

Welcome to the next session of embedded software testing, this is a unit four serious of lecture series, today is lecture 2. We will study and discuss more on embedded software testing software integration. And we will try to see that test strategy and its benefits.
 (Refer Slide Time: 00:26)

Integration Test Strategy

- Ad hoc strategy is to integrate the components in order in which they are ready
- As a component has passed the component test, check if it fits with another already tested component, or if it fits into partially integrated subsystem
- If so, both parts are integrated and tested
- Need to write stubs to help in integration
- Stub is a skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component

And comparison of different,
 (Refer Slide Time: 00:28)

Integration Strategies Comparison

TABLE 8.3 Comparison of Integration Strategies (Myers, 1978)

	Bottom-up	Top-down	Modified top-down	Big bang	Sandwich	Modified sandwich
Integration	Early	Early	Early	Late	Early	Early
Time to have working program	Late	Early	Early	Late	Early	Early
Component drivers needed	Yes	No	Yes	Yes	Yes	Yes
Stubs needed	No	Yes	Yes	Yes	Yes	Yes
Work available at beginning	Medium	Low	Medium	High	Medium	High
Ability to test particular paths	Easy	Hard	Easy	Easy	Medium	Easy
Ability to plan and control sequence	Easy	Hard	Hard	Easy	Easy	Hard

Strategies,
 (Refer Slide Time: 00:31)

Integration Strategies Comparison

Top-down Testing

Advantages

- If major defects are more likely at the top level modules top-down is beneficial.
- Getting I/O functions in early can ease test writing.
- Early demonstration of the main functionality can be helpful in highlighting requirements issues and in boosting morale

Disadvantages

- Too much effort on stubs.
- Stub complexity can introduce errors.
- Defining stubs can be difficult if some code is yet to be written.
- It may be impossible accurately to reproduce test conditions.
- Some observations may be impossible to make.
- Encourages the idea that test and development can overlap.
- Encourages deferring full testing of modules (until lower level modules are complete).



Then top down bottom up of advantages disadvantages.
(Refer Slide Time: 00:36)

Top-Down Adv. & Disadvantages

Top-down Testing

Advantages

- If major defects are more likely at the top level modules top-down is beneficial.
- Getting I/O functions in early can ease test writing.
- Early demonstration of the main functionality can be helpful in highlighting requirements issues and in boosting morale

Disadvantages

- Too much effort on stubs.
- Stub complexity can introduce errors.
- Defining stubs can be difficult if some code is yet to be written.
- It may be impossible accurately to reproduce test conditions.
- Some observations may be impossible to make.
- Encourages the idea that test and development can overlap.
- Encourages deferring full testing of modules (until lower level modules are complete).




Ref: Graef Anderson - web search

Integration testing other types also will be discussed.
(Refer Slide Time: 00:39)

Top-Down Adv. & Disadvantages

Bottom-up Testing

Advantages	Disadvantages
<ul style="list-style-type: none">• Helpful if errors are likely deep down in the dependency structure (e.g. in hardware specific code).• Test conditions are easier to create.• Observation of test results is reasonably easy.• Reduced effort in creating stub modules.	<ul style="list-style-type: none">• Need to create driver modules (but arguably this is easier than creating stub code – and tools like JUnit help).• The entire system is subjected to the smallest amount of test (because the top modules are included in the tests at the final stage).

 Prof. Grant Anderson – web extras 5

And before that I just would like to have a recap and add some more points to what we have discussed in the last session.
(Refer Slide Time: 00:50)

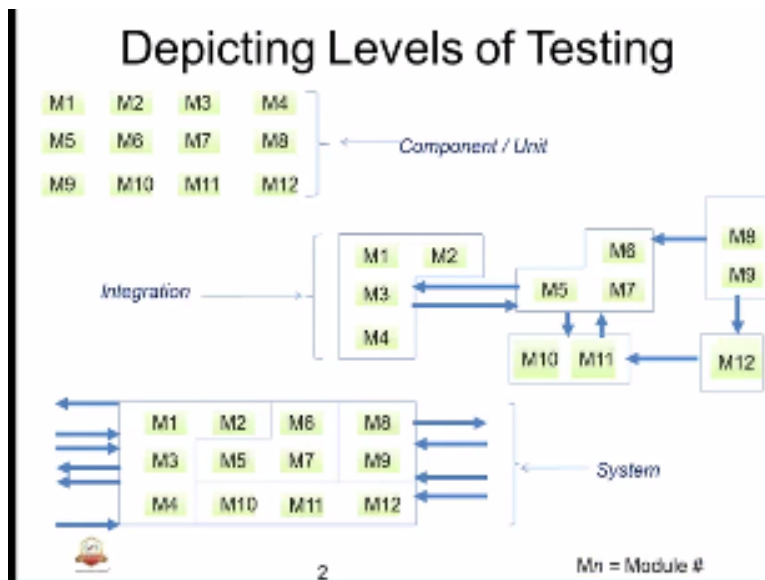
Embedded Software Testing

Unit 4: Software Integration

Lecture 1
Seer Akademi – NPTEL MOU

 1

That was integration we know that three levels of testing we can do.
(Refer Slide Time: 00:58)



Component integration and system, so basically we do individual components and those are under test the unit testing. Next one is the integration testing where we combine logically different components based on the complexity of the feature wise or functionality of those components, then we will interact with the other similar logical or interactive groups to make sure that those are getting integrated and the behavior is verified.

The last type will be the system testing where black box everything will be tested. So this will basically what we do? We defined the test objective for the intension test, what qualifies the system we want to verify, then divide the test cases, program test and executive test and analyze the aspects we will follow for testing.

(Refer Slide Time: 02:04)

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

3

Then we had gone through integration testing that means the process of aggregating different components to create larger components, so that larger components can be tested together with similar larger components. Integration testing done to show that even though components were

individually satisfactory the combination is incorrect and inconsistent, the combination is very important you know just enough to test at a component or individual level.
(Refer Slide Time: 02:34)

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



Ref: Testng. Eng. SW by Bert Brorsson & Edwin Jansenboom

8

So we also had gone through definition of what is integration? So basically it is not enough to have unit testing alone to get stable or functional or correct system, so many defects are related to the integration of modules, if the requirements are not formally described everyone has to make their own interpretation of those requirements basically. So this is not a problem as long as these interpretations are not related to interactions with other modules.

What incorrect interactions between modules C is often calls by these interpretations, this is the problem actually. So what is the best instrument to detect that is with the help of integration facility. So physically integration strategy is a decision about how different modules are integrated with a complete system. The integration covers both hardware and software as I said in terms of hardware it is kind of hardware software integration,

And it is called as software integration it is basically the different software modules and their interactions will be verified, and system integration, system hardware integration and system software integration.

(Refer Slide Time: 04:02)

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



The other term of embedded system integration that defined the task of creating a properly functioning system from its constituent components could be hardware, fir ware, software. where are in system hardware integration are the components wired together correctly, that means especially hardware components are tied properly in the system and those components are tested against their behavior.

And the last part is system software integration, typically it assumes hardware integration is largely complete no issues with the hardware, but before we deliver or accept the product software modules will be verified with the hardware how it is behaving and all that that is what we do with the system software and the integration.

(Refer Slide Time: 04:55)

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
 - conformity



So integration steps basically we need to create 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 could be based on the aspects like functionality, performance sort of system integration or interaction, databases the kind of database we use and what

interfaces specially from the signal prospective all which is derived then the control of the various program elements, then safety and security aspects and conformity.
So basically the goals of integration is,
(Refer Slide Time: 05:58)

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



To expose faults in the interface 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. So the faults could be in terms of interface formats of the memory or the file which is not compatible where are it is working individually like when you interface with a external memory or any file handling it could be a issue similarly that data exchanges are not properly there type casting with the issue when it fix with the subsistence always part of a interface issues and co ordination or cooperation problems within the modules will be revealed.

(Refer Slide Time: 06:54)

Embedded Software Integration testing

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




And we have types of integration as big bang integration, bottom up integration, top down integration I mean how the approach to the integration is been done, some time component integration test is also called as integration test that is subsidiary as small integration test. System integration test is also called as integration test in the large system. Where the external systems are more involved so it defines on the complexity of the system.

(Refer Slide Time: 07:30)

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 Fundamentals by Neil Broome & Felix Johnston


So in big bang is a simple or type testing type all modules are integrated and the system as tested as the home basically, so completely the system largely N system which is enough to test it that why it is going to be banged up from the testability prospective. The main advantage of that is no stuffs or drivers as we have seen required for this with this strategy the problems is it difficult to find the cause or defects and the integration can only start if all the models are available and it is not like a stepwise part of the thing so basically it is coupling with entire system.

(Refer Slide Time: 08:23)

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



Similarly as we have seen after all components are unit testing we will test the entire system with all its components in action
(Refer Slide Time: 08:37)

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: First SW by Bert Brorsson & Edvin Isaksson

11

The next type of integration testing is the bottom up integration testing this strategy useful for many of the systems it starts with low level modules with least number of dependencies. Basically it use the drivers, test drivers it is called, if the integration can start with early in the development process you don't want to wait for the all the modules to be completed. As in when the modules are there we can write the test which is called the integration test for the completed one and instrumentally we can close all the modules, basically advantage is that will lead to an early detection of interface problems and these problems can be isolated very easily disadvantage is that may have to deal with many drivers and we have to use it for carrying out different strategies and time consuming.

(Refer Slide Time: 09:43)

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.**

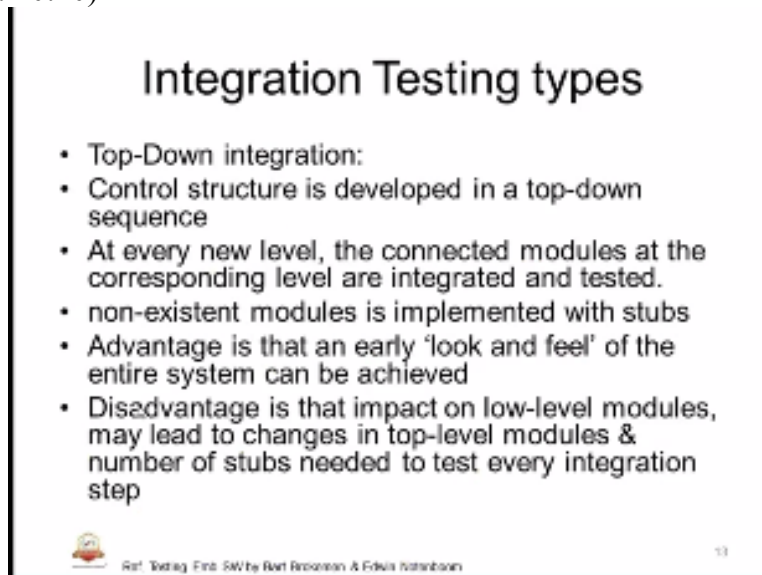


Ref: web source

12

And we also had gone through a bottom up integration testing example where higher level module J and lower level module B and C are there, so basically we develop the test drivers for B and C so we test B and C usually then we test A such that it calls B, so if any error we know that the problem is in A or in the interface between A and B. similarly we test A and we develop the driver since that it calls the C and it will be ever that has to the A or there are could be B and C.

(Refer Slide Time: 10:40)



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

Ref: Testing Fundamentals: Robert Rossman & Fredrik Holmstrom

The next type of integration testing is top down integration testing in this strategy the control structure of the system takes lead basically the control structure is developed in top down sequence and this offers the ability to integrate the modules and stop down starting with the higher level control module, that ever new level the connected modules at corresponding level are integrated and connected.

As we go down the connection modules will be tested one by one we roll up there could be some non existing modules also those can be exist with the stubs here we will call as stub. And driver we use in bottom up and test ups we use it in top downs, so advantage is that look and feel entire system can be achieved, that means user has an idea of what are the top level modules how it is accepted? And how it is getting flow down into individual model?

That is the advantage of taking up to level integration, disadvantage is that impact on low level modules may lead to changes in top level modules we don't know what is the impact? And you may have to wait for all the modules to be completed you can now come to a progressive kind until otherwise the top level module is done, so that some if any may not too low level modules are not come to be tested so we need a number of stuffs for every integration style.

(Refer Slide Time: 12:31)

Top-Down Integration Testing Example

```

graph TD
    A[A] --- B[B]
    A[A] --- C[C]
  
```

- 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

Ref: web search 14

So we also have to go through an example of top down integration where ABC modules are here higher level B is lower level, so A will touch it individually with the help of stubs or B and C, test A such that it calls B stub for C and if an error occurs we know that is a problem in the B or in the interface between A and B. similarly while we are testing C stub B so that all together we focus on C and if an error occurs that the problem has to be in C or interface between A and C. (Refer Slide Time: 13:16)

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.



So the advantage and disadvantages of integrating testing top down we will detail out in today's section, we also know that test drivers are not been made only simple ones are required as advantage because the high level components that have been tested serve as the main part of the test environment. So we don't have a test driver we have only have a top level module, disadvantage is that low level components not yet integrated must be replaced by stubs because we have not done it or it is not integrated, so this can be expensive because we need to develop the stub .

Bottom up advantage is that no stubs are required but disadvantage is that higher level components must be stimulated by the test drivers. So user will not have any idea about our top level modules look like how we can approach the integration testing? That is the disadvantage. (Refer Slide Time: 14:30)

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

And also the importance of why we need to plan the integration so basically successful integration with critical factors such as objectives, responsibilities and resource planning so that we know at what stage what other type of integration we are going to do and how the qualisiveness is going to be achieved. So basically the product of the life cycle or product lifecycle outcome is purely based on the integration.

Form the customer asks track or progress of development he will similarly asked the progress of integration, the first question I will ask you is are you able to integrate in first in the modules that we have developed how is it going on? What is the performance of integration testing? Whether you are able to integrate all the components because we always interact with the hardware and the actual system how is the hardware software integration going on.

How software modules interaction behavior is going or the same? So how to start of an things is an outcome is the integration testing, so definitely there is a need of plan for this aim because it has lot of stake holders in today researchers, time, personal which are going to be identified for planning that enable successful integration.

(Refer Slide Time: 15:59)

Integration Considerations

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

Similarly integration consideration that we need to have is that we need to have an understanding of actual system decomposition, architectural considerations could be open or closed architecture, integrated or modular integration depend on the type of system that we are trying to do definitely the architecture of high level that needs to be considered for integration, and also we need to have an understanding on the interfaces.

All the interfaces are associated compare types of one lords or discrete that have been involved all this will be part of interfaces signals, the next one is the hardware considerations in terms of hardware limitations or it could be anything so we need to have a idea of the hardware how it is built? Because without the hardware we cannot be test many of the software modules because title coupled will be hardware.

Last is being the software considerations you know that the software modules are to be very cursive so that they can be tested appropriately on the hardware, some examples I will try to provide interfaces that you are clear about sort of a used the interfaces could be memory port such as rs232 or can or spiel are also examples. Next we have our timer, next we have our power interface this is also very important.

Where some of the power of requirements had to be pattered such as it is so and so time the budding procedure has to be completed and system should be operated in sort of time so there we will look in the some of the reaches or any interrupts based on the power that also need to be considered for interfaces, of course we have analog interfaces, discrete interfaces discrete could be switches, signals, led all this will be discrete interface.

Then what are interfaces? It could be just like not a convertor of all it is gain this depends on occur on a drive we have it could be two port could be a parallel bars or it could be normal of interface basically the in target is the motor so how we are going to interface with it actually, so it very important for the us to understand the system is motor based close look control system definitely it tend to have a Quinter mechanism and understanding by the system testing, of course we have the AC, AC motor interfaces sub title interfaces, so hardware related interfaces you know that the hardware input output need to be used in the system integration test.

So that is also very important, software interfaces in terms of like any algorithms check some computations all this will be multiple all this things will have a multiple comprehensive mode so definitely we will consider the software aspects of the integration that is very important, so (Refer Slide Time: 20:50)

Integration Test Strategy

- Ad hoc strategy is to integrate the components in order in which they are ready
- As a component has passed the component test, check if it fits with another already tested component, or if it fits into partially integrated subsystem
- If so, both parts are integrated and tested
- Need to write stubs to help in integration
- Stub is a skeletal or special-purpose implementation of a software component, used to develop or test a component that calls or is otherwise dependent on it. It replaces a called component



11

The last part which we had studied in the session is that integration test strategy, ad hoc strategy is to integrate the components in order to which they are ready so this is not the good idea but you can do it you can afford to the beginning of the program, as a component has passed the component test check if it fits with another already tested component and or if fits into partially integrated subsystem.

You need to see whether the component you need to worry where it can fit? Whether it can be tested? First we need to identify the component associated integrated subsystem a component where it is going to fit and then accordingly the associate 1 needs to be addressed to make sure that both are integrated, need to write stubs to help in integration stub is a skeletal or special purpose implementation of a software company

Used to develop or test a component that calls or is otherwise depend on it, it replaces a called component so it is of use the other the form of stub where we have seen in the top norm is drivers where the high level components are going to drive the low level in a bottom up strategy. So that was about the previous session, now we will go through the integration test strategy in detail we know that stubs and drivers are needed for integration test so it is nothing wrong in having both actually again it is subjective.

We have to take a call depends on the complexity of the embedded software system elements we can have bottom up as well as top down approach. So it will again dependant for larger systems where we have means of code definitely we need to divide the entire system with a several systems those each several systems can be addressed in different strategies in terms of integration.

But all that have to be planned appropriately either it could be bottom up or it could be top down or some sort of we don't need to have integration at the top down or bottom up we can go for the

big bang integration process entire group of modules can be tested all together that strategy also we can adopt that is in subjective. So we have gone through that various types of integration strategies.

So we will try to compare this based on the book mayors 1779, so that table basically list out different types of integration in strategy we will not focus on those and reach and modified and reach type of embed integration testing so these are other type but where we have used or it is not required to be understood in this point. So you can see the features that are listed in the left side of the table and on the right hand side we have different type of integration strategy bottom up, top down, modified top down, big bang.

So basically it identifies integration when it should be done so bottom up early we have talked on the modified top down early, bug bang the latest stage because all modules have to be completed, time to basic working program very late bottom up early, early late so here to have it a matured working program the bottom up will be related stage whereas in top down early stage you can identify the understanding of the working system.

Component drivers needed for top down we need sorry bottom up we need not for the top down modified top down can also have test drivers or component drivers which are also called, big bang processes have drivers stubs we need we know that needed only for the top down approach not for the bottom up, so work parallel in the beginning there issues at the medium bottom up top down below modified top down as medium.

And big bang is very high so that means parallelism need to be worked out ability to test the particular parts what you mean by difficulty so bottom up is easier and in top down it is very hard and modified top down it is easy and big bang it is easy. We need to path plan and control sequences are easier in bottom up because we have the control on the smallest of the components of the entire system and it is very difficult for the top down.

We don't know that exact sequence and we need to go through each of them big bang it is easier so these are some of the matrix they have compared for integration strategies, modified top down is something like a mix of both top down and big bang so they adopt some of the medium size complex embed systems that is where they use top down testing. So that is about the strategies of different integration types.

(Refer Slide Time: 27:18)

Top-Down Adv. & Disadvantages

Top-down Testing

Advantages

- If major defects are more likely at the top level modules top-down is beneficial.
- Getting I/O functions in early can ease test writing.
- Early demonstration of the main functionality can be helpful in highlighting requirements issues and in boosting morale

Disadvantages

- Too much effort on stubs.
- Stub complexity can introduce errors.
- Defining stubs can be difficult if some code is yet to be written.
- It may be impossible accurately to reproduce test conditions.
- Some observations may be impossible to make.
- Encourages the idea that test and development can overlap.
- Encourages deferring full testing of modules (until lower level modules are complete).



Prof. David Anderson

1

Now we will try to go through top down advantages and disadvantages basically we have to called up to an in terms got from the web it is basically shows the advantages and disadvantages such as in top down the major defects are more likely at the top level module top down is beneficial that means we have more issues in the top level then it is going to be advantages, getting a input output functions merely can is fascinating.

That is as long as we have developed the more input output functions obvious to you it of the approach is the beneficial, the top down approach is beneficial it will use out the integration strategy of top down so early demonstration of made functionality it will be helpful in highlighting requirements issues and in boosting the morale. You have more clarity and idea of what the system is defining and your understanding will be better.

As you progress on the integration testing of the type top down approach because you start with the main functionality of the top level modules, now let us look in to the disadvantages of top down it is too much for the stubs you know that top down stubs if low level modules are not ready to not enough stubs the may have to leave it with test stubs for the low level modules and also another problem is by developing the stub.

If it is more complex it can inject some additional issues or in to these errors are disadvantages, defining stubs can be difficult in some code in yet t be done so we are unclear about what exactly is going to be integrated but we are in the knowledge of this functionality based on that we are going to help this stub but sometimes it is difficult to develop a stub. Here is the code is to be written it may be impossible accurately to read word is test conditions, if there are failures suppose we started top down testing and we founded that there is a issue in the system.

It is very difficult to figure out or isolate where the test issue is there and how to reconstruct that so some issues like intermittent issues are very difficult, so that is disadvantages of top down testing. Some observations maybe impossible to so we cannot observe some issues of the top down testing modules encourages the idea that test and development can overlap. That means, the top down testing have a close relationship in the within development aspects, So basically it encourages the ideas that understand development can overlap encourages differing full testing of modules because low level modules are not computing so we are leaving

the high level modules and basically we may not be completing the integrity process it is partially done partially passing so of them low level modules are to be explode so thing are difficult in the top down integration testing.

(Refer Slide Time: 31:38)

Bottom-up Adv. & Disadvantages

Bottom-up Testing

Advantages	Disadvantages
<ul style="list-style-type: none">• Helpful if errors are likely deep down in the dependency structure (e.g. in hardware specific code).• Test conditions are easier to create.• Observation of test results is reasonably easy.• Reduced effort in creating stub modules.	<ul style="list-style-type: none">• Need to create driver modules (but arguably this is easier than creating stub code – and tools like JUnit help).• The entire system is subjected to the smallest amount of test (because the top modules are included in the tests at the final stage).

Ref: Graet Anderson – web source

The next one being the bottom up testing I will correct it quickly so what we do here is we know that test drivers are needed for the bottom up testing and advantages are such that helpful in errors are likely deep down in the dependencies structure, if the system implement of the thing that it is more complexes at the deeper level especially on the device drivers of the modified top down then it is better to take up this activities.

It is going to be advantages the bottom up integration testing, rest conditions are easier to create basically you are lowest of the modules to understand and create facility, observation of test results is based on the easier that the ideal effort in creating stub modules you don't have issues in creating the stubs, disadvantage I

that need to create driver modules but arguably easier than creating stub code and tools like some j unit help but basically driver module we need to create so you need to understanding of that, that is a so it is easier in creating a stub because we can see the code and we van drive he values what we need to develop the driver. The entire system is subjected to its smallest amount of test it means we are focusing only on the smallest part of the entire system.

Because the top modules are included in the tests at the final stage we are not addressing at the higher level incrementally we are testing one by one on the bottom up low level modules and as we go in to the higher level we are going to test at the high level tests

(Refer Slide Time: 34:15)

Integration testing other types – Combination / hybrid

Hybrid Strategies

- It is clear that judicious combination of stubs and drivers can be used to integrate in a middle-out approach.
- Also for some groups of modules we may want to take a non-iterative approach and just consider testing them all at once (this means we choose a bigger granularity for our integration steps).
- Using such approaches there are a range of potential criteria for deciding how to group modules:
 - **Criticality:** decide on groups of modules that provide the most critical functionality and choose to integrate those first.
 - **Cost:** look for collections of modules with few dependencies on code lower in the dependency graph and choose to integrate there first. The goal here is to reduce the cost of creating stub code.



Ref: Grant Anderson – with source

5

So those are the advantage and disadvantages of the bottom up and top down integration testing, now there are other methods of integration testing of the touch basis what are those? First on being these are all basically referred from sources how they are aware from the industry in terms of defining term or creating plus every industry typically different than initiate than actually being lot of based on the weird in complexity skills and compartments rules so many aspects are there.

All these are to be considered as referred to normal as types based on that they will decide and they will define a name for the type of a integration testing strategy, next type of integration testing is hybrid testing and it is also called as a combination type combination integration testing, so it is clear that judicious combination of stubs and drivers from the use to integrated approach.

It may not be purely positive to stubs on the you can take integration or may not be possible to use a driver alone for the inside drivers aspect sometimes we have to mix up both based on the complexity in the kind of system that has been integrated and tested. so we need to give a judicial approach to combination of this we need to take care of this based on the tip boxes to that we use.

So that is called high build test strategy which is this both, as of for some groups of modules you may want to take non iterative approach and just consider testing that all at once which means to chose the big bang for our integration process, some of our groups of the modules may not be able to take up to the right relative to the bottom up and we can take separately as a one shot testing.

But when we are doing the other dependant modules those kind of the things also can be done depending on the type of modules that we have so this concerned a hybrid test strategy, so using this approach there are various of potential criteria for how to group those modules so basically hybrid test strategy is useful process other set grouping is an very important aspect in integration testing.

We are grouping based on the pictures, requirements punctualities, performance etc so there are different criteria that we need to follow for grouping the various modules, so what are those?

One is criticality you said on group of modules that provide the most critical functionality and chose to integrate those part first that is we identify critical groups first which are very important to testing very important part of the entire embed system.

First you broke it then rest of them will fall that is one of the deciding factor when the cost this is also equally important look collection so modules dependencies on code low in dependencies have and choose to integrate first, that goal here is to review the cost of creating stub code, so that what will happen is we cannot be doing stubs because it is going to add a cost so integration is based on how much cost for the front office stuffs.

So if we're going to be too much then we need to collect such modules which can have a many less dependencies on lower part of the code so that the stub mode we don't need to have much so that the past is appropriately taken care these two are very important here cost means no we speaking of dollars is basically we have fault of course at the end of the day all it matters the money.

But we look for the efforts how much efforts it is going to take and complexity or criticality we saw some of the important deciding factors for identifying the test strategy like bottom up and top down they have their own advantage and disadvantage where you have a system having complexity is my suggestion such as close look, and more interfaces and larger systems outside moderate large this kind of systems I suggest have a testability where it bases of both test stubs and drivers.

Test stubs and drivers means top down as well as bottom up, so both being followed coverage and are all were second aspects once we have that integration don with a satisfactory test coverage and test reports for each of the modules automatically coverage process takes place and wherever the gaps are there and you can fit in to any of the test strategies that are any system strategy all can be taken.

It is basically we need to address the entire system from this aspect so that is where the hybrid test strategy comes in to play before winding that we need to understand bottom up and top down so let we have an appropriate mix of test drivers and stubs so that is why I think forward (Refer Slide Time: 40:53)

Integration testing other types - Centralized integration

This type of integration is used when:

- a central part of the system is necessary for the rest of the system to function (for instance, the kernel of an operating system)
- the central part is necessary to run tests and it is too difficult to substitute this part with a stub;
- the architecture of the system is such that the central part of the system is developed first and made ready for production. After that new modules or subsystems are released to upgrade the system or add completely new functionality.



So there are other types of integration which we look centralized integration this type of integration is issued when central part of the system is necessary for the rest of the systems t function that means without this central part nothing works that is heart of the system. For example OS or a kernel so definitely we need to first address's the kernel or the operating system first that is centralized.

And surrounding is centralized part for our sub systems which can be tested subsequently, the central part is necessary to run the but it is too difficult to this part with a stub, so you cannot replace the OS with another stub OS has to be there or the OS won't be there for testing so that is the central part of the program of the entire embedded system, the Accenture of the system is such that if central part of the system is developed first and made ready for production.

This kind of systems what they do is first they develop the kernel of thebe schedule the lower of the OS and surrounding that they will increment the developed subsystems, so that is how the architecture will be headed first so after that new modules or the subsystems are released to upgrade this system or to add completely the updated functionality of the new functionality.

So that is how centralized integration is going to be done, but here also again centralized system we will have subsequent close up system we can adopt either top down or bottom up or hybrid whatever it depends on large the system is ? Or how complex the system is? So it is very important.

(Refer Slide Time: 42:58)

The slide is titled "Integration testing other types" and contains a bulleted list of integration types. At the bottom left, there is a small logo and the text "Testing Embedded Software" and "Testing Embedded Software by (Diet Brostman and Edwin Norderhorst)". At the bottom right, there is a small number "8".

- Layer Integration
- Client/server Integration
- Collaboration Integration
- ...

So there are other types of integration testing just we will discuss one or two line for this here integration the strategies used for systems with structure where you have libraries mid layer lower level layers where the systems are there so this type of integration testing they use it so interfacing only occurs between the layer directly below and the above so don't talk about bottom up or top down sort of things.

But each layer is tested in isolation using the top down or bottom up or big bang strategy so next step is the integration of the layer according to a top down or a bottom up order the advantages and disadvantages are same of top down and bottom up strategy the integration and isolation of

the interpretation is easier and therefore discover the causes of defects is also easier because we have addressed the layers.

So based on the layer s different other layers are going to be integrated in testing. Next type is the client server integration this strategy is used for client server architecture where you know that database and the server and client will have all sort of things embedded architecture is going to have this strategy will have more appropriation, the client is integrated either top down, bottom up or big bang the server is substituted by your stub and driver.

We can use stub and drivers to repair the server for the server the same approach is used for testing server itself like top down bottom up and stubs and drivers are developed for the client, finally server and clients are integrated collaboration integration so basically collaboration is a collection of objects which work together to achieve a common purpose or in sense realization of useful means studying of useful and test cases carried out of test cases they are test results, the system supports many awkward because it has to reveal many schedules many objects belong to more than one collaboration because we have lot of collaborations there can used in many scenarios.

So basically the collaboration is the focus here so choice of the collaboration will become the necessity in terms of covering the computer system that is also collaboration integration I done, so basically they use for object oriented systems or object based systems and which fully covers the component and interfaces where the collaboration is made, this also have advantage and disadvantage similar to top down and bottom up.

Where are interfaces are not clear dependencies are low something like a big bang sort of a thing because most f the components are collaborated together and only when the collaboration is complete so integration testing start big bang we can call it this collaboration kind of testing, but it does not require full sort of integration facility and the new tests are enough actually for the end to end functionality.

And the overlap of the collaboration pictures so you just say I am very subjective again to chose what sort of a integration testing types. so we have studies bottom up, top down, big bang these sort of type of integration testing's top down , bottom up, big bang, hybrid, centralized, these above fiver are very important most of them are used additionally we can use layer, client server, collaboration, integration.

So likewise many types these are the main type of the integration types that can be looked into, next once we have identified the integration all these types of strategies integration next step is to identify the environment form the integration models.

(Refer Slide Time: 48:50)

Integration Test environment, integration models

- Mix of component (unit) level testing & System testing
- Need of test drivers
- Can reuse test drivers that were used earlier for component testing
- Additional tools, called **monitors**, are required to read and log data traffic between components



9

How we are going to have so basically for integration test environment it is mix of components unit level and system testing it is both it takes care write so definitely we need to have an environment which supports both unit level sort of environment and system level, so we will have an need of a test driver perhaps we can reuse the test drivers that was loosed earlier for component testing.

That is why we told that these are component testing because for doing the unit level testing we probably developed some test drivers those test drivers can be reused for integration testing, so additional tools like monitors where we require to read and log data traffic between components and all that basically why we are interested is so interface between these components and while doing the interface definitely we need to monitor or the data is getting interacted.

So to understand the flow of data we need to have test environment identity different tools such as monitors.

(Refer Slide Time: 50:06)

Integration tests environment

- For the software unit (SW/U) tests and the software integration (SW/I) tests, a test bed is created that is comparable to the test environment for the simulation model.
- In the prototype stage the test object is an executable version of a software unit, or a set of integrated software units, that is developed on basis of the design or generated from the simulation model.



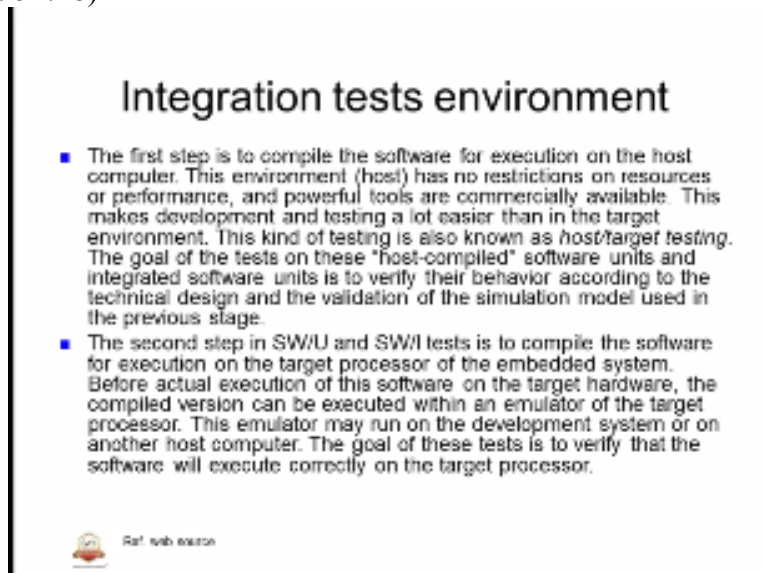
Ref: web source

So integration tests environment before the software unit and the system integration a test but it is created and is comparable to the test environment for the simulation model that means a word testament to specify or testament which will have an environment which is going to be used for unit as well as the integration process identify all the interfaces and higher modules and necessary imports which can be compared with the simulation of the test environment.

In a prototype stage the best object is an executable version of a software unit set of integrated software that is developed on basis of the design or generated from the simulation model, so the development could be based on the model or could be based on the code so we can develop a prototype we can develop the integrated software based on the prototypes that have been developed so far.


For the test environment and that prospect execution or executable version is will be used by the integrated software so that integrated software are basically developed this in the basis of the design of the architecture of the embedded system, that is how the test environment test bed is h=]going to be formed for the integration.

(Refer Slide Time: 51:48)



Integration tests environment

- The first step is to compile the software for execution on the host computer. This environment (host) has no restrictions on resources or performance, and powerful tools are commercially available. This makes development and testing a lot easier than in the target environment. This kind of testing is also known as host/target testing. The goal of the tests on these "host-compiled" software units and integrated software units is to verify their behavior according to the technical design and the validation of the simulation model used in the previous stage.
- The second step in SW/U and SW/I tests is to compile the software for execution on the target processor of the embedded system. Before actual execution of this software on the target hardware, the compiled version can be executed within an emulator of the target processor. This emulator may run on the development system or on another host computer. The goal of these tests is to verify that the software will execute correctly on the target processor.

 Ref. web session

so the first step is to compile the software for the execution basically build the process of the integration test environment, so step is to compile the software for execution in the host computer, the environment has no restrictions on resources of performance and powerful tools are commercially available basically we need to compile and post first because we have developed the code in host we need to compile based on the standards this makes development and tracking a lot easier than in target.

We have to show that we cannot afford to have on the scratch from the target only first host level will get complements by compile and collecting all this thing formation and install this and all so this kind of testing is also known as host kind of testing first we have to develop the host environment and try to test as much as the host done will be target larger system, the goal of this test tests of this hosts compiled software used integrated software used and verified according to the technical design and validation and simulation module used in the previous stages.

Incrementally starting point is validated there is all the simulated environment that is done first and the second stub is software use or software integration is to compile the software for execution on the target processor of the embedded system, so far we have done on the post target based environment I think we have studied in detail simulator emulator and all that similar to that same strategy but here the focus is all the interfaces integration of different modules.

So second is basically we are going to develop for the target and before actual execution of this software the complied version can be executed and emulated of the target processor, you know that there are emulators and with the emulators we know we can assume the code behavior and the function of the code and all this at the system before actually we execute the simulator may run on the development system or on host computer, so the chances of the emulator can also used by the portable systems. So goal of these tests is to verify the software we need to execute correctly and target the processor which can be software if compiled on the host should work same way and has been tested on the target systems this is where it s very important n this environment identified and used.

(Refer Slide Time: 54:46)

Integration tests environment

Table 13.1
Simulation level for
SW/U and SW/I tests

	Embedded software	Processor	Rest of embedded system	Plant
SW/U, SW/I (1)	Experimental (host)	Host	Simulated	Simulated
SW/U, SW/I (2)	Real (target)	Emulator	Simulated	Simulated

So there is a cable here based on the hook simulation level for the software unit and software integration basically this basically identifying based on the test but actual basically half of this doing the tests so we have embedded software and what sort of a processor and what can be tested on their simulation and this one based on the simulation process, the software we need the software integration.

So first it will be tested on the host as an experimental basis and the processor we have not going to use the target processor we are going to use the host processor such as six because most of the personal computers will have Exide this six structure but the tools will support what are the comparability on the target system? So rest of the embedded systems is simulated apart from the experimental course.

The second type of testing unit or integration is target than actual environment that will be target environment there we use the emulators and rest of them can simulate so this is how the test environment can be brought so but both the situations the test must or should give a simulates for

the input of the objects and provide a features to modify the test object and to record signals basically we validate signals inbounded we should support this environment and basically this can be done with the help of break point storing, reading memory and calculating variables all debug steps and all this can be time machines performance analyzes all this can be used there should be supported in the test environment.

(Refer Slide Time: 57:17)

Integration tests environment

- In the hardware/software integration (HW/SW/I) test the test object is a hardware part on which the integrated software is loaded. The software is incorporated in the hardware in memory, usually (E)EPROM.
- The piece of hardware can be an experimental configuration, for instance a hard-wired circuit board containing several components including the memory.
- The goal of the HW/SW/I test is to verify the correct execution of the embedded software on the target processor in co-operation with surrounding hardware. Because the behavior of the hardware is an essential part of this test, it is often referred to as "hardware-in-the-loop".

 Ref: test execution

so continuation of integration test environment in the hardware software integration test the test object is a hardware part on which the integrated software is loaded the software the software is incorporated in the hardware in memory basically a flash, the piece of hardware can be an experimental configuration percents hard wired circuit board containing several components including the memory.

So hardware is a focus here hardware software integration to goal of the hardware software integration is test to verify the correct execution of the embedded software on the target processor in cooperation with surrounding hardware so that is what will be elaboration of working of the hardware software integration because the behavior if an hardware is an essential part of this test it is often referred as hardware in the loop.

So those are different types of in loop situations software in loop simulation in loops hardware in loops basically these are integration strategies that are to be introduced used so that is there to identify the test environment for those hardware where we need to have an integration continuation of integration test environment the test environment for the hardware software integration is interface with the hardware depending on the test object and this degree of completeness and the following possibilities in this

(Refer Slide Time: 58:53)

Integration tests environment

- The test environment for the HW/SW/I test will have to interface with the hardware. Depending on the test object and its degree of completeness, the following possibilities exist:
 - offering input stimuli with signal generators;
 - output monitoring with oscilloscopes or a logic analyzer, combined with
 - data storage devices;
 - in-circuit test equipment to monitor system behavior on points other than
 - the outputs;
 - simulation of the environment of the test object (the "plant") in a real-time
 - simulator.

Basically it offers inputs stabiles it signal generators output monitoring with oscilloscopes or logic analyzer combined with data storage devisers in circuit equipment to monitor system behavior on points other than the outputs so these are some of the environment dependant tools that can be used then simulation of the environment test object in a real time simulator so what we do is basically we have a test hooks or interfaces from the target systems with the help of host book's this tools will be plugged and logic analyzers oscilloscopes or signal generators that can act as an input or output to conduct the test that is what the meaning of test environment here (Refer Slide Time: 59:50)

Integration tests environment

- Table 13.2 provides an overview of the level of simulation in the HW/SW/I test. The columns refer to the simulation areas in the generic scheme of an embedded system.

	Embedded software	Processor	Rest of embedded system	Plant
SW/U, SW/I (1)	Experimental (host)	Host	Simulated	Simulated
SW/U, SW/I (2)	Real (target)	Emulator	Simulated	Simulated
HW/SW/I	Real (target)	Real (target)	Experimental	Simulated

Table 13.2
Simulation level for
HW/SW/I tests

Integration test environment so in integration test environment basically we identify all the stimulated inputs and in focus on the captured output and tools that are used that are going to be used for capturing output so that we end to end the part is being verified so that is the focus of integration test environment so we will continue the test integration and integration environment in the next session and we will try to elaborate more on the integration environment

and generating test cases, example of test cases and how we can generate and we will also try to understand the integration test etc in the next class.