within these forty consistency checks. All forty are required for a conforming device. But the manufacturer can add its own requirements. Particularly if they’ve added their own new features and extensions, they may need to add some of their own consistency checks as well. And the agency needs to make sure that there aren’t any conflicts there. So examples are on the next slide. So one example of something you might want to check is downloaded database

configuration with consistency errors. We give a description and the variables aren't any in this case because we're gonna fix everything-- just to keep things simple. So the example is we're gonna assume that we have an eight-phase dual ring operation. So Phase 1 is introducing [ph?] your left-hand turn, Phase 2 is a straight-through. Phase 5 is your left-hand turn, but it's opposing that. Phase 6 is a straight through; it's parallel to that Phase 2. And then the other direction, the cross-street is handles by Phases 3,4,7 and 8. So that's pretty typical configuration for most signals in the country. All that means that the phaseConcurrency.1 object and the phaseConcurrency.2 object can-- in other words those two phases, Phase 1 and Phase 2, can time together with Phases either 5 and 6. So Phase 1 can time with 5 or 6, and Phase 2 can time with 5 or 6. So both of those objects, phaseConcurrency.1 and phaseConcurrency.2 should both be set to the hex 0506. There's other aspects of the dual ring-- the eight-phase dual ring configuration. But that is a known fact with that particular configuration. So if we start out with that precondition, then the GET operation will perform a CreateTransaction mode and then we will start off making sure it's normal, move into transaction mode, and then we're gonna set and change your phase concurrency. We're gonna set so that Phase 1 can concurrently time with Phase 2, and Phase 2 can concurrently time with Phase 1. Now this should create an error, because with-- if that's the only thing we're changing off of eight-phase dual ring operation, the signal controller should recognize that both of those phases are in ring one and because they're both in ring one they cannot time together and thus they cannot reference each other in the phase concurrency object. What we're trying to make sure is the device rejects this command. So we're setting it to reference each other. We're then gonna get the objects and we'll make sure that right now what should've happened is on Step 5, the SET operation should have been implemented correctly. At this level we're not checking any of the semantics, but that information is being buffered. So Step 6, the information we're getting is from the actual database, not the buffer. So it should return the original value as you see in Step 7. It should return just 5 and 6. We haven't seen any error back from the device yet because what we've downloaded, the error is still in the buffer and it's not been checked yet. When you get to Phase 8, you then also verify that Phase 2 equals the original value. You then go through the verify steps. So you set your transaction to verify in Step 9. Step 10 you get the transaction mode, and then finally in Step 11 you go through your loop of making sure until you're done you back to Step 10. Once you're done you go to Step 12 and you get your VerifyStatus and VerifyError. Here, in Step 13, you make sure you're "doneWithError" this time. Now you actually do have an error that should be once you receive back, then you can check your dbVerifyError object and it should be one of two things: it should either been a Phase 1 concurrency fault or a Phase 2 concurrency fault. All of that's defined in the standard, because both these objects are in error, either one of them could report the fault. But that shows you how you might do a negative test on your device. And then finally Step 15, you would set the transaction mode back to normal. That should delete the buffer entirely. So then you go back and get your objects again, Phase 1 concurrency and Phase 1 concurrency, and then Step 17 verify that-- 17 and 18 verify that

your concurrency values are the original values and you did not implement what was in the buffer. And that completes your test.

**Ken Vaughn:** So, once again, we're at the rather lengthy and complex process implementing it into a standardized script would be very, very useful. Another test you may be interested in is STMP. Now if you're connected via fiber and you're not concerned about bandwidth, this maybe unimportant to you. But an awful lot of signals operate on low-speed connections or have bandwidth constraints due to other reasons. And STMP is explicitly defined to illuminate a lot of the extra overhead that goes on with normal NTCIP operations. It reduces overhead by at least eighty-- sometimes as much as ninety-five or even more percent of what you normally get. The typical Internet standard protocol used by NTCIP uses about twenty-five bytes for overhead for each message and another twenty bytes for each object within the message. So even a one byte request is gonna end up being about forty-five- fifty bytes. STMP illuminates almost all that overhead. Also, it eliminates the echo values. So normal STMP operations you're sending out forty-five-fifty bytes for the request and then you get back a response that's forty-five to fifty bytes. An STMP you send out a request of one to three-- or one or two bytes, you get back a response of one to two bytes, depending on the exact scenario. And in some cases you don't even get back the response, if that's where you want to configure it.

**Ken Vaughn:** So it really reduces the communications loading on your network. It requires that the downside-- you don't get anything for free-- the downside is it does require a complex set up procedure, but that's typically one-time overhead. You do it once when you purchase the device and then the device operates for years in this really efficient mode and you really come out way ahead. It also requires an additional level of processing, because there's all these extra references that are being interpreted by the controller. So it does require a little bit more powerful controller, but once again, processing these days is relatively cheap if you're buying a more advanced processor. So it is a viable option and a lot of signal controllers have implemented this feature. But that does mean that you do need to test it. Agencies should be able to monitor signal operations in real time over low-speed communications networks. As for sample or user need for this sort of requirement, sample requirement in this case is ASC shall allow a TMS to configure up to thirteen user-defined bandwidth-efficient messages for monitoring and control operations. The reason we explicitly say "Thirteen" is that is an issue that's explicitly defined in the standard. That's how many the STMP protocol actually supports. So that's shown in NTCIP 1103 in the clause that's referenced there.

**Ken Vaughn:** Once again, those are just sample requirements. We want to emphasize that every test case should trace to a requirement. That's why we've defined those requirements for you as a sample. Along with that would be a sample dialog and this is similar to the transaction mode, but this is a different operation. It's configuring your dynamic object. So in this case you set the status of the object to invalid and then you set it to "under creation". Once it's under creation you configure what information you want to set in the object-- so which parameters you typically want to exchange in a very compact form and then you get back your-- then finally you set that object to valid. So that's the

basic dialog for configuring a dynamic object. Based on that dialog there are a variety of Tests you can perform, for example, you could configure dynamic object properly. You can get a dynamic object. You can set a dynamic object, configure dynamic object with incorrect data, attempt configuration dynamic object when using the wrong process, use a dynamic object as it's being modified. So when the status is invalid, how's the device going to respond? So all sorts of different test cases can be implemented. Testing should include dynamic objects that your system plans to use in addition to additional dynamic objects. So the concept here is the precise definition of a dynamic object will almost always be dependent on the central system design. Central system will be designed to require certain pieces of information at different frequencies that will impact how they configure their dynamic objects. If I switch out my central system that new central system may have a different design. It may define their dynamic object differently. As a result if I'm buying field equipment today, I don't know what central system I might have five or ten years from now when this signal control's still out in the field. Because of that, I want to not only make sure it'll work my system today, but also with my system in the future. It's very important obviously that it works with my system today, but ideally I want the device to be flexible to handle any future system. So the concept is make sure you test out all of your device, your dynamic objects for your current system in your current system plan. But also do some other testing of dynamic objects to make sure that you do have the full flexibility there to, hopefully, support any future central system.

**Ken Vaughn:** So as a sample, we configure a dynamic object. This test case verifies ASC allows the user to configure a dynamic object. After initialization, it configures the dynamic object and verifies the settings were saved. The test changes the definition of a dynamic object. You see the variables listed and the pass-fail criteria.

**Ken Vaughn:** The test case starts out with a variety of configure statements of all the variables that we identified. You then get into Step 6, where we start setting the dynamic object configuration to invalid and then under creation to be able to download what we want, we then set the specific objects. We then set the dynamic object status to valid. And then you can get the following objects and make sure that that configuration status is equal to valid. So making sure that it accepted our command and did not result in an error. So all of that should be done and that's your standard test case for example for that particular feature.

**Ken Vaughn:** Another feature you may consider is load testing, communication loading deals with the amount of data across a communications channel. So in other words, your device is gonna be sitting out there on a wire and it has to process not only commands to itself but it's gonna see commands to other devices that it has to filter out, all while it's trying to time my signal displays. So the ASCs are often multi-dropped which means that they're seeing commands to all sorts of different devices, and they also require frequent communications. You know, a lot of the systems are designed for once-per-second polling, in addition to any other background communications that are going on. So a traffic signal very likely is gonna be hit with multiple messages per second. This is why it's such an important issue to load test your signal controller. Because none of this communications can affect the ultimate signal timing device. So as a sample test case,

you would use something along the lines of-- the description would be “Test case verifies that the ASC will continue to time phases correctly while processing a large number of complex messages. So the procedure set up would be something along the lines of defining three different dynamic objects for different types of information and then you’d get into the actual details of the test, request the first dynamic object from the device, verify the data is valid; request the second dynamic object from the device, verify the data is valid; send a third request, but this time send it to a different device address. So your devices will properly filter it out and ignore it, but you also want to make sure that that request is “set no reply”, which is a type of request that will not generate a response from the device. Most importantly, the management station will not be expecting a device so it will immediately go to the next step rather than waiting for a response. And then you repeat. So what we’ve set up here is the request your one object, request the second object, request a third object for the third device and then immediately go back and request your first object again and go into that loop. This means the central system is almost-- is gonna be constantly sending a message, waiting to hear from your device and as soon as it gets that response it will immediately send out a second message, waiting for a response. As soon as it gets back your response, it sends out another message and then immediately sends out another message. This should really start load testing that device with a lot of data on the channel. And with this as a background-- all going on in the background, you then just make sure that your signal is properly timing its green and yellow displays-- green, yellow and red displays. So that’s just an example. Obviously, lots of other tests you could perform, but it gets you thinking in the right direction.

**Ken Vaughn:** There are a lot of other issues to consider that are kind of unique to signal controllers. These are discussed as well in your Student Supplement once you get down to that level. But things like response times, how long does it take for your device to respond to your request? If it takes a very long time, that’s gonna affect how many devices you put on the wire, you should have somewhere in your requirements list, as we discussed an A315b, that you should have a defined response time for your request.

**Ken Vaughn:** Also, updating globalsetID because signal controllers have such a massive data base, there is a parameter in there that you can monitor to see if that database ever changes. It’s a very kind of simple parameter, but that’s a good check to see if the database ever changes. But, likewise, because that’s such an important parameter, you should actually monitor that during testing to make sure that that’s been implemented correctly.

**Ken Vaughn:** Another issues to consider is how you’re gonna set time over your network and that can be an issue particularly if you have a complex communications network your management station-- it may take time for that message to transmit from your management station through the network to your device. So if you want your time and

device to be within one section of accurate for of your devices, then you need to make sure you have a way of getting that accuracy. Because there could be a few seconds delay in getting that message from Point A to Point B. That's discussed a little bit within the Student Supplement. Security issues as well: The basic NTCIP protocol does not really have any significant security so you need to make sure that your physical security or your virtual security of your underlying protocols is provided to meet your needs. Data-link control issues, protocol-specific objects, and compound testing: Once again, all of these are discussed in the Student Supplement. We encourage you to investigate those specifically.

**Ken Vaughn:** Well, that brings us to kind of understanding the implications for interoperability. The standard does provide a rather limited definition. It does provide very important information in standardizing specific data that's exchanged back and forth. Unfortunately, it does not currently define the user needs, requirements, or dialogs-- any of that system's engineering information. That means the other issue is there are various non-standard features that are used in the industry. Where I live they use a flashing yellow arrow during left turn areas to show permissive-type operation. So that's kind of a non-standard, but it's something that some locales use. There's also manufacturer-specific consistency checks that you have to deal with and perhaps non-standardized-- or there are no standardized test procedures for this. So you have to develop your own. With all that in consideration, that means that if you just go out to the market and you just buy a controller, we'll call that off-the-shelf. That's not technically off-the-shelf, 'cause the market's not large enough, but if you buy a controller from a manufacturer-- you just buy their standard product, there's no guarantee it's gonna interoperate with another manufacturer's base model. So you need to make sure that you test your equipment and that everything implements properly according to your interpretation, according to your central system. Despite all of that the standard still offers huge benefits. We are in vastly better shape today than we were, say, ten years ago when we didn't have the standard. So we are moving in the right direction. Hopefully, in the future we'll standardize some of these other aspects, but we don't have a complete solution yet and you need to make sure that your custom test procedures get you that final step.

**Ken Vaughn:** That brings us to our next activity. The question is "Which of the following statements is true? Option A: The transaction mode must be used for all downloads. Option B: The manufacturer may not impose its own consistency checks. Option C: STMP testing only needs to worry about your dynamic objects. And Option D: There's not a guarantee of off-the-shelf interoperability. So which one of those statements is true? We'll take a look at the answers in a second.

**Ken Vaughn:** So the correct answer is there's no guarantee of off-the-shelf interoperability. Your specifications can impact whether off-the-shelf devices are

interoperable. All of the other statements are not true. Transaction mode is only required for objects designated as “P2” parameters. The manufacturer may define additional checks and agencies need to be aware of these and then, finally, c) testing should ensure that dynamic objects will work with any future system upgrade. You do that by just kind of throwing out other examples that aren’t using your system.

**Ken Vaughn:** Well, that completes Learning Objective Number 4, where you viewed how to test a variety of different detailed features for signal controllers including transactions, consistency checks, STMP and load testing. And we also discussed where to find other information regarding the testing key requirements.

**Ken Vaughn:** That brings us to our last learning objective for this module, Learning Objective Number 5: understanding testing results for NTCIP 1202. We’re gonna talk about a sample test summary, investigate failures with test incidence reports, interpret what failure means, inspect test blogs to evaluate test results and appreciate the need to repeat tests.

**Ken Vaughn:** The test results are defined in primarily in a test summary that provides an overview of all of your results. That summary document includes identifier-- several sections. An identifier section that just identifies documents, an overall summary, variances from the way the tests were performed from the plan, an indication of how comprehensive this test was, a high-level summary of our results whether each test passed or failed, an evaluation of if there were failures, what does that mean still for our project? A summary of activities and a set of approvals showing that everyone understood this test was performed and agrees with the results. So an example is provided in the Student Supplement.

**Ken Vaughn:** The details of your test report are provided within the instant reports. The test summary only indicates a pass-fail for each test. The details are contained in the incident reports. The incident report is relatively short document, identifies an identifier for the ports that can be readily referenced from the test summary, the summary of the particular incident, the incident description, which we’ll discuss, and the impact, which we’ll discuss. The summary references every single test incidence report and your test incidence reports are generally associated with any test that has a failure. So the test incidence report-- the description field includes the inputs that were provided to the test, the results that were expected, the actual results which presumably are different because there’s an incidence involved, any other anomalies that occurred, a date and time, the procedure step that became an issue, the attempts-- any attempts you had to repeat this test to show it is a repeatable issue and this is how you get it and any other observations. Now recognize that there are other anomalies that may-- those other anomalies may be

the issue that caused the incidence or they may be anomalies that are associated with what you discovered. So anything else? Because a failure may be a defined verification step or any other anomaly. So either one of those. So while you're testing you may have your test procedure defined listing out very precise verification points, but as you're testing you notice something else that's occurring that you know does not meet the standard. Those also need to be documented that also results in a failure. The incidence descriptions should provide in the assessment of the cause. So, you know, what caused the failure. Was this due to the device not performing properly? Was it due to the user not performing the test properly? And in that case you can kind of chart the test procedure because you should just read from the test, make a note that the user made an error, but you don't really have to worry so much about those particular tests. That's user error. There may also be though a problem in the test procedure itself, the way it's written up that may mean you have to go back and re-write that test procedure. There may be a problem in the specification. There may be a problem in the standard. So there should be an assessment within the incidence report, "What caused the problem?" You also need to-- the incidence impact-- identify what impact this failure may have on the project. So this is where the traceability tables come into play. They can help identify the requirement that failed. From that requirement you can identify the user needs that were impacted. There may be secondary impacts due to interrelationships. And they also maybe identify impacts may include unrelated device operation. So, for example, if I can't define a new timing pattern, then that's gonna prevent me from properly doing my schedule of timing patterns. So it may be a broader concept of affecting a whole suite of features, depending on exactly what the failure is.

**Ken Vaughn:** Now the incidence description will also point to details that are contained in the test blog. That allows the developer very importantly to go out and investigate the exact details. So the test log should include what we'll see here in a second, that includes very detailed information, becomes very useful for a developer even though it may not be terrible interesting to the tester or agency. But it can help them debug the application. And it should also reveal exactly how to reproduce conditions. You know, if you have this data logger, external data logger that we said is an optional component of your environment, that can be very useful in capturing any time you go off your test procedure plan. It'll show you exactly what you did and exactly how you ended up where you did with a failure. It exists in diagnosing the problem as well. That test log normally from the test application looks something like this. It will record each step, and it's a log of when each step was performed, whether it passed or not and what you did during that step. And with a lot of automated tools, there's actually even additional details you can access by, you know, clicking in on that get global time-- you can find all of the individual eight verification steps that are behind that get-- or global time step.

**Ken Vaughn:** Now that was for the test application, if you also have a data logger that's going to just record the data packets going back and forth. So an example in this case

would be shown here where you see the frame from your central GET operation on global time and then almost immediately you get back a GET response. And then that's followed up with a SET. Once again in a particular color with the response coming back in a different color showing that it's from a different device. So that test file also provides a lot of low-level details and, once again, you can look at global time, you can see what value it was set to, what the value returned was, all sorts of low-level details. And by looking at all of these low level details, looking at the byte-by-byte example of what's contained in those packets, that developer can really get into and debug what's wrong with her implementation. Likewise, if the user had an error in what he did, he can go back and check his steps to see if there's a problem.

**Ken Vaughn:** So which brings us to the concept of repeating tests. It's one thing-- maybe systems are very, very complex Sometimes you have an anomaly that occurs, you know you had an error once. You go back and immediately see that same sort of thing on the device and this time it works. So that's why you make sure your tests are repeatable. If you automate scripts, then you can do an exact repetition of that test multiple times, you have it all recorded on a file, and this allows you to determine, "Does this failure problem come up often? Every single time? Periodically? Very seldom? Only once?" And depending on if it's every time, you know there's a real bug, you can generally fix it pretty quickly. Intermittent problems are a lot more difficult to solve, but at least you know your base software in that portion of the code probably is working right. You're now looking at different types of issues when you're trying to debug that sort of problem. So repeatability of tests is very important to debug the software. Even mainstream software has frequent updates and proper testing goes beyond "Does it work?" You want to make sure that when I send invalid data it responds properly. If there's byte errors in my packets-- or bit-level errors in my packets due to communication-level errors then the device responds properly. So. The identified failures need to be corrected, reported to the manufacturer, corrected, new device sent back, and then you go repeat all of your tests. This often requires multiple rounds of testing. This multiple rounds of testing take time and budget, as we discussed before. So how did all of that-- how does all that come into play? Hopefully, you've addressed all of that in your test plan and you know how it's all gonna impact. You know who's gonna pay for that extra testing. Very often once you do a signal controllers because they are complex devices, because you do expect some problems, you'll say, "Okay, you get one round of testing for free, we'll allow you to fix it, but the second round we expect you to pass. If it goes beyond that then we expect you to share the costs with us." But also schedule penalties for schedule delays, schedule constraints, consider those. Do you have a big event coming up that you have to be ready for? And you need to consider is there a point where are just gonna through our hands up and say, "No, we're not gonna buy this equipment. We're gonna go to someone else"? If you plan for all of this properly upfront then it minimizes the close-out problems.

**Ken Vaughn:** That brings us to our final activity, "Which documents discuss potential impacts for testing failures?" Test summary? Test incident report, is Option B. Option C is test log. And Option D is test summary and test incident report. So which one of those documents discuss the potential impacts of testing failures. We'll talk about the answer in a second.

**Ken Vaughn:** Well, the correct answer is both A and D-- or A and B, which is Option D. The test summary includes in the evaluation clause providing an overview, and each incident report provides detailed impact clauses. So Option A was incorrect simply because Option B was correct [ph?] as well. Option B is incorrect simply because Option A test summary also provides that information. And Option C was truly incorrect; the log does not contain that sort of information.

**Ken Vaughn:** Well, that completes Learning Objective Number 5. We did talk about the test summary. We talked about test incident reports and the test log. We talked about understanding what a failure means and the importance for planning multiple rounds of testing. What we learned today includes testing ASCs is important due to their safety-related issues. And every ASC requirement should be tested a least one time in order-- in at least one test phase using at least one test method and by at least one test party. In Section 3 we talked about NTCIP, the test deign specification, test case specification and test procedure are typically defined in a combined test procedure table. And Learning Objective Number 4, due to various complications there's no guarantee of off-the-shelf interoperability for ASCs. And, finally, in Learning Objective Number 5 we talk about the test summary, test incident reports reveal the impact of failures on a project.

**Ken Vaughn:** There are several resources for this document. There's NTCIP 1202, Version 2.19, that was published in 2005. And also NTCIP 9001, which is the NTCIP guide, version 4, published in 2009, that 1202 actually is the standard itself for certain controllers. There are actually other NTCIP standards related to this topic, including the global object definitions, 1201, and there's also NTCIP 8007 that we mentioned. All of those are available on NTCIP.org website. In addition we talked about the IEEE 829 standard from 2008. That is available at IEEE.org website. With that, that brings us to our questions. One of the questions that we have heard from this topic is "What percent of budget would you typically use for testing and percent of the overall project?" And the answer there is unfortunately a lot of implementations within ITS under-test their deployments and that's where they get into problems. If you look at the broader community most systems advise, indicate that you should be spending fifteen to twenty percent of your overall budget on testing. And it really kind of doesn't matter what sort of procurement you're involved with. If you're procuring well proven, known devices, well, those devices should've already been developed so your development costs should be fairly low, which means your overall project costs would be relatively low. You're only

spending twenty percent of that relatively low project cost. If you're developing new software for a particular project, your project is gonna be more expensive, but because you're developing more-- brand new software, you should be testing it more rigorously as well. So, once again, that fifteen to twenty percent budget would still apply. So that's a pretty good rule of thumb of fifteen to twenty percent. What I've seen in the industry is closer to five to ten percent. Maybe ten to fifteen percent in some projects. Very few projects spend what the IT industry recommends for testing. It should be in the range of fifteen to twenty percent. So, another question we get is "How can we get assistance?" Well, the Federal Highway does have resource centers. You can access via the Federal Highway website at fhwa.dot.gov. You can also access the peer-to-peer network there. But the recourse centers have operations teams and within the operation team there are number of very knowledgeable individuals who should be able to assist you.

**Ken Vaughn:** Another question we get is "How can we get additional training?" And there are a variety of modules available at probably the same location you got this information: PCB as in Professional Capacity Building www.pcb .its.dot.gov and that website will get you all sorts of the training modules that we've developed through this course as well as lots of other training for ITS. And with that, that concludes our presentation and I thank you for attending. Thanks a lot.

#### End of T315_edited.mp4 ####