Software Testing

Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

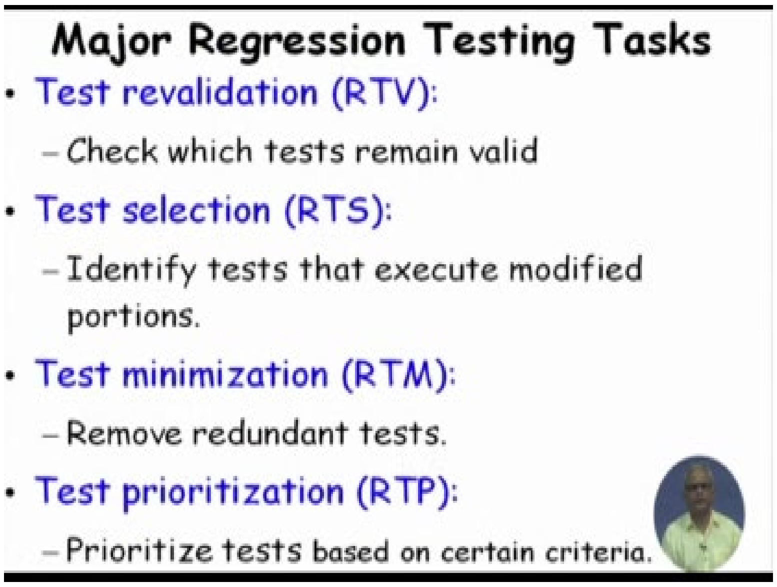
Lecture – 19
Testing Object-Oriented Programs

Welcome to this session. In the last session, we were discussing about regression testing and we saw that regression testing is an important type of testing, where we need to test the unchanged part of the software after each change.

Typically the change occurs to only few lines of code and few tens of thousands of line or hundreds of thousands of line of code does not change, but then we have to retest all those parts unchanged parts even though earlier they had passed. The reason is the dependency, a change may induce problems in the unchanged part because of the variable value propagation, some value which is changed in the during change of the some few lines of code propagate through the software and surface edge bugs in the unchanged part, and then we had seen that the set up test cases which are originally used to test the software they can be partitioned into the obsolete or invalid test cases, which we need to discard.

Then we have the redundant test cases which we need not as well run because they have zero chance of detecting a bug because they test the features which are truly independent of the change. And then we have the regression test cases which test those parts of the software which have some dependency with respect to the changes software.

(Refer Slide Time: 02:45)


Major Regression Testing Tasks

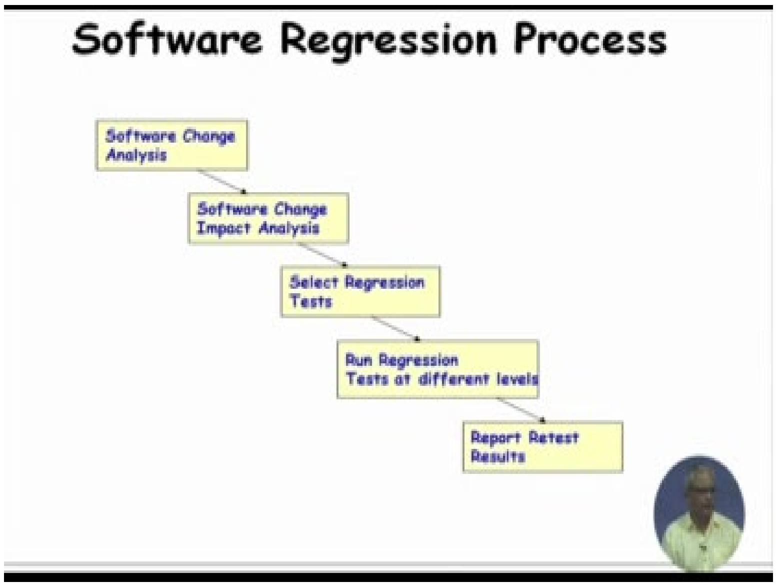
- Test revalidation (RTV):
  - Check which tests remain valid
- Test selection (RTS):
  - Identify tests that execute modified portions.
- Test minimization (RTM):
  - Remove redundant tests.
- Test prioritization (RTP):
  - Prioritize tests based on certain criteria.

Now, let us see, what are the different steps through which we select the regression test cases, so the major regression testing tasks are the steps in selecting the regression tests; the first thing is to identify the valid test cases because as we said that after every change some test cases become invalid, they need to be discarded. So, we have to first identify the valid test cases and out of the valid test cases, we need to identify and ignore the redundant test cases and that leaves us with the regression test cases and that is called as the regression test selection problem.

If the regression test cases are too many thousands then we might need to do minimization, where we remove test cases which you do not really they are kind of duplicates and they do not really detect additional bugs. So, the minimized set of test cases is as good as running the entire regression tests and we might also do regression test prioritization out of the regression test cases that have been selected.

We might prioritize them into 1, 2, 3, 4 with the highest priority given to those regression tests which have the maximum probability of detecting errors. Why do we need to prioritize? We need to prioritize because if you can detect the regression bugs earlier during testing then the development team can get started and that will reduce our delivery time. We might have hundreds of regression test cases and it may take a week to execute all of them and if we can detect that bugs in the very first day, we might get the development team started on the testing.

(Refer Slide Time: 05:06)

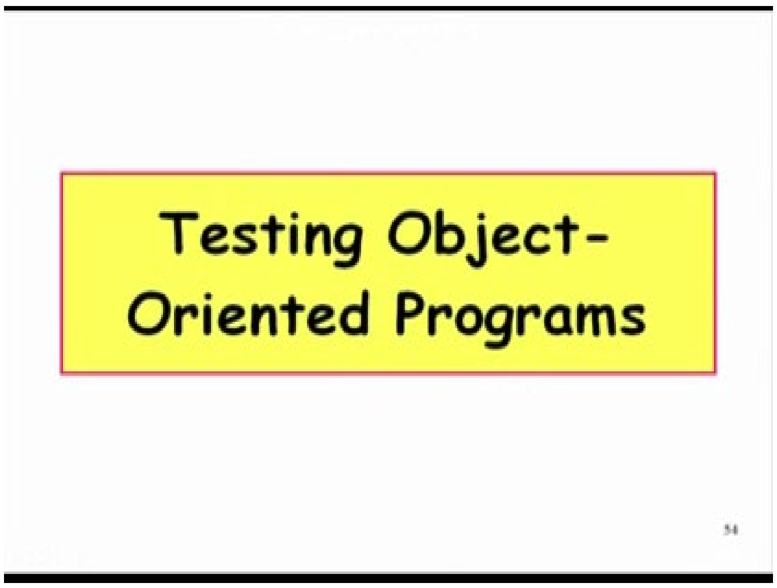


Software Regression Process

Software Change Analysis
Software Change Impact Analysis
Select Regression Tests
Run Regression Tests at different levels
Report Retest Results

So, that is the idea behind regression test prioritization. If we represent that in the form of a schematic diagram, we first do the change analysis and based on the change analysis we find out which part of the code is impacted, which part of the unchanged code is impacted and which part is independent and based on which part is the impacted code and the test cases which run or execute these impacted code, we make the regression test selection and then we run the regression tests and report and retest the results.

(Refer Slide Time: 05:54)



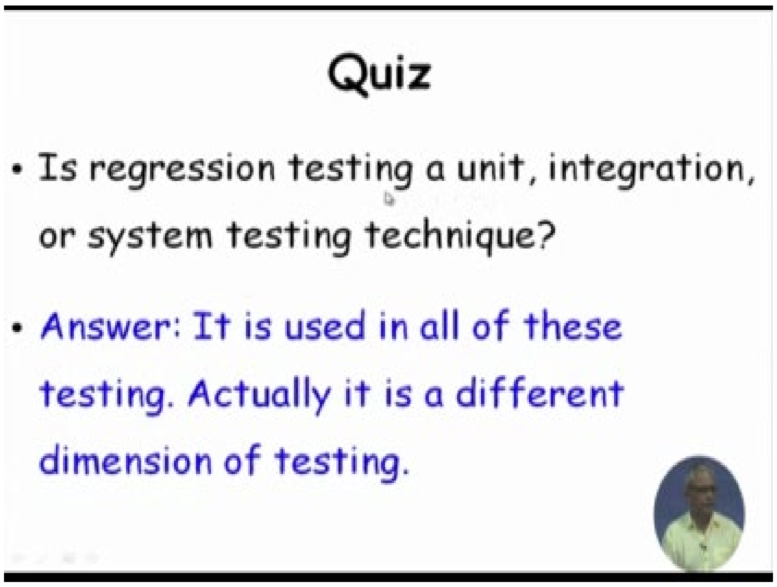
Testing Object-Oriented Programs

54

Now, let us look at another important topic in testing which is testing object oriented programs. We had said earlier that object oriented programs were introduced with the assumption that if we have very good programming practices inculcated then the chances of bugs will be less, for example, inheritance encapsulation and so on.

The different features of object oriented programming, they are known to be very sound programming concepts and they potentially reduce the chances of bug reuse adjusting could reduce development effort and at the same time they are also expected to reduce the chances of bug, but we will be surprised to learn now that actually complicate the testing. They introduced new types of bug which you were not aware in procedural code and therefore, we need different testing for object oriented code. We cannot just use the same techniques we used for procedural programming in object oriented code, we need different testing strategies.

(Refer Slide Time: 07:26)

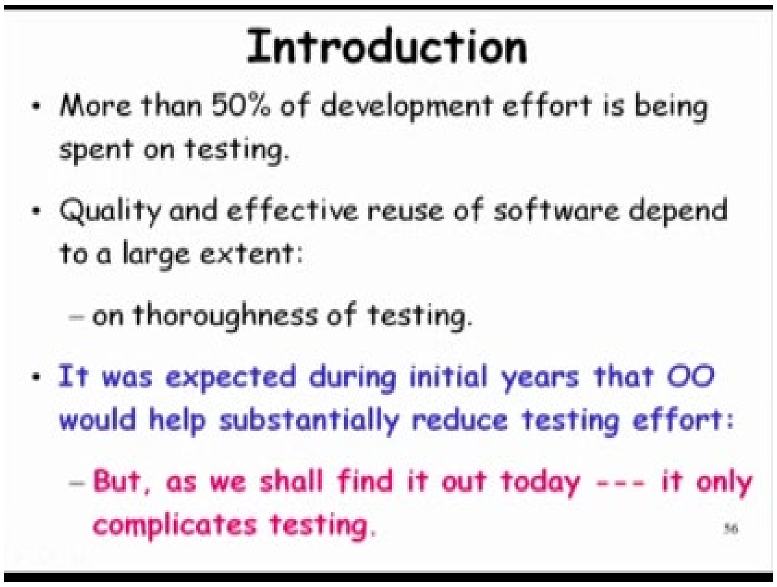


Quiz

- Is regression testing a unit, integration, or system testing technique?
- Answer: It is used in all of these testing. Actually it is a different dimension of testing.

Let us look at that, but before that let us have small a quiz on regression testing which we just discussed. So, the question is that is regression testing applicable at unit, integration or system testing is regression testing applicable to unit testing, integration testing or system testing, the answer is that actually it is applicable to all this three levels of testing unit testing integration testing and system testing actually regression testing is a different dimension of testing and therefore, we cannot say that regression belongs to only one level of testing. So, level is one dimension and regression testing is a different dimension of testing and it is applicable to all levels of testing.

(Refer Slide Time: 08:24)



Introduction

- More than 50% of development effort is being spent on testing.
- Quality and effective reuse of software depend to a large extent:
  - on thoroughness of testing.
- It was expected during initial years that OO would help substantially reduce testing effort:
  - But, as we shall find it out today --- it only complicates testing.

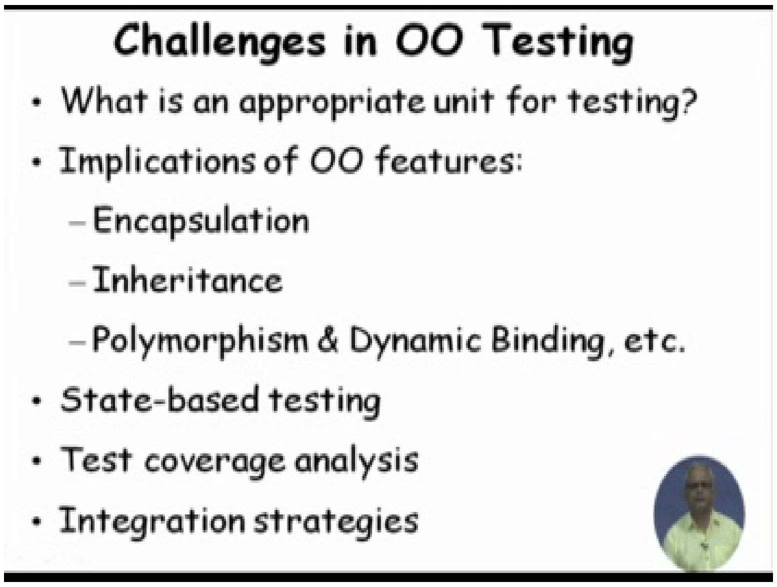
56

Now, let us look at testing, how do we go about testing object oriented software? If you look at the data that is made; research data that is made available a typical organization development organization spends 50 percent of its effort on testing, the rest 50 percent on specification design, coding, etcetera and the main reason behind spending such huge effort on testing is that the quality of the software depends to a large extent on the thoroughness of testing and now the customers are very quality conscious. So, if an organization releases software full of bugs and not only the customers will reject that product, but also they will not buy products from that organization.

So, every organization spends significant effort on testing and in that context, the object orientation was a welcome change when object orientation was proposed. In the initial years everybody expected that it will substantially reduce the testing effort, the 50 percent effort that is being spent may be reduce to 20-30 percent that was the expectation, but then as we self find out now that object orientation actually complicates testing and may need more effort rather than reducing the effort.

Let us look at what are the challenges of testing object oriented programs which did not exist in procedural programs. So, here we have new problems, new types of faults and therefore, very difficult challenges in testing object oriented programs which did not exist in procedural program testing. The first challenge is that what is the unit of testing? If you remember the context of procedural programs, we had said that a unit is either a function or a module.

(Refer Slide Time: 11:02)



Challenges in OO Testing

- What is an appropriate unit for testing?
- Implications of OO features:
  - Encapsulation
  - Inheritance
  - Polymorphism & Dynamic Binding, etc.
- State-based testing
- Test coverage analysis
- Integration strategies

So, the idea is that in procedural programs we test the units and as long as the units have been thoroughly tested, we then concentrate on integration testing removing the interface errors through the integration testing and then we do the system testing, but if you look at the analogy of a procedure and a object program you might think that, if unit is a function in procedural program a unit of testing in object oriented programs may be a method because methods are analogous to functions, but as we will find out, no, it is not correct.

The other things is that the good features of object orientation, encapsulation, inheritance, polymorphism, dynamic binding will investigate, do they help in testing, do they reduce the effort in testing or do they make it more complicated and need to spend more effort in testing.

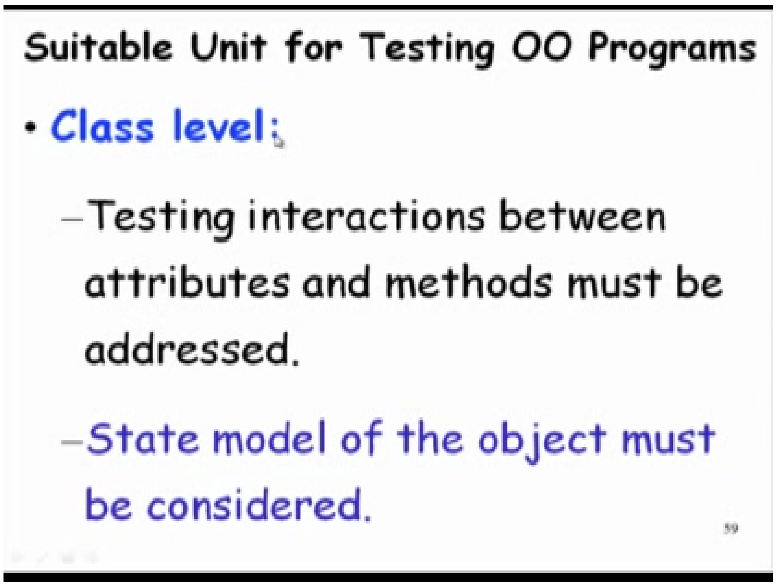
Let us investigate that issue and then let us examine the new type of testing here needed state based testing and then let us look at what is the notion of test coverage analysis in the context of object oriented programming and also the integration strategies that we had discussed in context of procedural programming top down bottom up and so on, is not really appropriate here because these are not really hierarchical designs clear designs these are only the objects interacting with each other. So, we need different integration strategies.

(Refer Slide Time: 13:09)

First let us address this is a very fundamental question what is a suitable unit of testing for object oriented programs because this is the foundation of any testing that we need to do first the unit level testing. So, which units or which elements of the software we need to test independently and then do the integration testing in the conventional procedural programs the unit is a function, but as we will see just now that method is not really the basic unit of testing for object oriented programs.

The analogy, if we extend function to method to be unit of testing in object oriented program is not really correct, we will see what the basis of making this claim. Actually, the basis of making this claim is attributed to the Weyukar's Anticomposition axiom, which says any amount of testing of individual methods cannot ensure that a class has been satisfactorily tested.

(Refer Slide Time: 14:48)



Suitable Unit for Testing OO Programs

- Class level:
  - Testing interactions between attributes and methods must be addressed.
  - State model of the object must be considered.

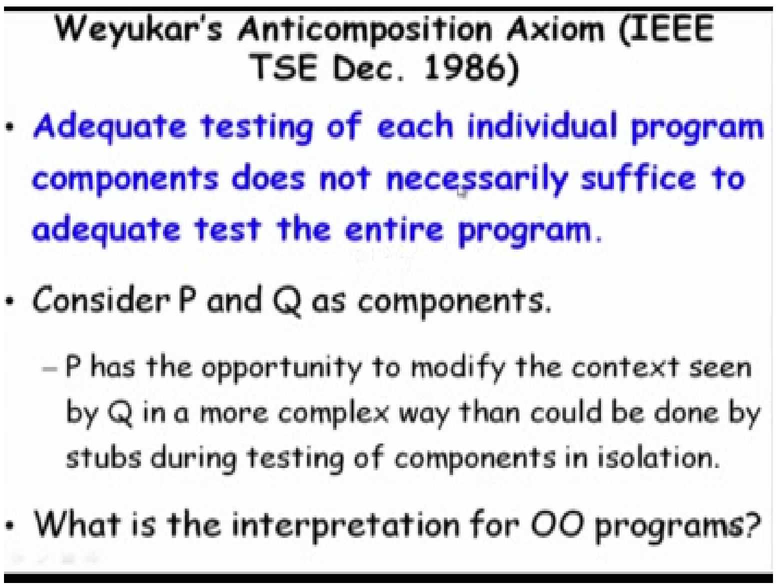
59

Let see what it really means. So, that is Weyukar, actually the suitable unit for object oriented programs is class, we need to test all classes in isolation and then finally, integrate those classes. So, why is class a good unit of testing and testing the methods and then integrating will not work? So the answer to that is that, if we just simply test the methods then we are not doing several things correctly. One thing is that we have this class attributes which are shared between various methods.

We are not testing those just by testing the methods and also the classes typically have state models, since they store data every class stores data the classes.

A significant state model that means that a class or an object can exist in a number of states and if we test a method in one of the state it may be working correctly in one state, but it may not be working when the class is in some other state. So, we need to consider the state model of the class and how the objects assume different states and then we need to test the methods there. So, not only that we have to test the method interactions through class variables but also we need to test the states the methods at different states of the class.

(Refer Slide Time: 17:04)



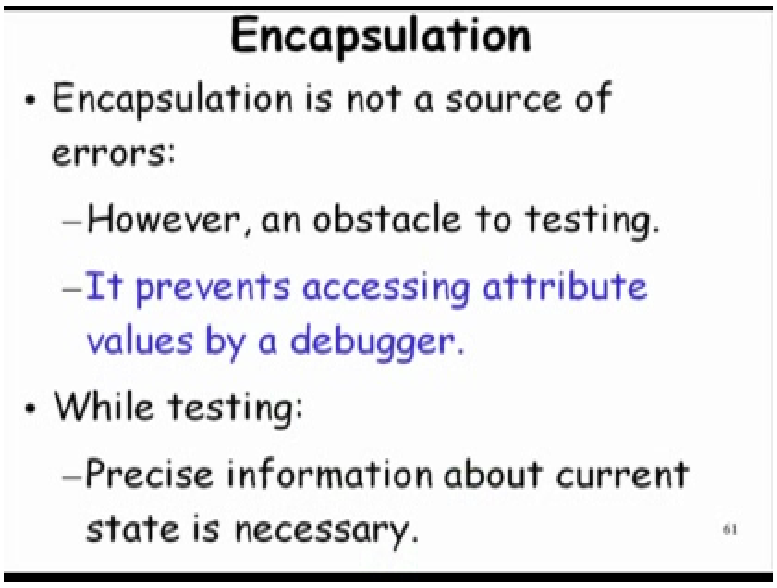
Weyukar's Anticomposition Axiom (IEEE TSE Dec. 1986)

- Adequate testing of each individual program components does not necessarily suffice to adequate test the entire program.
- Consider P and Q as components.
  - P has the opportunity to modify the context seen by Q in a more complex way than could be done by stubs during testing of components in isolation.
- What is the interpretation for OO programs?

The Weyukar's Anticomposition axiom which is a land mark result in this area appeared in IEEE transactions on software engineering in way back in 1986. The statement of his axiom is that adequate testing of each individual program components does not necessarily suffice adequate test of entire program. So, he says that if you just test the individual component that does not really guarantee that the entire program will be working. If p and q are components, p has an opportunity to modify the contexts in by q in a more complex way then that could be done by stubs during testing of components in isolation and therefore, as long as p and q can interact we need to test them together.

So, the interpretation of this axiom for object oriented programs is that we cannot consider methods as even it is just testing methods and then integrating them may not be correct, we need to test the classes it is the unit level and integrate the classes.

(Refer Slide Time: 18:28)



Encapsulation

- Encapsulation is not a source of errors:
  - However, an obstacle to testing.
  - It prevents accessing attribute values by a debugger.
- While testing:
  - Precise information about current state is necessary.

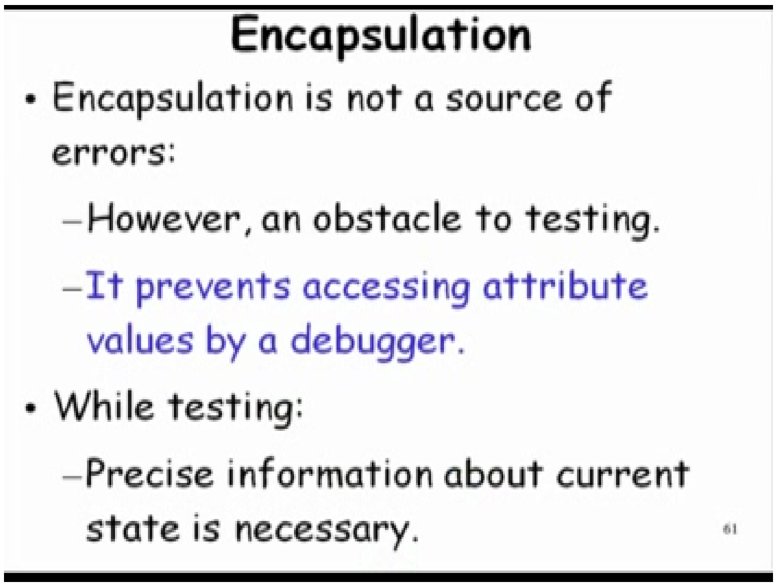
61

So, that is the first important result that class is the basic unit of testing for objects oriented programs. Now, let us look at some of the object oriented features important features and let us see whether they can help in testing or they create problems, hinder testing. First let us look at encapsulation.

Encapsulation as we know is having the data in the methods together. Actually encapsulation is not a source of errors, but then it is an obstacle to testing, why is it because due to encapsulation, we cannot directly access the private variables of a class and therefore, a debugger it would become difficult to find the values of private variables of classes because there are no methods which will report and for the accessing this private variables you need to access it through methods, you cannot directly access it and therefore, it is becomes complicated.

So, how does one overcome this problem several solutions are possible, one is state reporting methods.

(Refer Slide Time: 19:59)



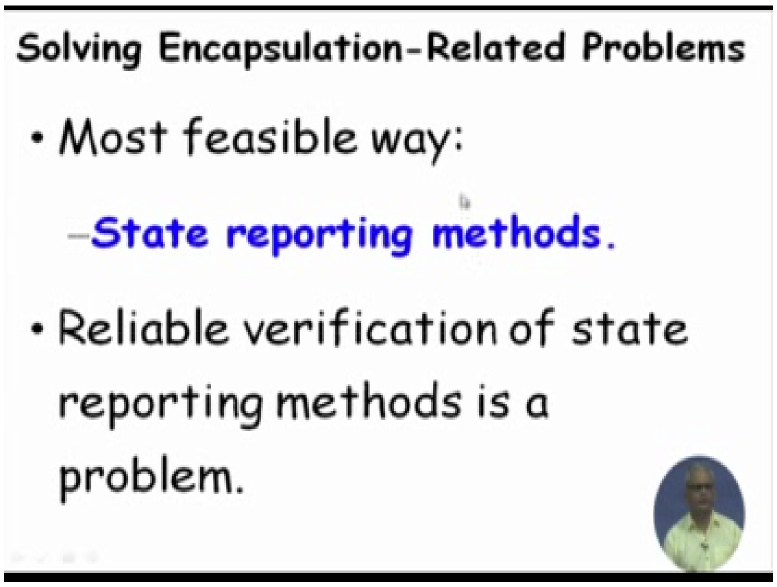
Encapsulation

- Encapsulation is not a source of errors:
  - However, an obstacle to testing.
  - It prevents accessing attribute values by a debugger.
- While testing:
  - Precise information about current state is necessary.

61

So, we might have additional methods in a class which report the values of these private variables low level probes to manually inspect object attributes or formal proof of correctness techniques do not really look at the variables, but proof of correctness techniques.

(Refer Slide Time: 20:36)

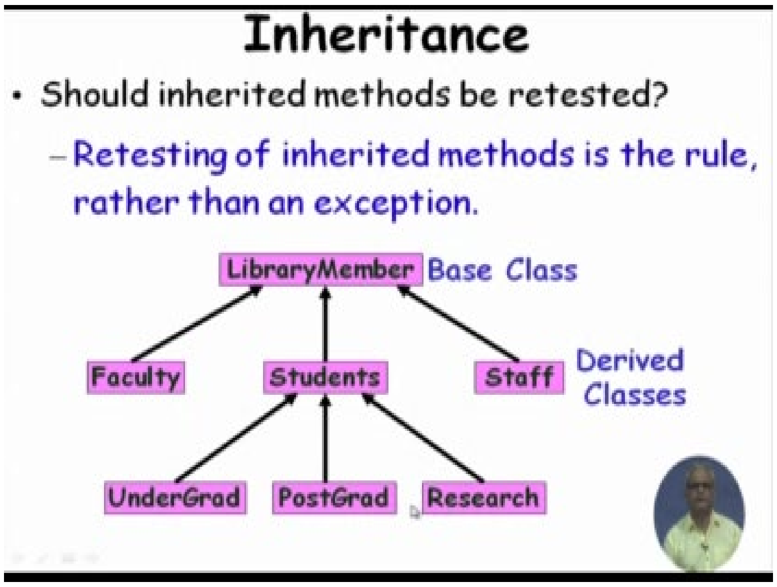


Solving Encapsulation-Related Problems

- Most feasible way:
  - **State reporting methods.**
- Reliable verification of state reporting methods is a problem.

But the most promising way is the state reporting methods and normally this is the way encapsulation is handled by the testers, they have state reporting methods and they overcome the encapsulation problem and can check the values of the private variables.

(Refer Slide Time: 20:47)



Inheritance

- Should inherited methods be retested?
  - Retesting of inherited methods is the rule, rather than an exception.

$$\begin{tikzcd}
 & \text{LibraryMember} & \\
\text{Faculty} \arrow[ur] & \text{Students} \arrow[u] & \text{Staff} \arrow[ul] \\
\text{UnderGrad} \arrow[ur] & \text{PostGrad} \arrow[u] & \text{Research} \arrow[ul]
\end{tikzcd}$$

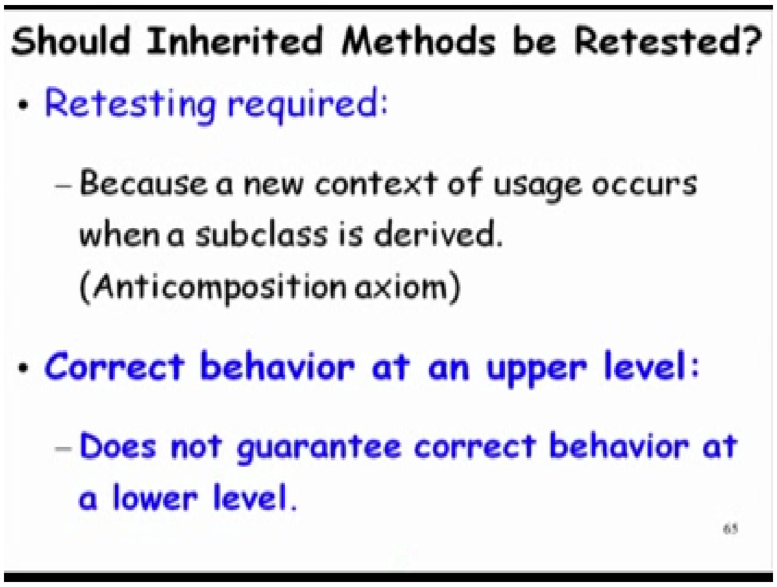
Base Class
Derived Classes

Now, let us look at another important feature of object oriented programs which is inheritance. Inheritance, we know that one of the important benefit of inheritance is that once a class has been tested is working well we can add new features in its sub class and therefore, we reuse working code do not have to write all that code it has been tested and written. We just add few methods of class variables, but then the question here is that, once we have a derived class and it inherits many of the methods from the ancestor classes should the inherited methods be tested in the derived class.

Surprisingly and unfortunately, it is mandatory to test the inheritance inherited methods retesting of the inherited methods is the rule rather than an exception. Why is that because these were the methods that were all tested in the base class to be working fine and we have just inherited them in the derived class, why do we have to test them again thoroughly in the derived class? Let us examine that problem.

So, this is a base class library member and we have derived classes and several hierarchies of derivation and the methods, attributes here become available in students and then the students are further attributes and methods. So, all that is inherited by these children classes and so on. Actually retesting is necessary because of the new context of use in the sub class. So, that is basically the anti composition axiom,again.

(Refer Slide Time: 22:59)



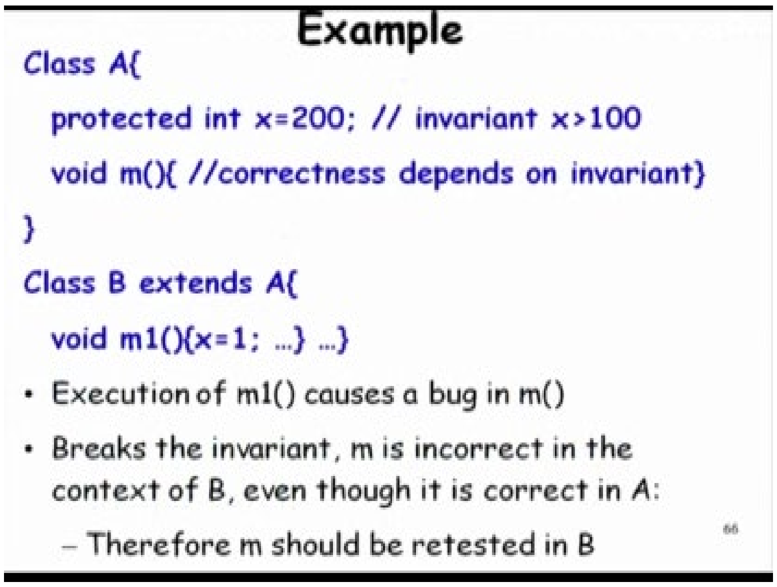
Should Inherited Methods be Retested?

- Retesting required:
  - Because a new context of usage occurs when a subclass is derived. (Anticomposition axiom)
- Correct behavior at an upper level:
  - Does not guarantee correct behavior at a lower level.

65

We will see this with some examples what we mean by new context of usage, but then we can say that, if you observed that a method has behaved correctly at an upper level does not guarantee that it will behave at a lower level and.

(Refer Slide Time: 23:30)



Example

```
Class A{
  protected int x=200; // invariant x>100
  void m(){ //correctness depends on invariant}
}
Class B extends A{
  void m1(){x=1; ...} ...}
```

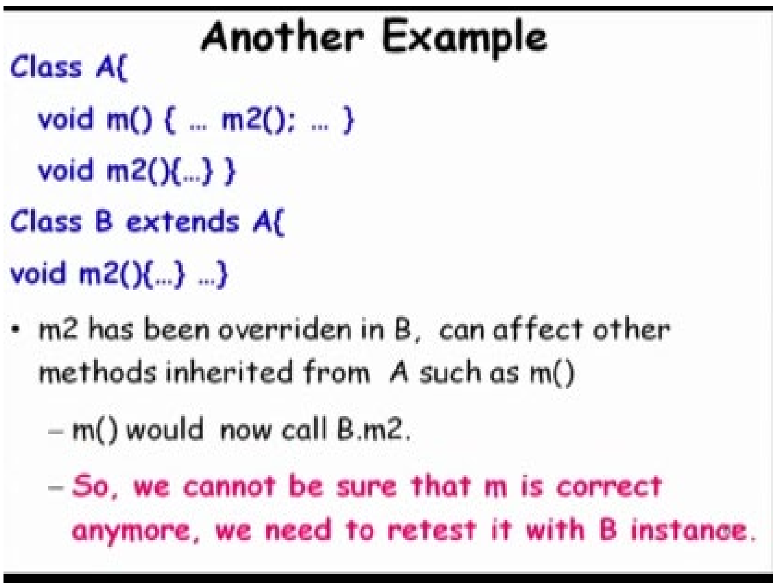
- Execution of m1() causes a bug in m()
- Breaks the invariant, m is incorrect in the context of B, even though it is correct in A:
  - Therefore m should be retested in B

66

Let us look at some examples. Let us assume that this was the code here class A had this variable x having value 200 and the invariant or the requirement on this variable is that x should always be greater than 100 and then we had several methods here, which we are using this variable and they were all maintaining this invariant that is the way the class was designed and they were working correctly, but then some programmer they extended the class A and he did not really look into the code just extended it and here he introduced the additional method that he needed m 1, where he set the variable x to 1.

Now, once we had extended A into B, the methods which are inherited here, m, etcetera which are working earlier will fail to work because the ( ) invariant has been violated here. So, this is one simple example. Let us look at another example, a class A had a method m which was in it is body calling another method m 2 and m 2 was also defined in class A. So, both m and m 2 are defined in class A and m 2 is called by m. Now, class B extends A, but then it has overridden method B, now when m is called in class context of class B.

(Refer Slide Time: 25:30)



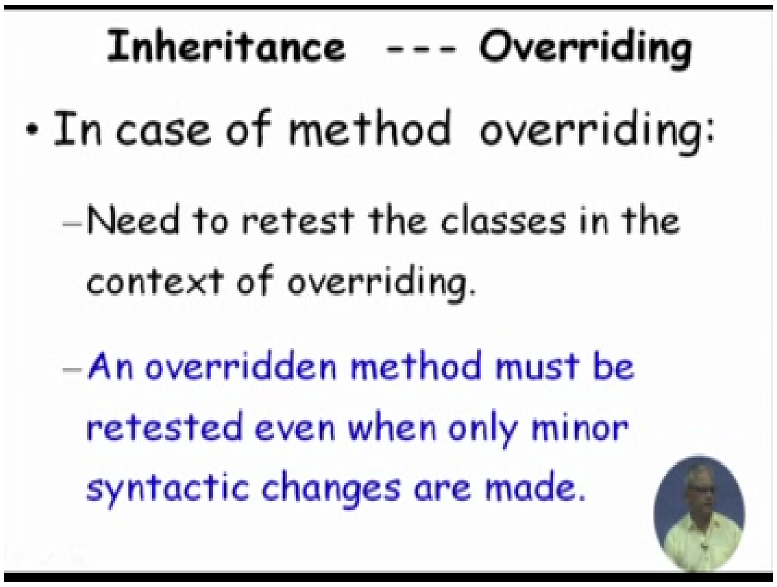
Another Example

```
Class A{
  void m() { … m2(); … }
  void m2(){…} }
Class B extends A{
void m2(){…} …}
```

- m2 has been overriden in B, can affect other methods inherited from A such as m()
  - m() would now call B.m2.
  - So, we cannot be sure that m is correct anymore, we need to retest it with B instance.

Then m is inherited here and we call m, m was working all right in class A. It was tested now for class B object when we call m, it is class called in the context of class B and therefore, due to polymorphism the call m will now, call B dot m 2 rather than calling A dot m 2 and therefore, it was working well with m 2, but now on the same m is calling B dot m 2 and therefore, it may fail and of course, when we have method overriding we have written a new method overriding a base class method obviously, we need to test. So, the inherited methods we need to test and also the overridden methods we need to test.

(Refer Slide Time: 26:22)



Inheritance --- Overriding

- In case of method overriding:
  - Need to retest the classes in the context of overriding.
  - An overridden method must be retested even when only minor syntactic changes are made.

So, the original hope that the methods have been tested at base class will not have to be tested in the derived class is not true. We have to test actually all those method which have been inherited from base class.

We are just running out of time in this session, we will continue discussing about how to test an object oriented program in the next session.

Thank you.