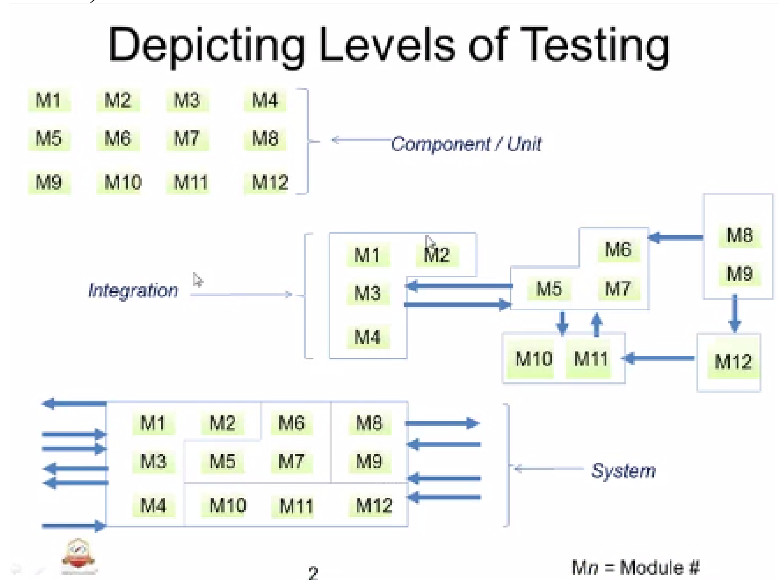
# Embedded Software Testing
## Unit 4: Software Integration
## Lecture 1
## Seer Akademi – NPTEL MOU

We are on a new session series which is on software integration and embedded software integration, embedded software testing, you know we need to have a software integration, basically there are two things we are need to be understanding, embedded software integration aspect is an activity of embedded software life cycle and we should be considered as an activity from both developers and expected, that means to say that they are going to have an integration, definitely integration is the integral part of the embedded software development life cycle.

Meanwhile we are going to test an integrity how it is going to be done, so the second aspect in terms of integration testing, we are going to focus, so you should not confuse between the software integration, from the developer assertive and expected perspective, we should be separate this, developers have, they have their own life cycle aspects in terms of integrating the various models. Basically here we focus on the testing column of embedded software integration, which is what we are trying to focus on.

(Refer Slide Time: 01:32)

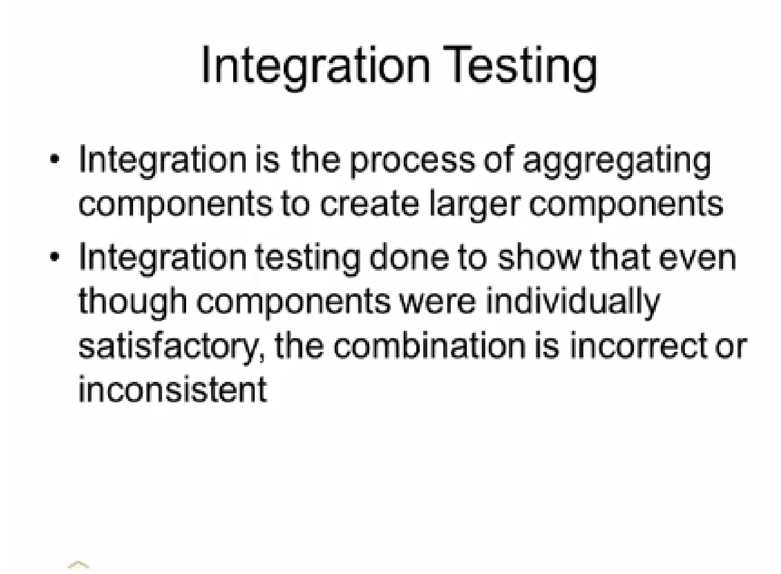


## Depicting Levels of Testing

So, to recap on some of the levels of testing, just you want to understand what is embedded software integration, we will see the three level, layers of software testing we know that there are about, for example 12 components in an embedded software unit, how they are going to be tested at different levels, we know that component at different level, we are going to test it individually, with the help of stumps, or any other drivers, that is all the part of the unit testing, where the individual units are the input.

So those are focused with their objects, data, or parameters, and on the higher set, I am coming to the bottom part of the types of level of testing, system level were all these can be used in the components of the embedded software systems, which it is completely put together and tested next Monday, you can see the arrow mark completely outside the box.

So you can consider M1 to M12 are the complete unit at the system and the system is being verified externally with external triggers or signal fields and injections likewise, so this is systems level, first is the component level, the one that we are trying to focus is integration at where there are different module boxes which will be integrated to make it to the system.
So we start with the components we have tested it, the components are fine, and the units are okay, then these units basically grouped into the logical index, so these logical index we have 1 logical unit and also two and it can logically grouping, so it is again based on the various aspects as I said on like its functionality, complexity and any performance related, control system related , all these feature files, depending on the system that we are going to have the various logically divided units into different groups, that's what we are doing.
So here we have example, 1 2 3 4 units, here logically we cannot separate the input, so this is what we need here the logical are integrated, so may be different types of integration are there, like bottom up top down, we will study that, and the primary intention is to see that this modules when it is getting integrated how they value, what are the inputs and outputs to getting handled by these logical groups by what they are tested, so there are 5 groups here, in this example and we are going to integrate is, while integrating there are different types of testing aspects we are going to test the embedded software testing.
(Refer Slide Time: 5:55)

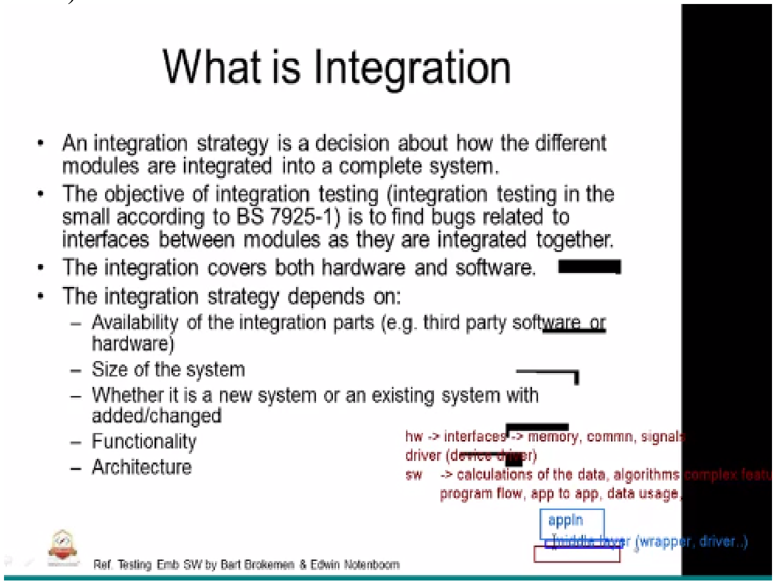


## Integration Testing

- Integration is the process of aggregating components to create larger components
- Integration testing done to show that even though components were individually satisfactory, the combination is incorrect or inconsistent

So we will try to understand the scope of objectives then we have considerations strategies, we will take up few examples, in the series of sessions of embedded software integration testing okay, integration is the process of aggregating components to create larger components as is said we have different components we are going to aggregate them like M1, M2, M3, M4, are the individual components we are going to aggregate on logical group so we are going to aggregate those components so it will become a larger components, so there are 5 groups here and we are going to integrate is, while integrating there are different types of testing aspects we are going to test the embedded software testing so that is was the process of integration is done.
Integration testing is done to show that even though components are individually satisfactory, the combination is incorrect or inconsistent, so testers are going to make sure that this is fine and

there are errors with certain types of inputs we are going to combine each other, so it is not we are going to satisfy individual components, any work individual is fine, but when integrating each other what is the behavior of that, so that is what the tester is going to test.
So the objective is to make sure the components are satisfactorily working or if there are any issues the tester is going to be caught out and inconsistency and incorrect all this will be part of the errors will occur while doing integration test.
(Refer Slide Time: 8:30)



## What is Integration

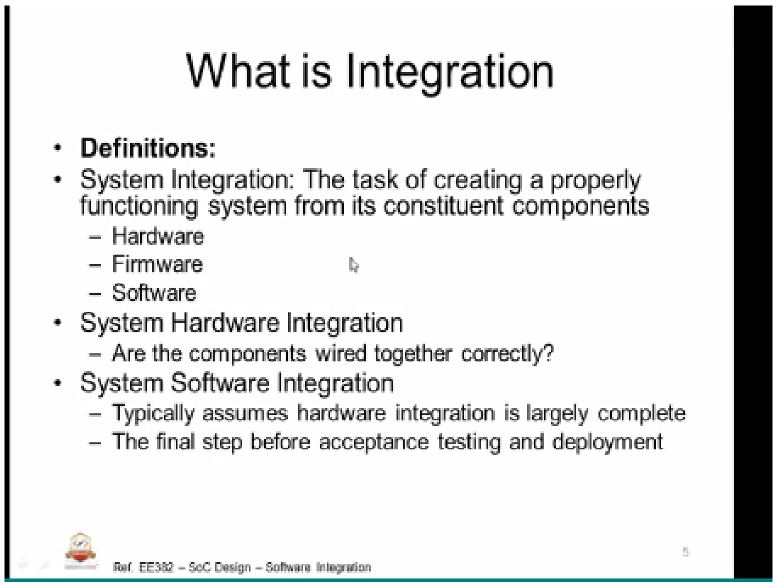
- An integration strategy is a decision about how the different modules are integrated into a complete system.
- The objective of integration testing (integration testing in the small according to BS 7925-1) is to find bugs related to interfaces between modules as they are integrated together.
- The integration covers both hardware and software.
- The integration strategy depends on:
  - Availability of the integration parts (e.g. third party software or hardware)
  - Size of the system
  - Whether it is a new system or an existing system with added/changed
  - Functionality
  - Architecture

hw -> interfaces -> memory, commn, signals
driver (~~device driver~~)
sw -> calculations of the data, algorithms complex featu
program flow, app to app, data usage,

Ref. Testing Emb SW by Bart Brokemen & Edwin Notenboom

Okay going with definition, just try to understand, an integration strategy is the decision about how the different modules are integrated into a complete system. The objective of integration testing in the small according to small BS7925-1 is to find bugs related to interfaces between modules as they are integrated together, that is we do not want to integrate everything together, we can do a progressive decision in terms of positive integrate components 1, 2, 3, 4, 5, etc then implemental it is added to the behavior, this is what we will do in modules as in when they are integrated each other, and the integration covers both the hardware and software, as I said in one of the earlier session, there are two session of integration systems.
So we have hardware involved and software involved, and you know hardware is what, from the various inspective through interfaces, so it will be using any of the hardware part, it could be a memory and it could be a communication device, or any signals, so it is a drive also called devices driver.
So through these interfaces we are going to interface the hardware, so this is what we are going to do in hardware integration, so in software what we do, we have calculations of the data, algorithms on the component features, then they have program flow, fan in and fan out, but you know that software there are it could be the data usage and transformation into low level or high level aspects, what we have is three are two are drawn in simple measure okay, the first layer is talking about the application and the second one is the middle layer you can also called it as rapper or driver level etc...The third one will be the actual hardware so we have scope for integration in two aspects. What are those one is between this two this and this, other one is between this and this so the one between this and this as well as this is another application, so we

know that applications can be multiple right, between application to application or application to another subsystem app.
In case of any supposes or portion and apps all those components. Likewise you may have so all this will be all software-software integration, where the other part where you use the hardware and software in mode, such as application to drivers. Here application could be a middle layer or rappers.
It should interact with the driver. So this is called hardware-software integration. So please remember that two aspects of software integration. Software-software integration have to be considered and taken care will be the integration testing. Some industries what they do is they may not have a segregation of hardware-software and software-software. They might have a purely software-software, they take care of this as the small device, device drivers, internet source, signals and all this.
Within the software itself they do not have those segregations. They can call it as an in general integration software or software integration. But most of the industries we have like space we have to hardware to software integration, software to software integration. So they are very important. So integration covers both the aspects that are what it is means. Integration covers both hardware and software the next one, the integration strategy depends on availability of the integration parts.
That means if any subsystem library or apps or third parties of memory or signal it could be etc. so that availability is very important. What doing there integration? Because we need to integrate them also as part of the complete embedded system, and the next one is the size of the system. That means what is the size that system has? How it can be broken up? Further in to different integrated or module of elements.
So that is all given based on the size. Here size means not the lines of code. Please remember, as I said in my earlier class that size is not the LOC, it is the structure and the complexity of the system that is may be test of the integrations software testing. So that also matters in terms of working of the integration strategy. The next being the whether it is new system with added change.
So, we know that the system is working stable and qualified and we want to enhance it or you want to modify, or you want to add something fresh so, that information also important in terms of integration so that the strategy can be worn down. Then the features for the functionality that is also important so what are the various features that make to be tested like feature could be externally.
So all this matters so basically system knowledge is very much important in terms of hardware-software integration or software-software integration. This is the vey key element of the embedded software testing. The integration testing. The last one being the architecture, we have right architecture basically here all the different blocks and cases, use cases, and the sequences and classes and all that infoprmation so that is also very important identifing the integration testing strategy.
(Refer Slide Time: 17:04)



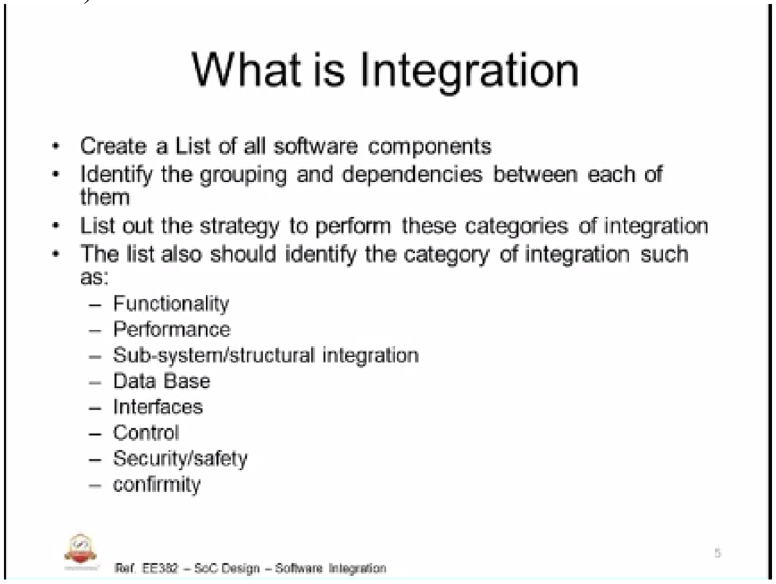
## What is Integration

- **Definitions:**
- System Integration: The task of creating a properly functioning system from its constituent components
  - Hardware
  - Firmware
  - Software
- System Hardware Integration
  - Are the components wired together correctly?
- System Software Integration
  - Typically assumes hardware integration is largely complete
  - The final step before acceptance testing and deployment

Ref. EE382 – SoC Design – Software Integration 5

Okay, so we will move to next so from the web, I have colletral some the important embedde software integrration definition. System integration, the another aspect of the integration so we are studied about software-hardware integration, software also can called as the embedded system next one is the embedded software integrstion testing, the third being the system integration near all the individual complete components like externall memory.
Internal memory external signals, on field related inputs in terms of firmware, hardware, software all this in to put together as a complete black box and tested that is calle dsystem integration it is also can be called as system tesing. But basically the different system integration that is third level of integration.
Before that software integeraion and hardware integration complete. System integraion it task of crarting a properly functioning system from its constient compounts. So what are those constitient compounds? One is hardware other wise firmware and the other wise software. So all these constituent components of the system integration. Next one being the system hardware-software integration.
Are the components wired together correctly? That meands whether here wired means not we are going to check the hardeware part as the in terms of any wiring soldiring above that. That is not the pupose. Works, that is fine we are going to make it validate, make it sure that we are able to interact which those wired components. That is all harware of capability aspects whether we are able to interact read and write to the memory.
And to read and write to the memory or the external communication with the specified moderate and specified speed all the aspects are important. Just wiring may not be suffiecient it has to be done appropriately such that we are able to do the interactions in a intented way what specification of the reqiurements talk about. So that is what the system hardware integration that is it. the next defintion is system software integration this is typically assumes hardeware integration is larglly complete that is the above part is almost done initial step before acceptance testing and applying. So this is the last step they say but during the sofware intgration is also one

of the type we do for putting all together the build they complied software piece in to the completed hardware.
So this is what we do in the system software integration.so we have a hardware-hardware, hardware-software in complete system, so this three levels of integration testing they will do it.
(Refer Slide Time: 20:31)

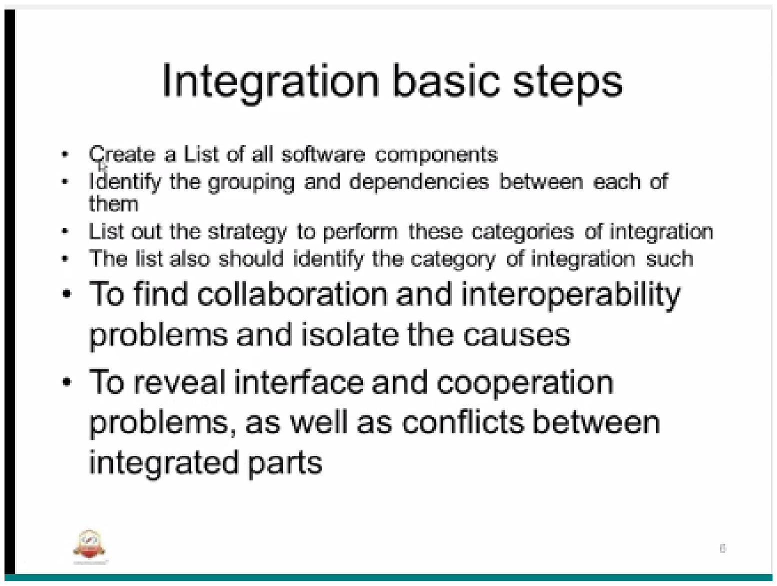


## What is Integration

- Create a List of all software components
- Identify the grouping and dependencies between each of them
- List out the strategy to perform these categories of integration
- The list also should identify the category of integration such as:
  - Functionality
  - Performance
  - Sub-system/structural integration
  - Data Base
  - Interfaces
  - Control
  - Security/safety
  - confirmity

Ref. EE382 – SoC Design – Software Integration 5

okay, so next thing of the integration test so basically interagration what are the thing we going to take care. This primally what I am trying to put here is integration it is an activity this can be taken credit from the test prospective as well as devlopers prosepective. Definitly, developers will have to follow some of this strategies or elements of the integraion like you need to take care of software-software, software-hardware and all that. But this focus will be from the developers point of view.
Wheras testing prospective is fromunit prospective, in terms of integration and how they are integrated and how it make it fail and what are the injection in terms of error of false and each and right. So that the maximum performance will be done. So integration tests basic steps are as per below.
(Refer Slide Time: 21:41)



## Integration basic steps

- Create a List of all software components
- Identify the grouping and dependencies between each of them
- List out the strategy to perform these categories of integration
- The list also should identify the category of integration such
- To find collaboration and interoperability problems and isolate the causes
- To reveal interface and cooperation problems, as well as conflicts between integrated parts

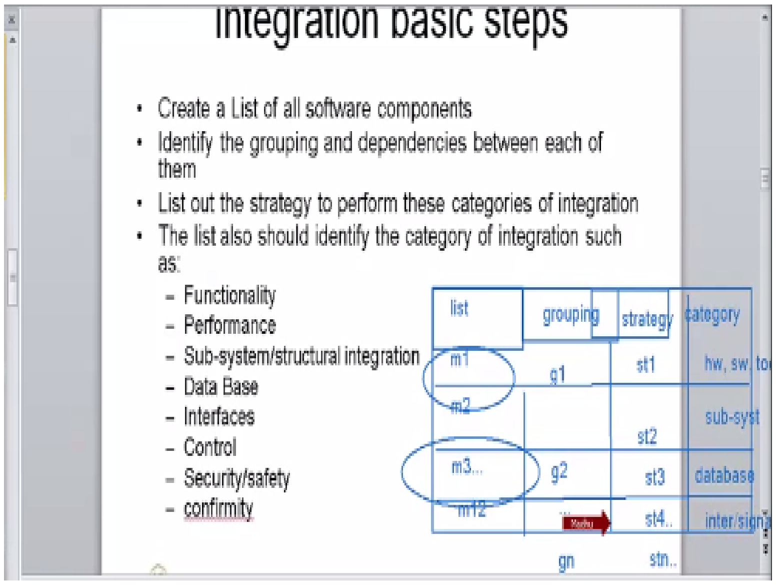
Create a list of all software complexes. As I said we different components in earlier example slide I have put well components so what are the software components that we have? The next step is identify the grouping and defined it between each of the so once we identify different components, identify the group we aspects how we individual components are shuffled each other.

And what is the dependency of each of them. Because the one functionality or couple of functionality cannot be implemented in to definitely we have to break it down in to individual software components. So we need to identify what is the dependency and interfaces in terms of the grouping between each of them. That is another aspect that we need to identify. Then the next thing what we have going to do list out the strategy to perform these categories of integration.

That means they identify the grouping and dependencies how we are going to test this grouping? How we are going to test this dependency between each software components. That is what the strategy. This strategy will bring out all the aspects of integration like hardware-software, software-software and system software of the system integration to perform these categories of integration.

So basically the list should identify the category of integration such as the category of different categories of testing strategy so what at those functionality, performance subsystem or structural integration, we have data base, we have interfaces, we have control, security or safety, conformity. So this is some of the steps that are to be taken care.

(Refer Slide Time: 24:00)



## Integration basic steps

- Create a List of all software components
- Identify the grouping and dependencies between each of them
- List out the strategy to perform these categories of integration
- The list also should identify the category of integration such as:
  - Functionality
  - Performance
  - Sub-system/structural integration
  - Data Base
  - Interfaces
  - Control
  - Security/safety
  - confirmity

| list | grouping | strategy | category |
|---|---|---|---|
| m1 | g1 | st1 | hw, sw, to |
| m2 | | st2 | sub-syst |
| m3... | g2 | st3 | database |
| m12 | | st4.. | inter/sign |
| | gn | stn.. | |

Okay, so I will just try to draw a small table here, so what we do with this, we will try to identify the various lists, VVS the integration aspects. First provide the list of all software components, they want to profile the grouping aspects they want to identify strategy, then again another column you can have a category. So these four things are first identified in the typical integration test strategy tool or a sheet, I would say this is basically an identifier, which identifiers?

These aspects what are the steps involved for integrating the test step. So first we are going to create list of software components, so this will list. Right, okay, I am just trying to draw an example of how we can list out the steps. So the first thing is you create a list of component like we did there, m1, m2, m3, we have m12 right, in the left side, we are going to group together somehow way we can group m1, m2, m3, m4 as one group then m3, m4, m5, m6, like this so we are going to identify group with that name, group one or a group two, likewise they have group n.

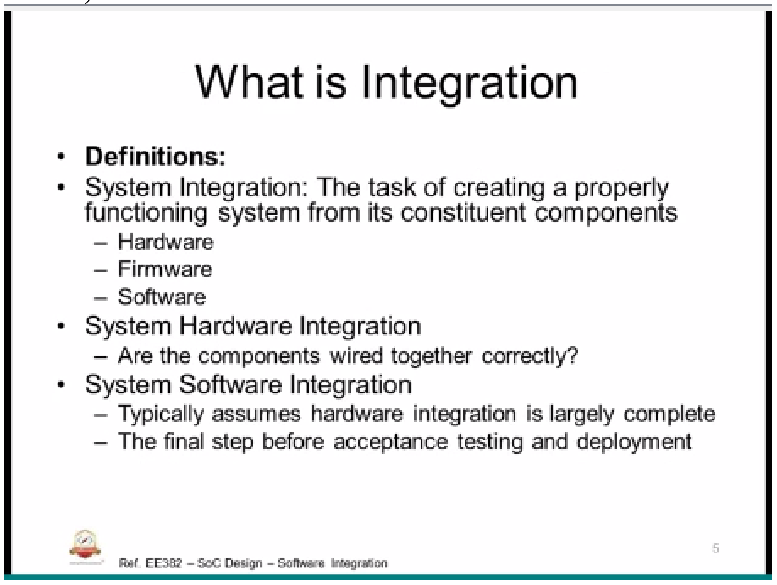
So how we are going to group perform the grouping basically identifying the similarity of the different software functionality, how they can be grouped and dependencies that each group of the software functionalities that it has, is one of the identifier. With the help of that, this two columns are as per on, the next for each of this group, what kind of strategy I can have, what kind of integration strategy I can have.

This can be done with the functionality in terms of externally injecting a some values and this can be done with the help of the tools in terms of memory or in the terms of speed or communication aspects, this can be tested with large data base and any data base modification or depend types of data base we can use, all these strategy will be part of this third column and called strategy. And this strategies will have its own types of test and Normand type of , see in the testing aspects, like strategy one, strategy two, strategy three, strategy four etc., we will have strategy n. so for each of this strategy, so what kind of category we can make, like it depending on the hardware, software any tool, likewise you can have a list like what sort of category, this category has a system dependencies then any database or database automation, then we need an any interface or single specific inputs for this strategy all these will be listed.

So with the help of this we are going to conduct the integration so these are some of the main basic steps of the integration testing. So the list should be identified the category of the integration, such as functionality, performance, subsystem or structural integration, database, interfaces, control, security or safety, conformity etc., conformity is something like the integrated part can be confirmed with the values of whatever the intended inputs.

Similarly security and safety aspects something like, if requirements says within this limit, this as to behavior, which as to behave and upon certain safety interact, this should lock, those kinds of requirements can be there, and that also needs to be taken care while doing the categories of integration. So the each category will have its own sort of inputs, conditions, and the expected results. I guess that we need to write the test cases and draw out the integration test procedures and execution, this that means.

(Refer Slide Time: 30:32)



## What is Integration

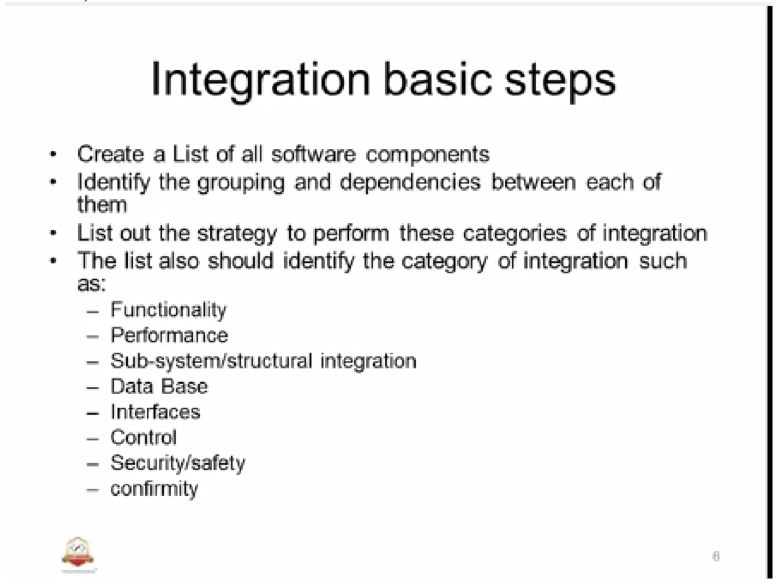
- **Definitions:**
- System Integration: The task of creating a properly functioning system from its constituent components
  - Hardware
  - Firmware
  - Software
- System Hardware Integration
  - Are the components wired together correctly?
- System Software Integration
  - Typically assumes hardware integration is largely complete
  - The final step before acceptance testing and deployment

Ref. EE382 – SoC Design – Software Integration

5

So we have seen the definition of integration,

(Refer Slide Time: 30:34)

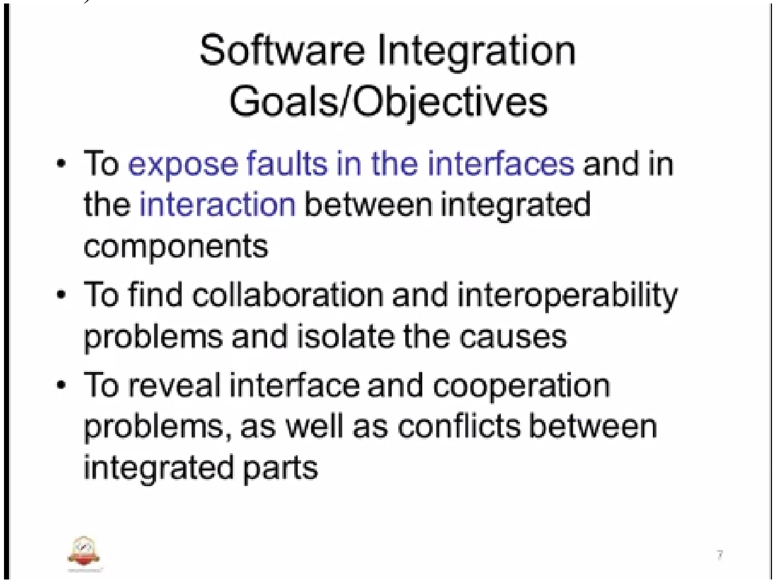


## Integration basic steps

- Create a List of all software components
- Identify the grouping and dependencies between each of them
- List out the strategy to perform these categories of integration
- The list also should identify the category of integration such as:
  - Functionality
  - Performance
  - Sub-system/structural integration
  - Data Base
  - Interfaces
  - Control
  - Security/safety
  - confirmity

6

We have seen the integration basic steps,
(Refer Slide Time: 30:38)



## Software Integration Goals/Objectives

- To expose faults in the interfaces and in the interaction between integrated components
- To find collaboration and interoperability problems and isolate the causes
- To reveal interface and cooperation problems, as well as conflicts between integrated parts

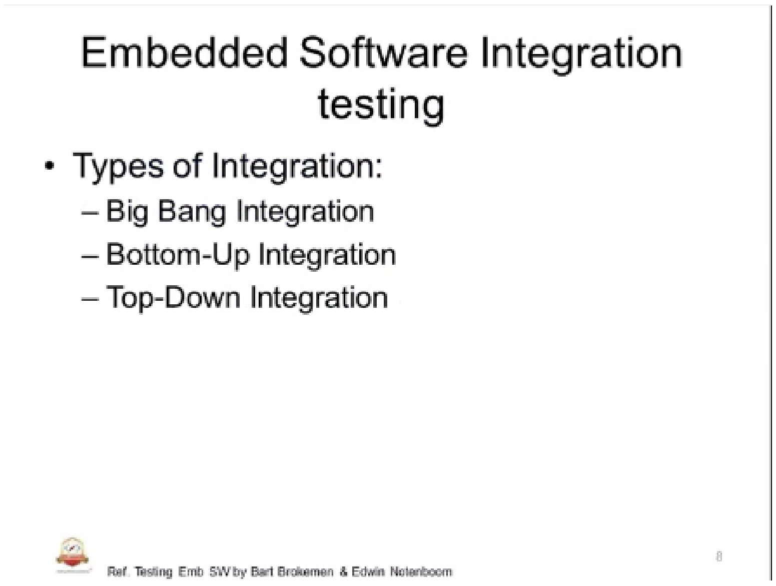
Software integration, what is our goal? Basically to expose faults in the interfaces and in the interaction between integrated components, that means it is not just enough to find out the faults, but also to see the interfaces are properly behaving and the interfaces are properly giving in the terms of speed, performance, any issues in the terms of full load not possible or it is the collapsing after sometime durability all these aspects will be exposed, that is one of the integration object.

Next one is trying to collaboration and interoperability problems and isolate the causes, that means interoperability is inter operation we are going to have along the external system or software systems, how it can interoperate with those systems is what we are going to find out while during the integration testing. Also see the collaboration of different components as we studied in the earlier slide. The next one is reveal interface and cooperation problems, as well as conflicts between integrated parts, so it is very important to identify the intention of different integrated parts, why because those parts are basically developed by different people and if you have the different perspective in the terms of requirement understanding and you would have implemented wrongly or you would have objected those modules wrongly.

So all these conflicts in the terms of requirements it could come out easily when doing the interface and cooperation problems, testing in terms of software integration. Why I am telling this is, because we cannot point out the faults in the terms of integration as well, integration alone but it could be the problem in terms of specification and design also. So all this will be directly or indirectly identified, if there is any loss are there in requirements or specification or the customer inputs.

Those conflicts will all be brought out while doing the software integration, so these are some of the software integration goals and objectives.

(Refer Slide Time: 33:24)


## Embedded Software Integration testing

- Types of Integration:
  - Big Bang Integration
  - Bottom-Up Integration
  - Top-Down Integration

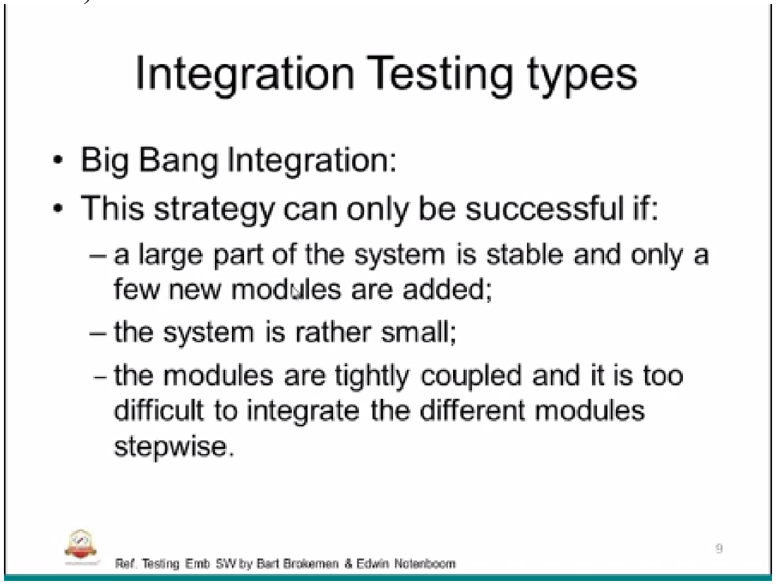
Ref. Testing Emb SW by Bart Brokemen & Edwin Notenboom

8

So, now testing types, we have seen the levels like hardware software, software, software, system software integration. How we can achieve with those and what are the types that we have? And we can pick out any of this or what are these, all are this depending on the complexity of embedded system software. All of this has the own advantage and disadvantage. The first one is big bang integration; the next one is bottom-up integration, and last one is the top-down integration.

So we know that we have different modules and components. How it can integrate? What aspects of the integration we can take care whether we can start with the lower level or start with the higher models or we can directly that agree entire bundle, so all these will decide in terms of integration testing level type definition in this test I think method, so base on that we can categorize that testing types as big bang integration, bottom up integration and top-down integration. We will try to study,

(Refer Slide Time: 34:41)


## Integration Testing types

- Big Bang Integration:
- This strategy can only be successful if:
  - a large part of the system is stable and only a few new modules are added;
  - the system is rather small;
  - the modules are tightly coupled and it is too difficult to integrate the different modules stepwise.

Ref. Testing Emb SW by Bart Brokemen & Edwin Notenboom

9

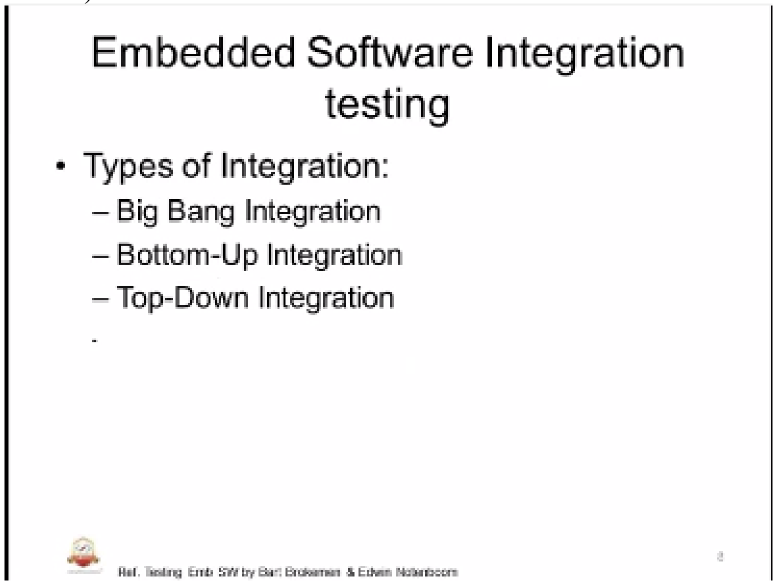
Which of the emirate, big bang integration, this strategy can only be successful if a large part of the system is stable and only a few new models are added, the system is rather small; or the modules are tightly coupled and it is too difficult to integrate the different modules stepwise, so big bang integration basically this is not used very heavily, that I will say in the industry, but in industry where we have difficulties in the terms of integrating that individual models or if it is the individual module tightly coupled or it is small to test it in the integration level.

What we do is, we will go with the big bang integration where we put together those modules and small modules which are parted from the main modules together it will decide to integrate. We bang it in the terms of reading the values on the field or in the emulation of, whatever the test environment we have, that is why this strategy will be applicable for those kind of systems, so the last part of the system is stable I mean we know that system is working we don't want to break it but some features or some modules are added or some components have been taken out or modified. So what we will do? We will verify that are interface or that aspects which have been added or modified founding, so we take with the big bang integration strategy for those basically.

And the system is too small we cannot disintegrate and it is difficult to test it, the modules are tightly coupled so that we cannot make it apart and it is too difficult to integrate the different modules stepwise. You start with module two three four then five six seven like that so we prefer from the two three four we can disintegrate whereas five six we cannot we have test five six together and it is integrated the higher with two three four or another modules.

Those kind of situation what we do we will test it as a whole that is called big bang integration.

(Refer Slide Time: 37:21)



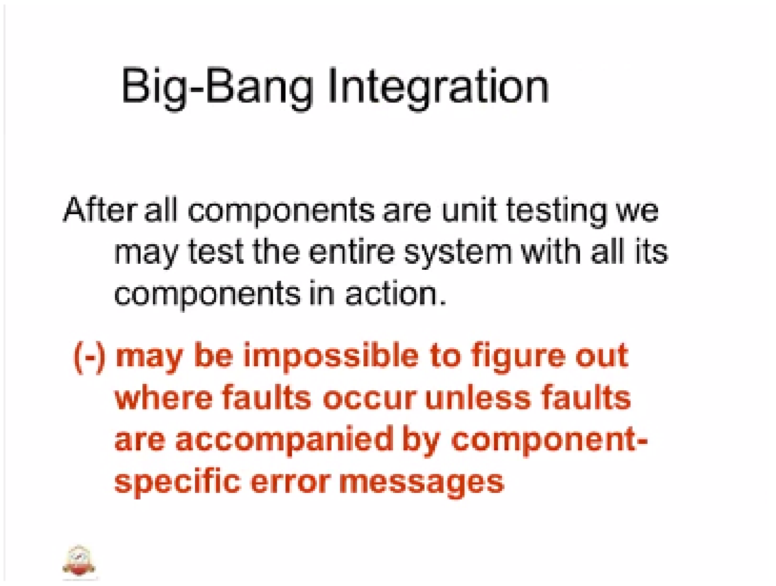
## Embedded Software Integration testing

- Types of Integration:
  - Big Bang Integration
  - Bottom-Up Integration
  - Top-Down Integration
  -

Ref. Testing Emb SW by Bart Brokeman & Edwin Notenboom

The next type of software integration testing is bottom up integration here what we do is,

(Refer Slide Time: 37:26)



## Big-Bang Integration

After all components are unit testing we may test the entire system with all its components in action.

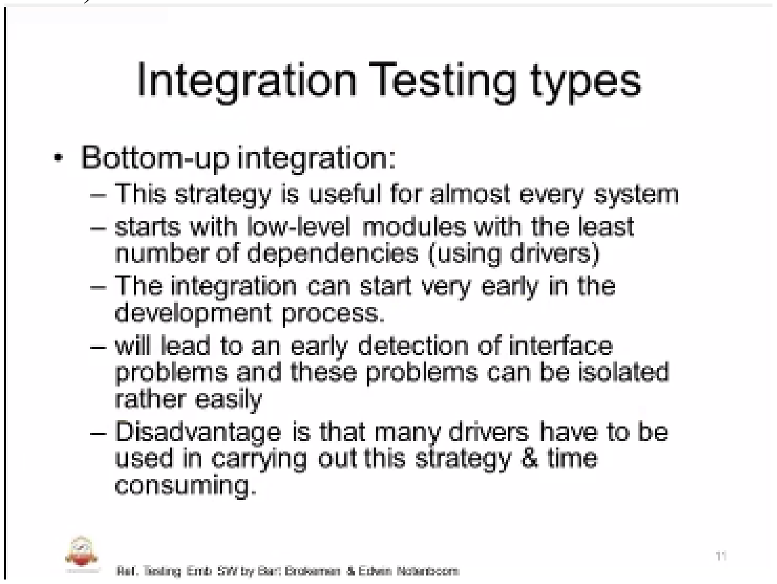
(-) may be impossible to figure out where faults occur unless faults are accompanied by component-specific error messages

There are few for big bang integration, after all components are unit testing we may test the entire system with all its components in action, that is what integration testing does. May be impossible to figure out where faults occur unless faults are accompanied by component-specific error messages, that means here this meaning is we may have to take a credit of components so what will happen is component testing we have some errors.

So component testing we have some other, issues or faults or if any pass fail situation is there so we will also consider that while doing the big bang testing, because we are not sure if we face issue or faults as we have not integrated completely, so but still we fells some issues or faults are occurring what we will do? We will focus on the integrated components and we try to attack the complete target in terms of big bang integrations so that is what we do in big bang integration.

So the next type of software integration testing,

(Refer Slide Time: 39:13)



## Integration Testing types

- Bottom-up integration:
  - This strategy is useful for almost every system
  - starts with low-level modules with the least number of dependencies (using drivers)
  - The integration can start very early in the development process.
  - will lead to an early detection of interface problems and these problems can be isolated rather easily
  - Disadvantage is that many drivers have to be used in carrying out this strategy & time consuming.

Ref. Testing Emb SW by Bart Brokeman & Edwin Notenboom

11

Is bottom up integration, in this what we do is? Basically we take up the low level or the lower most modules we start with that and dependence with here with other components are there, he started that and read it one by one step and we will be there, basically this is very useful in first of the systems they because whereas it could be anywhere and it is easier to identify when you start with the lower most part.

So the higher levels modules are integrated as the module space that is of the usefulness of the bottom up integration, starts with low level modules with the least number of dependencies using drivers, drivers are the module testers, test drivers it is called, where we have a thumb columns which calls the integrated low level modules and the expected results are expected with that drivers and the input will be generated in the tests

With the help of printers or whatever, it leads early the primary component testing how we do? With the instrument but it is done as a functionality integration level basically, because the intention is to integrate from the lower level modules in the higher level that was the idea. The integration can start with very early in the development process. Why this type is difficult is? The development process we will be done it incrimination of the lower level layers.

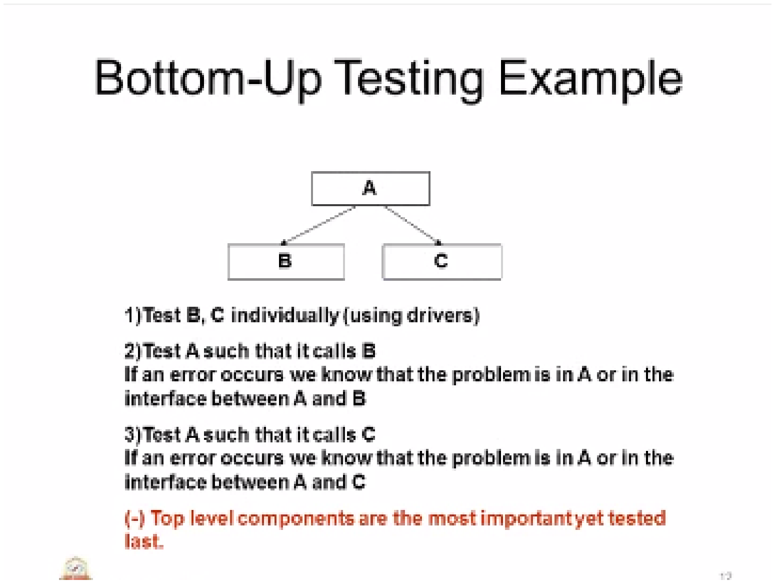
And still the computers integration is not done so without the development previously done in the complete integration of the entire system it may be difficult for us to do the top down integration process. So this can be started as soon as the lower level modules are completed by the development team so that is the good thing about bottom up integration and also it will lead to the integration of interface problems and these problems can be isolated easily.

So before we attack the complete system as entirely it is better to start withal modules and identify the interface issues and the integration issues so that the higher level issues are less, but the disadvantage is that many drivers have to be used in carrying out this strategy and it is TDS you can understand this why because we are attacking from the bottom most layer of the embedded software system where we target that low level modules.

And there numerous low level modules and each modules will have their own dependencies and identify the working strategy testing the each one of them is definitely a time consuming and that is the disadvantage of bottom up integration in kaleidoscope money and resources are available I won’t suggest to follow this type of integration in your system where did not very complexity is somewhat modular complex and,

This modular moderately not very big medium sort of an embedded system this strategy can be used as per, so we refer basically from embedded software testing book from book by mark,

(Refer Slide Time: 43:21)



## Bottom-Up Testing Example

1)Test B, C individually (using drivers)

2)Test A such that it calls B
If an error occurs we know that the problem is in A or in the interface between A and B

3)Test A such that it calls C
If an error occurs we know that the problem is in A or in the interface between A and C

(-) Top level components are the most important yet tested last.

We will take an example of how integration bottom up testing is done, it is basically from the web resource see you take there are two leveled or two layered architecture design is there and A is the higher level component B and C at lower level ho are going to integrate? With the help of bottom up integration the first step is test B, C individually we are going to test it that means first we are going to verify B, next we are going to verify C.

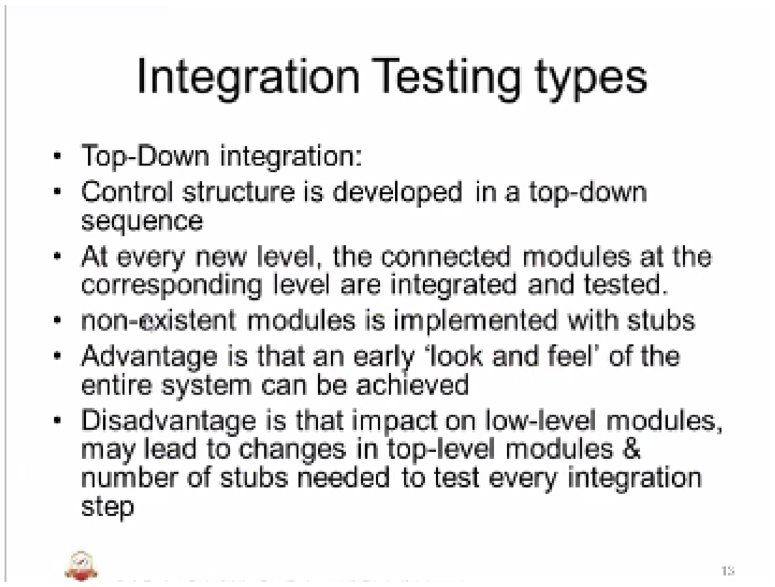
So with the help of driver's means, test drivers for example driver C where you call the C function with parameters and data whatever it is integrated for calling the C and once we attend with this mostly it is done at component level, the next step is test A such that it calls B. So we are going to test it A to make sure that it has to call B, so that we know that the interface between A and B are verified.

Similarly test A such that it calls C, so that the interface between A and C are, so if an error occurs we know that the problem is in A because we already tested B in the first step or in the interface between A and B so the error could be there or error could be left here and we know that it don't have issue with the B because we have already tested with individually at a lower level and now we are again to trust the A.

And we are going to call the test driver such that A is going to call B and A is going to call and if he finds out an error or a fault we are sure that no problem with B but there is an interaction problem as the A is calling wrongly B or B is not getting call properly all that could be identified with the A itself so that is out we bring out the issues in integrating A and B, similarly once you had done with A and B we are going to call A.

Trusting such that any interfaces between A and C are tested along with any issues in the A side so this is how we take care of integration testing from bottom up aspect. So the problem is we are going to test it after the MCR done daily that is what the negative point about this bottom up testing, next we take up the integration testing,

(Refer Slide Time: 47:22)



## Integration Testing types

- Top-Down integration:
- Control structure is developed in a top-down sequence
- At every new level, the connected modules at the corresponding level are integrated and tested.
- non-existent modules is implemented with stubs
- Advantage is that an early 'look and feel' of the entire system can be achieved
- Disadvantage is that impact on low-level modules, may lead to changes in top-level modules & number of stubs needed to test every integration step

13

Top down approach, what do we do with the top down approach? It is exactly the opposite way of doing the bottom up, so what do we do? We start with the top most elements the higher level and reach the lower level so it has its own advantage and it has its own disadvantage what are those strategies? Control structure is developed in a top down sequence so that entire structure of the flow or the sequences that is being used in that software system.

In the embedded software system is being developed first and those will be used for top down integrations sequences test set every new level that connected modules at the corresponding level are integrated and tested that means it started one level that higher up and as if programs we are going to connect the next level, next level, the next level elements or the models one by one and we are going to test.

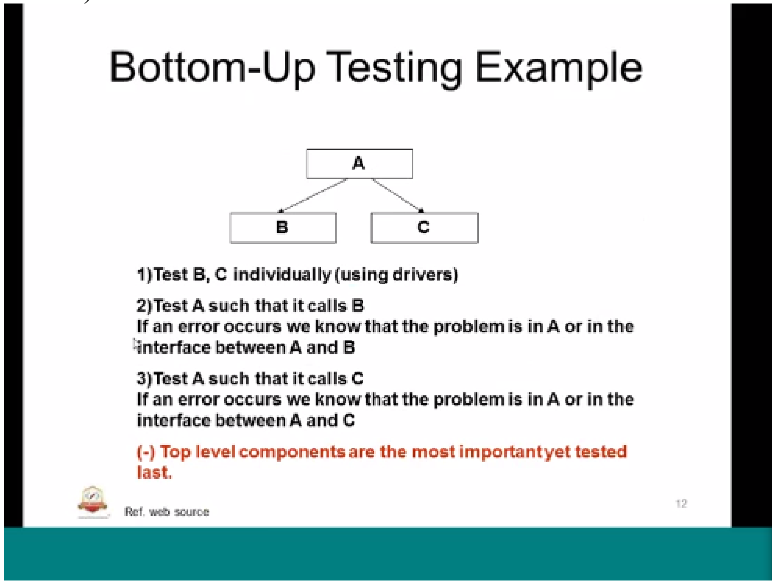
It we are going to integrated and tested non- existent modules is implemented with stubs so if there are as said in earlier slide that we have seen the complete models would not having developed so but still I have the higher level models but those models is stub level models you can stub it and we can test it stubbing is also useful when we are not able to provide a different negative values or the higher level values.

So that system can be modified very tested for the different types of instrumentations advantage is that an earlier look failure of the entire system can be achieved this is what the very important aspect of embedded software testing integrations to understand the system how it behaviors based on the knowledge that we have or the complicity that we understand to better state it test it is completely put the box under the various conditions test form an independent.

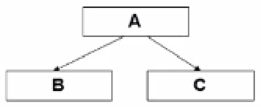
How it is going to behave it as per expected system behavior so and pick up the individuals models, individual functionality and try understand where it is going to less or where it is going to take up the lower level models so that we have a understanding of the system we have a look and feel of the entire system so that is great advantage of top level integrations testing this typically important where we have a larger or very complex embedded system.

So but to huge or to large complex system they basically they can divide also to both conduct both levels of integrations testing they can either both of them bottom as well as top down in that cases what we they did is they segregate the different piece of software accordingly so that is

how they take in a larger complex systems but both of them have to done pretty closely disadvantages is that impact on low-level modules may lead to changes in top-level modules &number of stubs needed to test every integrations step.
So they also stubbing his important why because some of the un implemented modules or some modules integrations to develop with the help of stubs so those stubs will automatically develop more work and they is disadvantage and it may impact the low-level modules because low-level modules have will not have a technique so that could lead in terms of changes in the top level modules so that is the disadvantage so basically we use stubs in top down integrations.
(Refer Slide Time: 51:30)



## Bottom-Up Testing Example



1)Test B, C individually (using drivers)

2)Test A such that it calls B
If an error occurs we know that the problem is in A or in the interface between A and B

3)Test A such that it calls C
If an error occurs we know that the problem is in A or in the interface between A and C

(-) Top level components are the most important yet tested last.

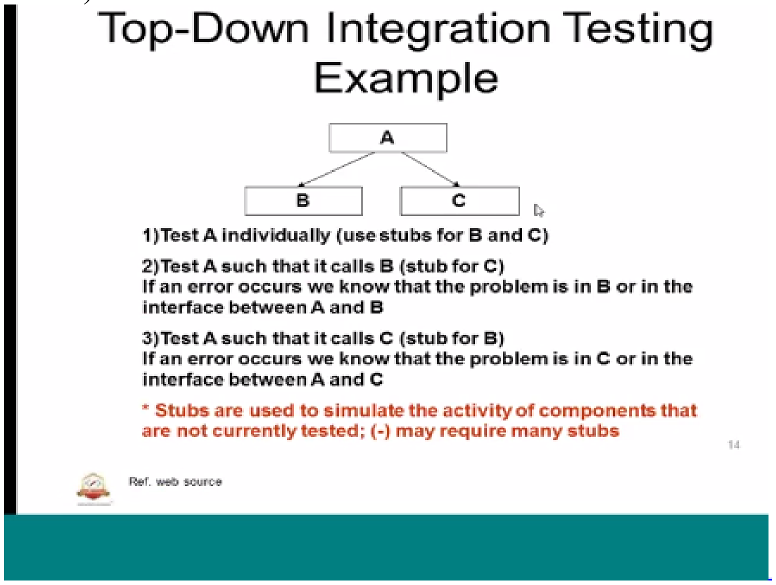
Ref. web source

12

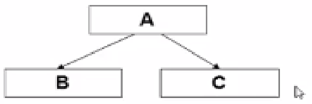
And we use the drivers in bottom of testing integrations okay we will take an example the same example where we have cleaning modules A, B, C with the examples how we are going to test integration tested with the top down approaches okay the first step is test A individually use stubs as a set for DMC that means definitely we stubs because A is use to call in one of the executable that it has the executable line that it has.
So we are not having the B or we do not have the B so we are going to have the small stubs which is similar to actual B and similar to actual C so first we will test AB with the stubs it is fine or not with the help of these systems then we are going to test A so that it calls B stub for C so this is very important because A is going to work along with B and C and it can call B or it can call C each one has to be tested with actual B and the other one is stub when we are testing C the B is stub that is the third step test A sub set it calls C.
So first time it call B and second time it calls C the first they want to have the same stub of C and the second time they going to have same stub for B so that we know that the working behavior or the actual output is same in all END that is what the approach so if an error occurs we knows that the problem is in the or interface between A so we know that while doing the integrations defiantly they could be an interface sections or the issues is definitely with the stubbed part because we have stubbed.
It the IAS is getting code and we are getting the error so if an error occurs we know that the problem will be or it interface between END similarly the third step is that it call C with stubbing

B and if there is an error we know that the problem is in C or it interface between A and C in terms of calling mechanism or whatever it could so this is how all the error of A,B,C and A to B A to C they will bought up so that is what we do or we will bring out the top down integrations issue this is exactly the other way of doing here, here what we do we start the DMC and we identify the interfaces issues either in A or in the interface with the DMC here we give the drivers, drivers at the A levels because it is want to drive DMC.

(Refer Slide Time: 55:06)



## Top-Down Integration Testing Example



1)Test A individually (use stubs for B and C)

2)Test A such that it calls B (stub for C)
If an error occurs we know that the problem is in B or in the interface between A and B

3)Test A such that it calls C (stub for B)
If an error occurs we know that the problem is in C or in the interface between A and C

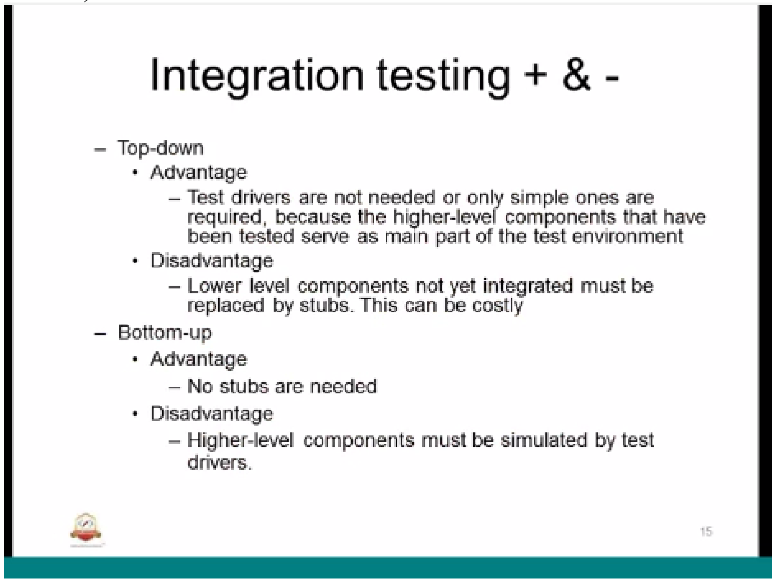
* Stubs are used to simulate the activity of components that are not currently tested; (-) may require many stubs

14

Ref. web source

So in top down it is the other way they are going to do with the help of DMC stubs are used to stimulate the activity of components that are not currently tested a disadvantage is that they require many stubs that is what it is going happen in top down integrations testing is an example okay.

(Refer Slide Time: 55:37)



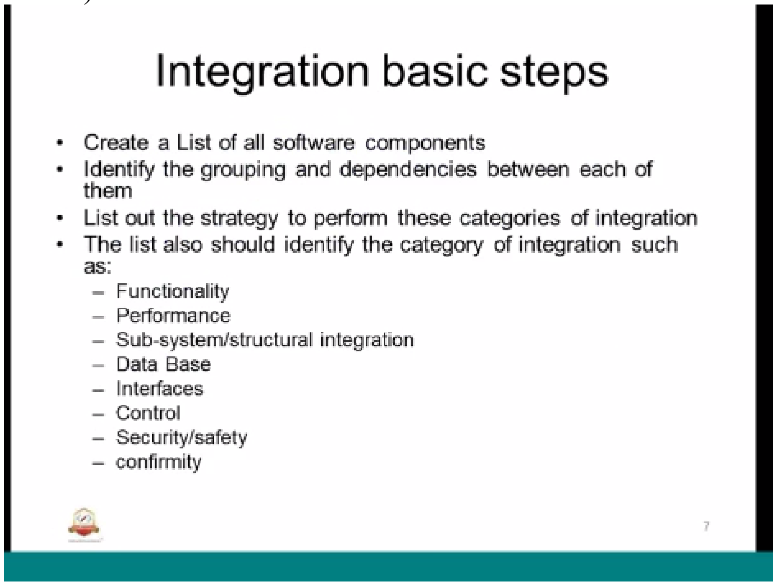
## Integration testing + & -

- Top-down
  - Advantage
    - Test drivers are not needed or only simple ones are required, because the higher-level components that have been tested serve as main part of the test environment
  - Disadvantage
    - Lower level components not yet integrated must be replaced by stubs. This can be costly
- Bottom-up
  - Advantage
    - No stubs are needed
  - Disadvantage
    - Higher-level components must be simulated by test drivers.

15

Over all integrations testing of top down bottom of advantages several based are listed out top down advantage is that test drivers are not need or only the simple ones are required because the

higher level components that have been tested serves as main part of the environment the disadvantage is that low level components not get integrated must be replaced by stubs this can be costly because either is definitely effort and cost involved for developing the stubs by the team and teams has to be B and C for developing the stubs and the bottom advantage no stubs are needed because we are going to drive.
It then we have modules but disadvantage is that high levels components is must be simulated by test drivers so in bottom of we need drivers to down we need stubs so it is up to the test strategy team or the test plan team or test manager to definite what sort of strategy we can go up for the particular embedded software testing it again depends on the various factors.
(Refer Slide Time: 56:53)

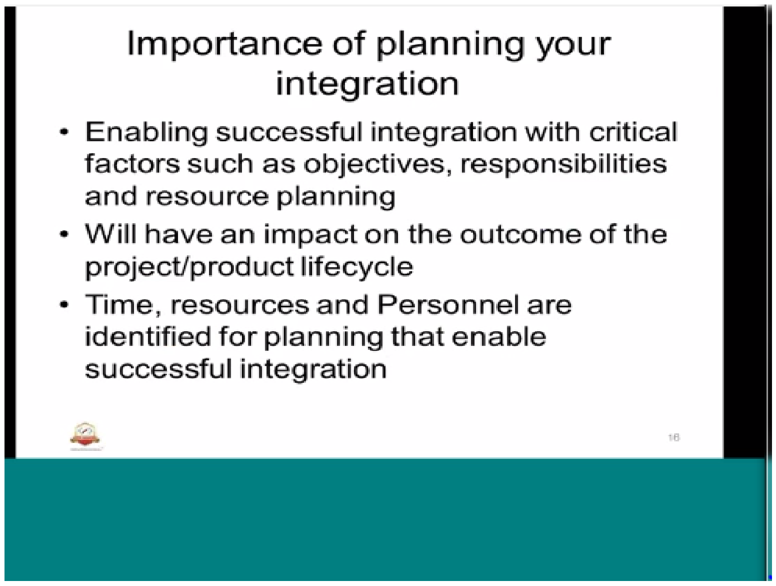


## Integration basic steps

- Create a List of all software components
- Identify the grouping and dependencies between each of them
- List out the strategy to perform these categories of integration
- The list also should identify the category of integration such as:
  - Functionality
  - Performance
  - Sub-system/structural integration
  - Data Base
  - Interfaces
  - Control
  - Security/safety
  - confirmity

7

As I said the complexity functionality, performances, sub system or structure integrations, data base what sort of data base it has whether they have a components that it will work or we need to test of each of the data with the help of all that boundary values and control all the control flow is there for entire embedded systems why are the security aspects or the safety aspects all this need to considered for deciding the bottom or the top down of the integrations we can have both as well does not matter does not impact in terms of standards or the complex both can be taken care okay.
(Refer Slide Time: 57:40)



## Importance of planning your integration

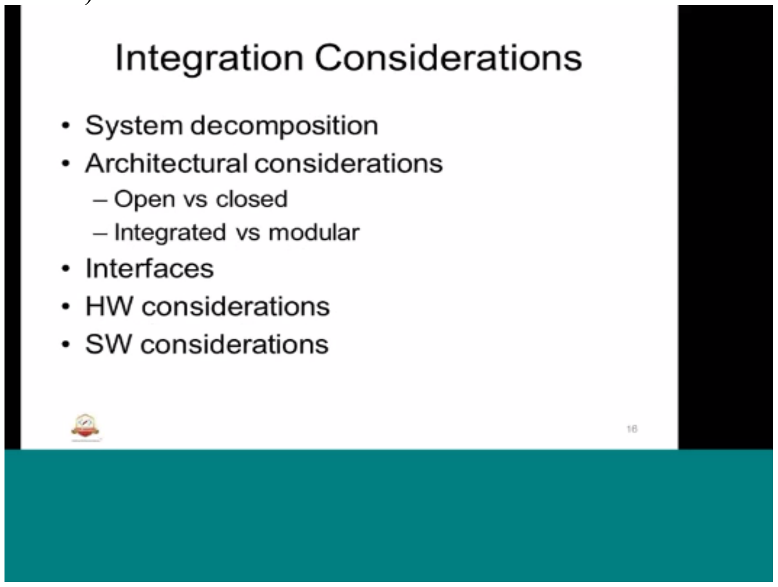
- Enabling successful integration with critical factors such as objectives, responsibilities and resource planning
- Will have an impact on the outcome of the project/product lifecycle
- Time, resources and Personnel are identified for planning that enable successful integration

16

So we need to plan accordingly or identify that so what is the important of planning the integrations by enabling successful integrations with critical factors such as objectives, responsibilities and resource planning so basically it will enable we will have an impact on the outcome of the project or product lifecycle because without integrations we do know what is going to behave in the field or with the customer so definitely it is very important and it has been impact on the total outcome of the product or the life cycle.

So this is the one of the critical path of the product or project life cycle of the embedded software time resources and personal or identify or planning that are enabled that enable successful integrations so as part of the planning of the integrations testing is based on advantages disadvantages types of the integrations testing the time resources planning, personal planning or the tool all this will be identify so without having the integrations we cannot identify this so we need have the planning and conducting the integrations that is where the important of the planning integrations lines.

(Refer Slide Time: 59:05)



## Integration Considerations

- System decomposition
- Architectural considerations
  - Open vs closed
  - Integrated vs modular
- Interfaces
- HW considerations
- SW considerations

16

So integrations considerations what are the aspects that we need to considered for integrations system decompositions how the system is decomposed basically it is drawn from the design or the architecture it could be higher level architecture lower level design then we need to have a architecture considerations in terms of open or close that means the architecture elements can be open or the close in terms of the down wards and integrated or modular that means the modularity is required or the integrated parts.
As whole is required all this considerations we need to take care during the integrations testing and the interfaces how it is going to interfaces externally, internally or sub systems or participations applications etc. then we need to understand the hardware considerations we need to know what sort of hardware is used whether it is untargeted or it is a evaluations board all this gave to be considered and tools of course that tools can be having a hardware independences likewise.
We need to have this is very much important particularly in a figure systems here we have an automatic test equipment it is so hardware considerations is one of the important element of the integration testing HSIT it is called as that is hardware software integrations testing software considerations where we need to considered the software components such as algorithms there could be a check sums or it could be a stringent completions all this have to be considered so that is where we do the planning or the integrations software testing okay so we will continue in the next class in terms of integrations test strategy and the other aspects of the software integrations testing