



Software Testing Methodologies

BITS Pilani

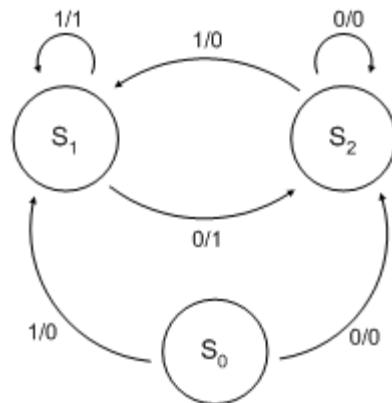
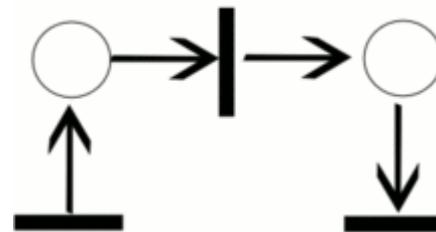
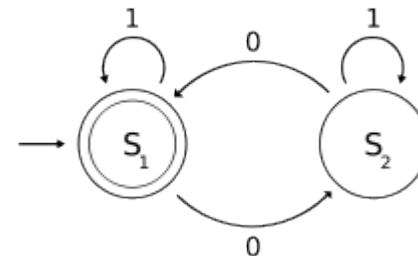
Prashant Joshi

Model Based Testing

- Process of creating a **Model** results in deeper insights and understanding of the system
- Adequacy of MBT – depend on accuracy of the Model
- Sequence of steps
 - Model the system
 - Identify the threads of system behavior in the model
 - Transform threads into test cases
 - Execute the test cases (on actual system) and record the results
 - Revise the model(s) as needed and repeat the process

Executable Models

- Finite State Machines
- Petri Nets
- StateCharts



Modelling

- **What the system is**
 - Emphasize structure
 - Components, their functionality and interfaces
 - DFD, Entity/Relation models, hierarchy charts, classes diagrams and class diagrams
- **What the system does**
 - Emphasize behavior
 - Decision Tables, FSM, StateCharts & Petri Nets

Refer: Page 225 and 226 of T1

Model Based Testing Tools

- Modelling the system provides ways to generate test cases automatically
- Example: <http://graphwalker.org/index>

Finite State Machines

- Method of expression of a design
- Simple way to model state-based behavior
- State Charts – a rich extension of FSM
- Petri nets – useful formalism to express concurrency and timing

State Based Modelling Techniques



- State transition diagrams
- Extended Finite State Machines

The Fault Model

- Process of Design
- Conforming of the implemented system to the Requirements
- Fault Model defines a set of small set of possible fault types that can occur
- Our focus here is FSM or EFSM (*Lets talk modelling later!*)

Fault Categories

- Operation Error
 - Error generated upon transition
 - Incorrect output function
- Transfer Error
 - Incorrect state transition
- Extra State Error
- Missing State Error

Some examples to discuss

- Garage Door
- Building Lighting Control System
- Lift/Elevator Control System (One or Multiple)
- A MMI Interface of an instrument

Simple Vending Machine



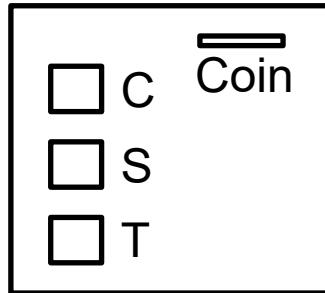
- Tea/Coffee vending Machine
- Options
 - Accepts token/coin
 - Sugar

Vending Machine

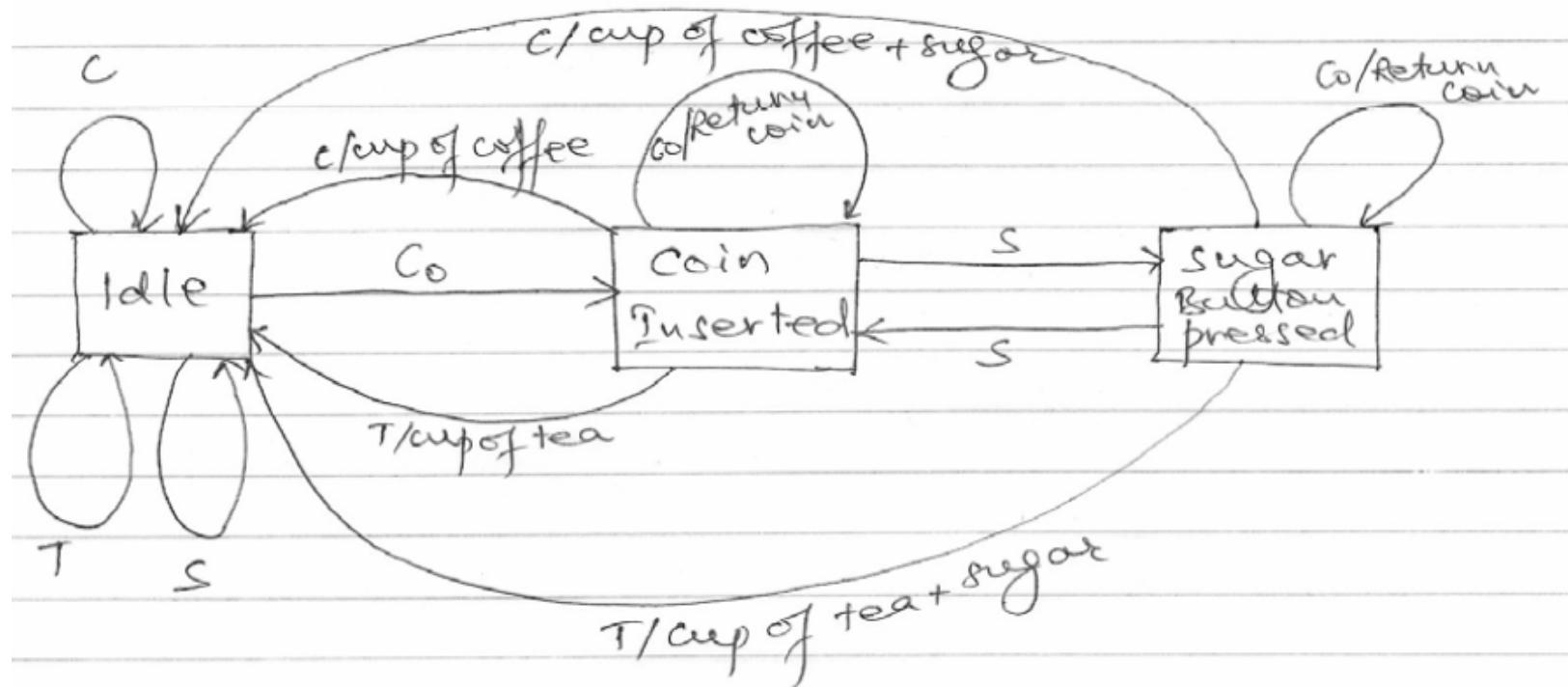
innovate

achieve

lead



C: Coffee Button pressed
T: Tea Button pressed
S: Sugar Button pressed
Co : Coin inserted





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 7: Agenda

Module 7: Model Based Testing (1/2)

Topic 7.1

Model Based Testing – Introduction & Overview

Topic 7.2

Finite State Machines & Fault Model

Topic 7.3

Examples

Topic 7.4

Case Study



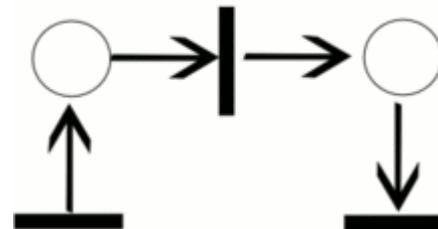
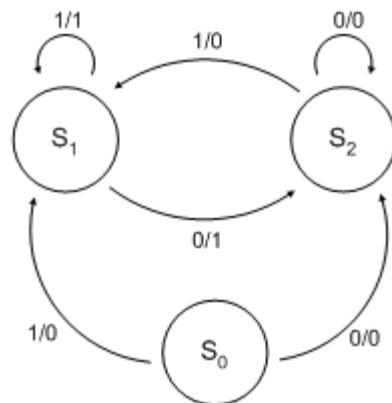
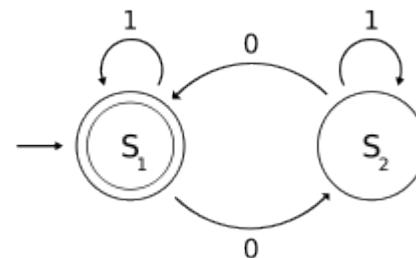
Topic 7.1: Model Based Testing – Introduction & Overview

Model Based Testing

- Process of creating a **Model** results in deeper insights and understanding of the system
- Adequacy of MBT – depends on accuracy of the Model
- Sequence of steps
 - Model the system
 - Identify the threads of system behavior in the model
 - Transform threads into test cases
 - Execute the test cases (on actual system) and record the results
 - Revise the model(s) as needed and repeat the process

Executable Models

- Finite State Machines
- Petri Nets
- StateCharts



What System Is?

- The Components
- Their Functionality
- Interfaces
- All that emphasizes structure

What System Does?

- Decision tables
- State Charts
- Petri nets/EDPN
- FMS/EFSM
- All these describe System Behaviour
- Look for expressive capabilities of the system

Modelling

- **What the system is**
 - Emphasize structure
 - Components, their functionality and interfaces
 - DFD, Entity/Relation models, hierarchy charts, classes diagrams and class diagrams
- **What the system does**
 - Emphasize behavior
 - Decision Tables, FSM, StateCharts & Petri Nets

Refer: Page 225 and 226 of T1

Model Based Testing Tools

- Modelling the system provides ways to generate test cases automatically
- Example: <http://graphwalker.org/index>



BITS Pilani

Software Testing Methodologies

Prashant Joshi



Topic 7.2: Finite State Machine & Fault Model

Finite State Machines

- Method of expression of a design
- Simple way to model state-based behavior
- State Charts – a rich extension of FSM
- Petri nets – useful formalism to express concurrency and timing

State Based Testing

- State “Behaviour” exhibited
- Example: Stack
- Operation pop

```
s.push(5);  
y=s.pop();  
print(y);
```

Y=5

```
s.push(5);  
s.push(7);  
y=s.pop();  
print(y);
```

Y=7

State Based Testing

- Testing state based components
 - State-full
 - State-less
 - Testing only individual methods or functions for state-full components is not sufficient
-

State-full Component

- A set of States
- Transition between states

State-full Component

- Objects/Classes
- Control Systems
- Embedded Systems
- Communication Systems
- ...

State Based Component

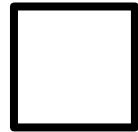
States	Values (of some data)
Empty State	top=0
Full State	top=10
Partial State	1<=top<=9

State Based Modeling Techniques



- State transition diagrams
- Extended Finite State Machines

State Transition Diagram



A State



A transition



The Fault Model

- Process of Design
- Conforming of the implemented system to the Requirements
- Fault Model defines a set of small set of possible fault types that can occur
- Our focus here is FSM or EFSM (*Lets talk modelling later!*)

Fault Categories

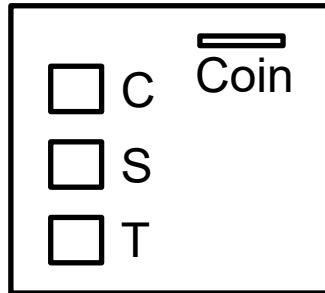
- Operation Error
 - Error generated upon transition
 - Incorrect output function
- Transfer Error
 - Incorrect state transition
- Extra State Error
- Missing State Error

Vending Machine

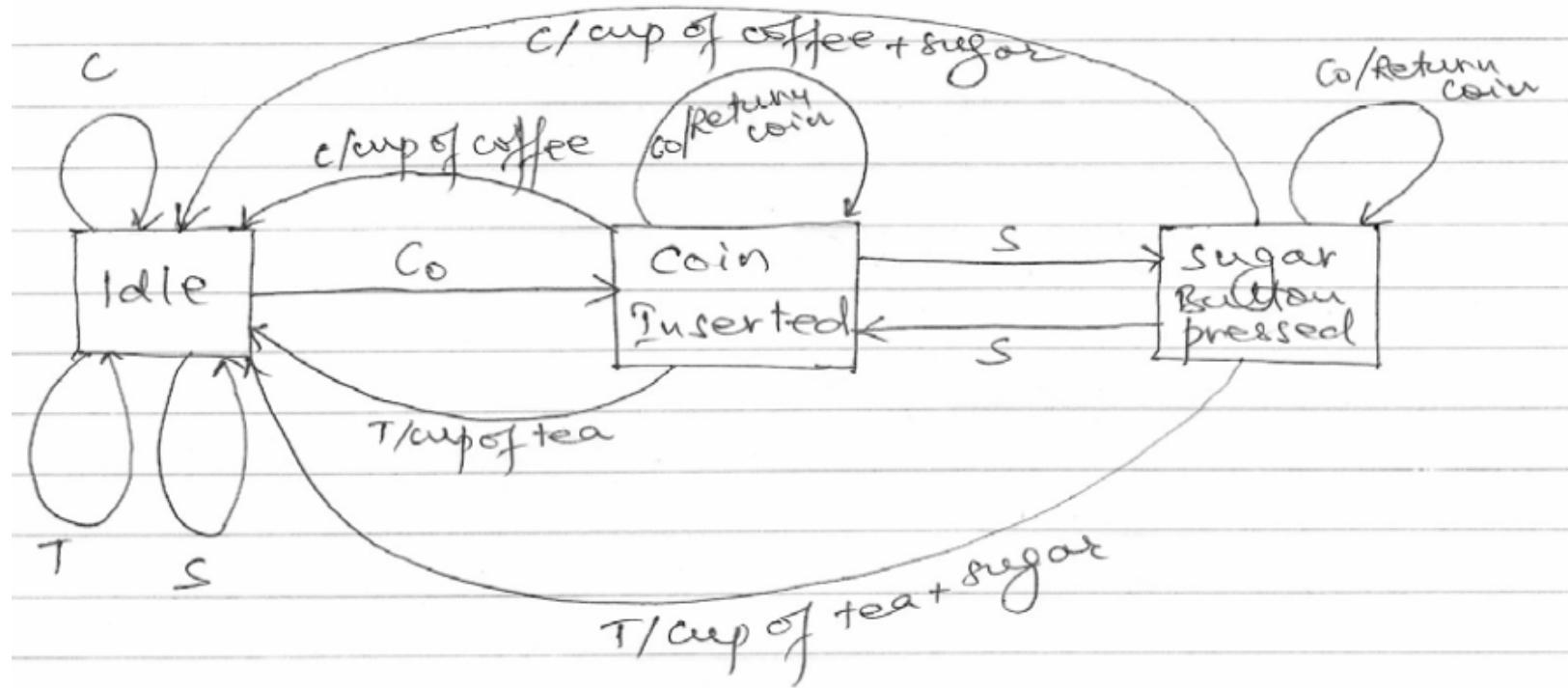
innovate

achieve

lead



C: Coffee Button pressed
T: Tea Button pressed
S: Sugar Button pressed
Co : Coin inserted



Some examples to discuss

- Garage Door
- Building Lighting Control System
- Lift/Elevator Control System (One or Multiple)
- A MMI Interface of an instrument



BITS Pilani

Software Testing Methodologies

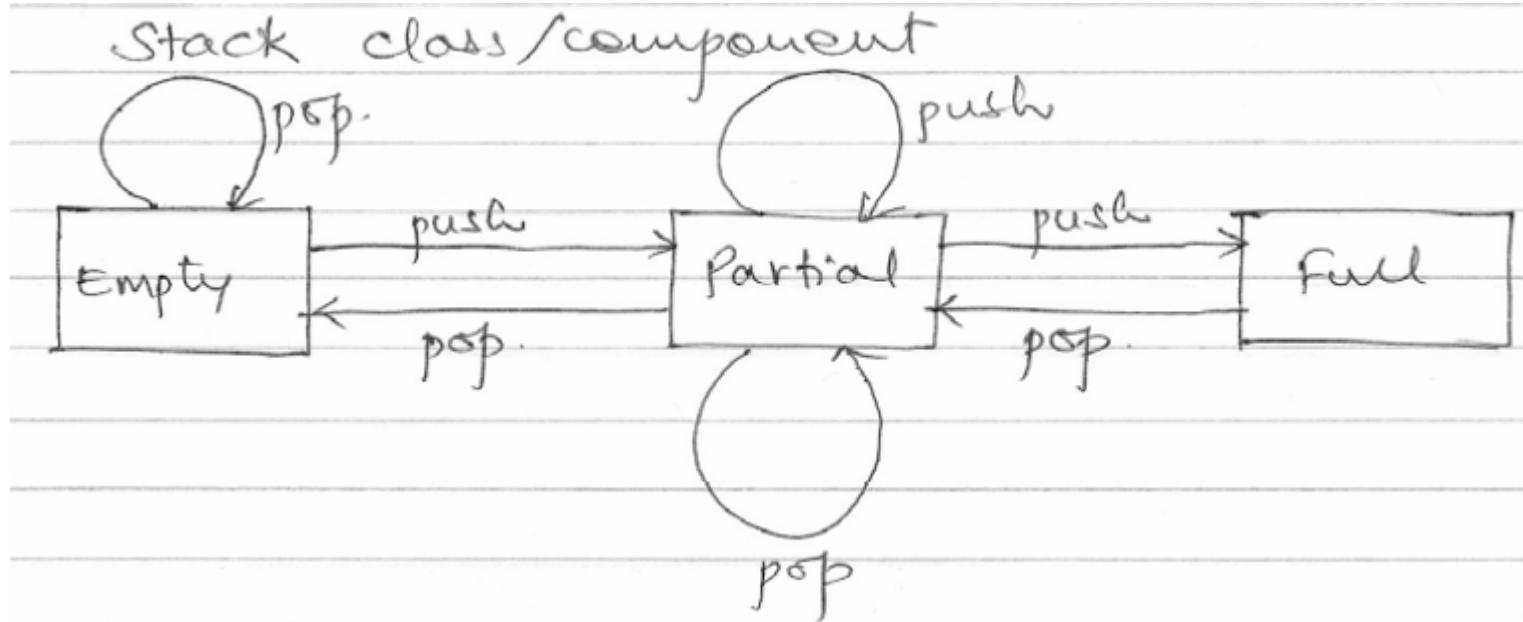
Prashant Joshi



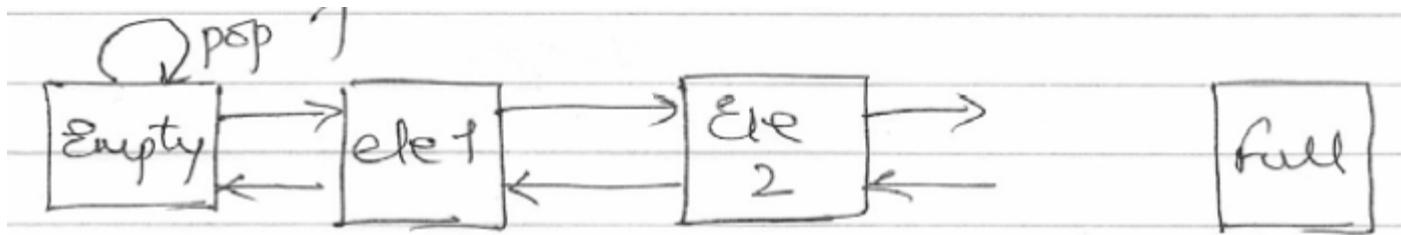
Topic 7.3: Examples

Stack

- Simple stack
- Operations (push and pop)



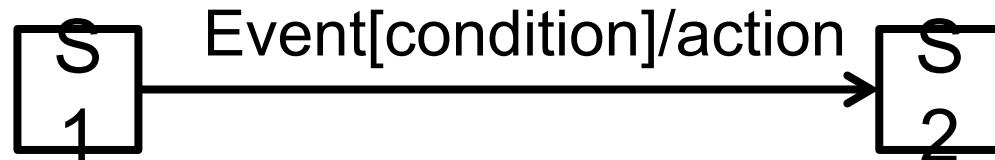
Notion of State Explosion



- Too many states
- State Explosion problem!

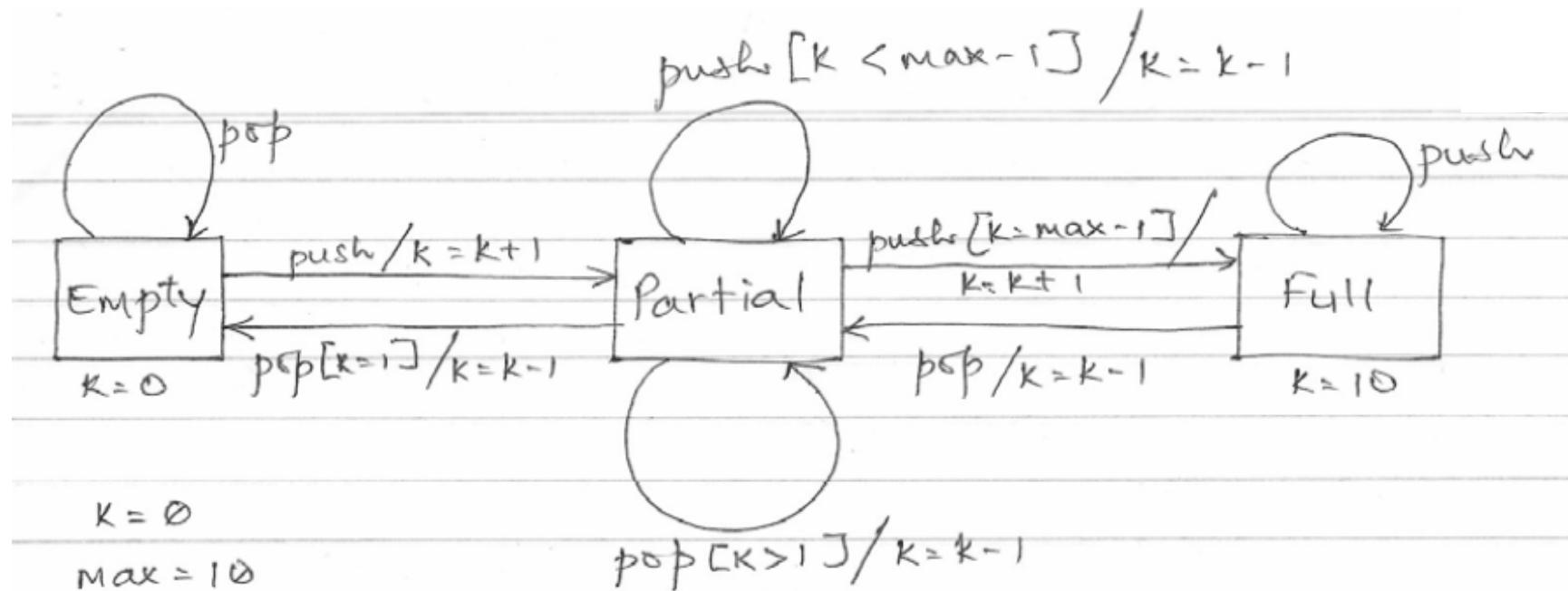
Extended FSM

- Extended Finite State Machine
- Extension of the state transition diagram by introducing
 - Variables
 - Conditions



1. The system is in S1
2. Event occurs
3. Condition evaluates to true
4. Transition from S1 to S2 takes place
5. Action is performed

Stack



Testing Stack Component

- Operations/methods
 - Push
 - Pop
- State based testing

Testing with Criteria

- State Testing
 - Every state in the model should be visited at least once
- Transition Testing
 - Every transition in the model is “traversed” at least once
- Path Testing
 - Traverse every path in the model at least once

State Coverage

Test #1: s.push(5) //partial state

Test #2: s.push(5)
s.push(7)
s.push(20) } 10 push operations
//full state

- State Coverage Satisfied

Transition Coverage

Test #3: y.pop()

Test #4: s.push(5)

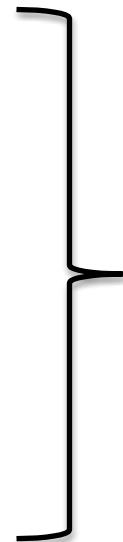
y.pop()

Test #5: s.push(5)

s.push(7)

s.push(20)

s.push(12)



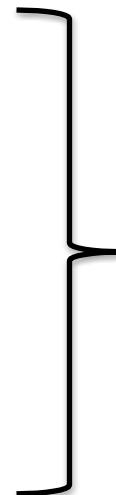
11th push

Transition Coverage

Test #6: `s.push(5)`
`s.push(7)`
`y=s.pop()`

Test #7: `s.push(5)`
`s.push(7)`

`s.push(17)`
`y=s.pop()`



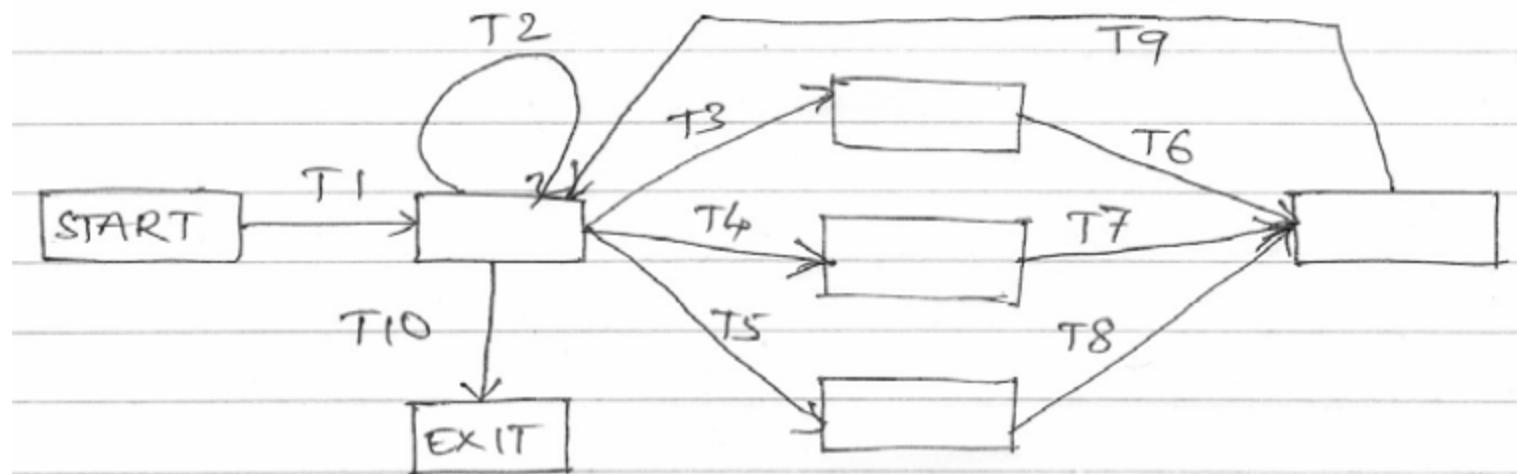
10 push

Constrained Path Testing

- Modified Path Testing
- Traverse every path in the model under the constraint that any transition in the path is traversed at most N times

Constrained Path Testing

Use of $n=1$ (Say repeat only once)



Constrained Path Testing

T1: T1, T10

T2: T1, T2, T10

T3: T1, T3, T6, T9, T10

T4: T1, T2, T3, T6, T9, T10

T5: T1, T3, T6, T9, T2, T10

T6, T1, T4, T7, T9, T10

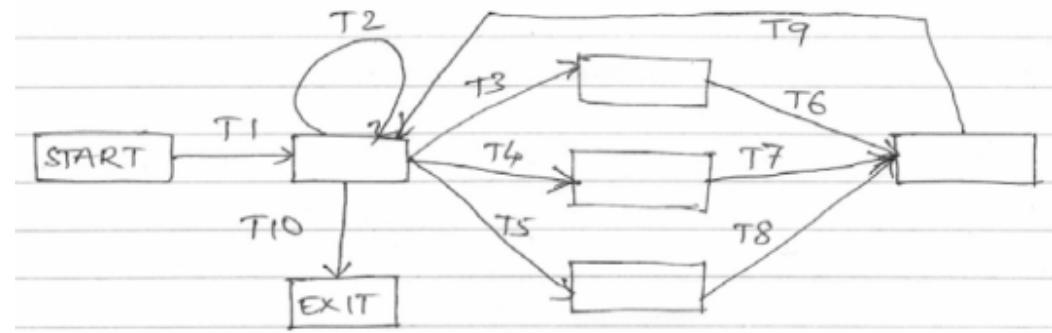
T7, T1, T2, T4, T7, T9, T10

T8: T1, T4, T7, T9, T2, T10

T9: T1, T5, T8, T9, T10

T10: T1, T2, T5, T8, T9, T10

T11: T1, T5, T8, T9, T2, T10



State Based Testing

- We use state model to design test cases using different strategies
 - State Testing
 - Transition Testing
 - Path/Constraint path testing
- Non-executable elements



Software Testing Methodologies

BITS Pilani

Prashant Joshi



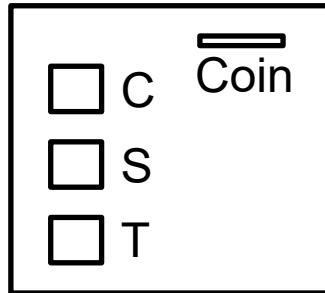
Topic 7.4: Case Study

Simple Vending Machine

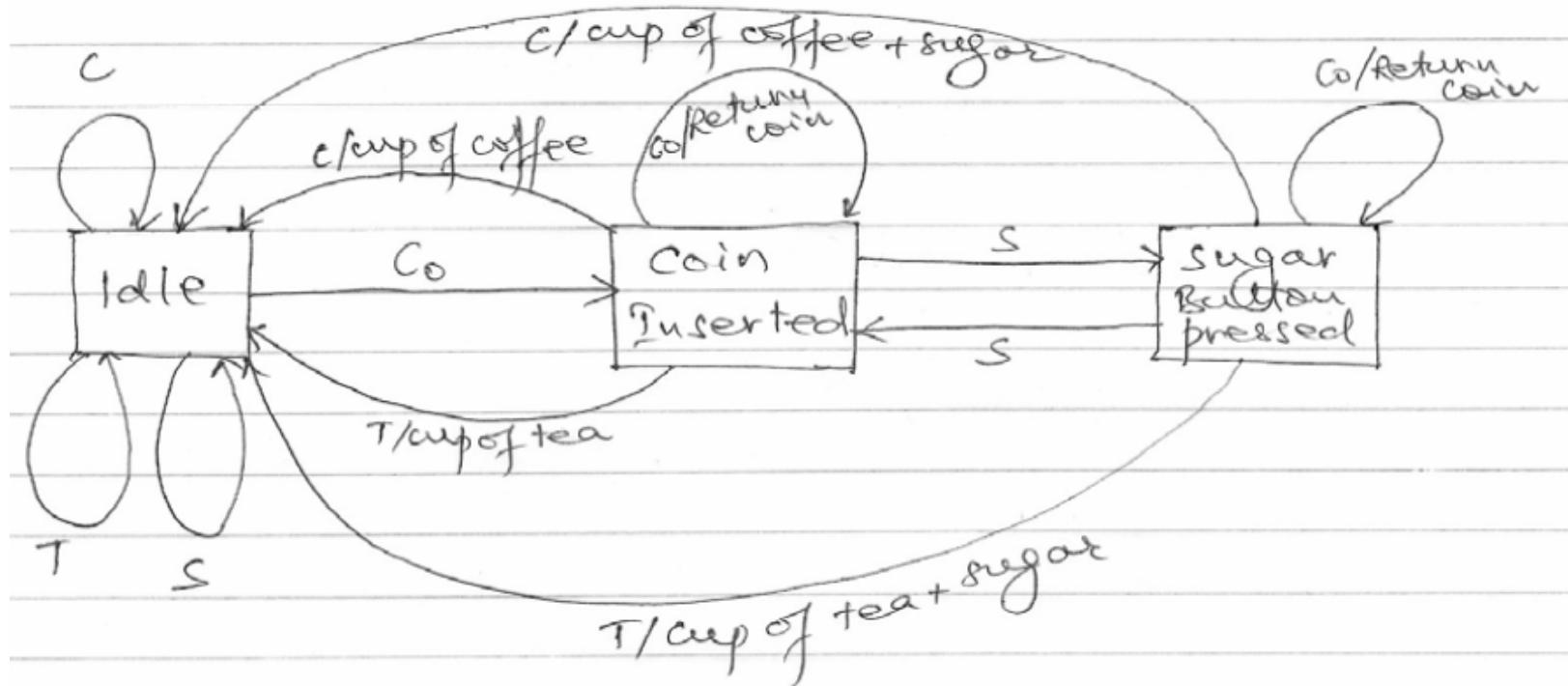


- Tea/Coffee vending Machine
- Options
 - Accepts token/coin
 - Sugar

Vending Machine



C: Coffee Button pressed
 T: Tea Button pressed
 S: Sugar Button pressed
 Co : Coin inserted





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 7 Self Study

- SS7.1 Compare & Contrast – FSM, State Charts and Petri Nets
- SS7.2 Choose a system or a product of day to day use. Model the same using FSM. Derive the test cases based on the Model.



SS 7.1 Compare & Contrast – FSM, State Charts and Petri Nets

7.1 Self Study

To explore:

- Compare & Contrast – FSM, State Charts and Petri Nets

Study Work:

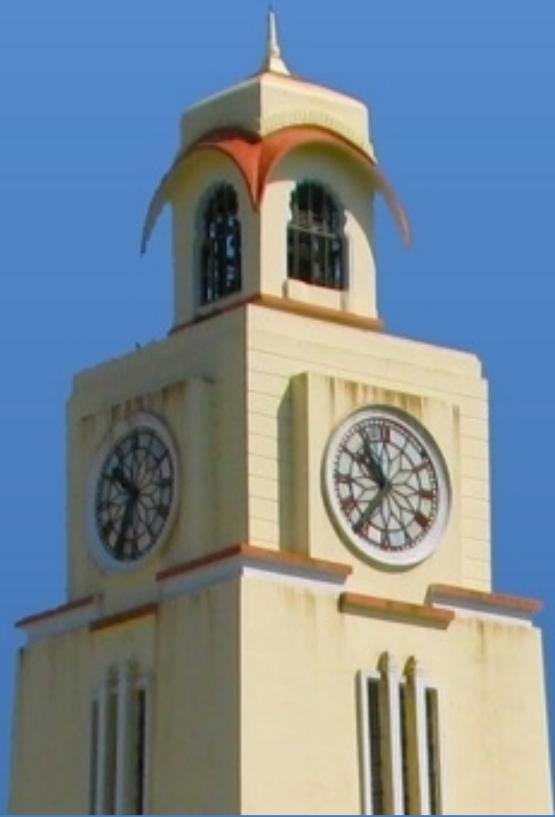
- Take up some among of research of literature on the three – FSM, StateCharts and Petri Nets
- Compare an contrast these techniques on at least 3 unique attributes



BITS Pilani

Software Testing Methodologies

Prashant Joshi



SS 7.2 Modelling a system/product of day-to-day use

7.2 Self Study

To explore:

- Model a system

Study Work:

- Choose a system of day to day use (example: Traffic Light, Lift)
- Model the system using one of the discussed techniques
- Derive the test cases for the system



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Model Based Testing

- Process of creating a **Model** results in deeper insights and understanding of the system
- Adequacy of MBT – depend on accuracy of the Model
- Sequence of steps
 - Model the system
 - Identify the threads of system behavior in the model
 - Transform threads into test cases
 - Execute the test cases (on actual system) and record the results
 - Revise the model(s) as needed and repeat the process

System of Systems - Characteristics

- A super system
- A collection of cooperating systems
- A collection of autonomous systems
- A set of component systems

Specific Attributes – Maier

- They are built from components that are (or can be) independent systems
- They have managerial/administrative independence
- They are usually developed in an evolutionary way
- They exhibit emergent (as opposed to preplanned) behaviours

Definitions (Maier)

- A directed system of systems is designed, built, and managed for a specific purpose
- A collaborative system of systems has limited centralized management and control
- A virtual system of systems has no centralized management and control

Larger Systems

- Garage Controller (Directed)
- Building Automation (Directed)
- Air Traffic Control Systems (Acknowledged)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 8: Agenda

Module 8: Model Based Testing (2/2)

Topic 8.1

Model Based Testing – Systems

Topic 8.2

Model Based Testing – System of Systems

Topic 8.3

Examples



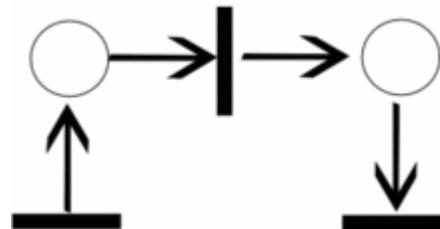
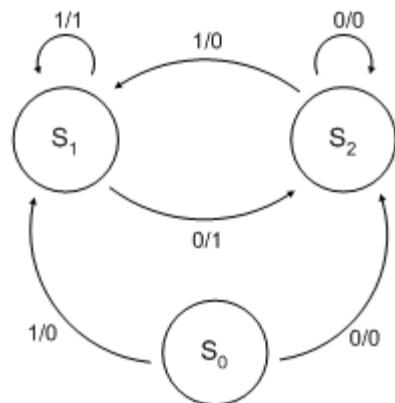
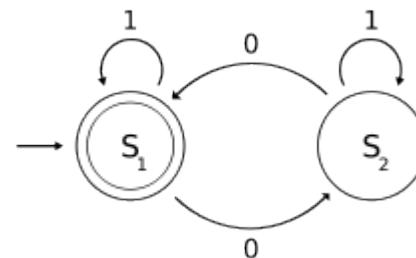
Topic 8.1: Model Based Testing – Systems

Model Based Testing

- Process of creating a **Model** results in deeper insights and understanding of the system
- Adequacy of MBT – depend on accuracy of the Model
- Sequence of steps
 - Model the system
 - Identify the threads of system behavior in the model
 - Transform threads into test cases
 - Execute the test cases (on actual system) and record the results
 - Revise the model(s) as needed and repeat the process

Executable Models

- Finite State Machines
- Petri Nets
- StateCharts





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 8.2: Model Based Testing – System of Systems

System of Systems - Characteristics

- A super system
- A collection of cooperating systems
- A collection of autonomous systems
- A set of component systems

Specific Attributes – Maier

- They are built from components that are (or can be) independent systems
- They have managerial/administrative independence
- They are usually developed in an evolutionary way
- They exhibit emergent (as opposed to preplanned) behaviours

Definitions (Maier)

- A directed system of systems is designed, built, and managed for a specific purpose
- A collaborative system of systems has limited centralized management and control
- A virtual system of systems has no centralized management and control

Larger Systems

- Garage Controller (Directed)
- Building Automation (Directed)
- Air Traffic Control Systems (Acknowledged)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 8.3: Examples



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 8.1 Systems & Systems of Systems

8.1 Self Study

To explore:

- Systems and Systems of Systems around us

Study Work:

- Review systems and system of systems around us
- Write down their characteristics and working (in terms of use cases)
- Further, analyse use of techniques learnt so far for designing test cases
- Document your findings in form of a whitepaper (IEEE format recommended)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Object Orientation

- Well-conceived objects encapsulate functions and data “that belong together”
- Composition
- Enable – Reuse without modification or additional testing

Units for OO

- A unit is the smallest software component that can be compiled and executed
- A unit is a software component that would never be assigned to more than one designer to develop

Implications of Inheritance

- Role of inheritance complicates the choice of classes as units

Class Flattening

- Flattening of classes
- A flattened class is an original class expanded to include all the attributes and operations it inherits

Issues with Flattening

- Uncertainty – flattened class is not part of final system
- Similar issue as instrumented code

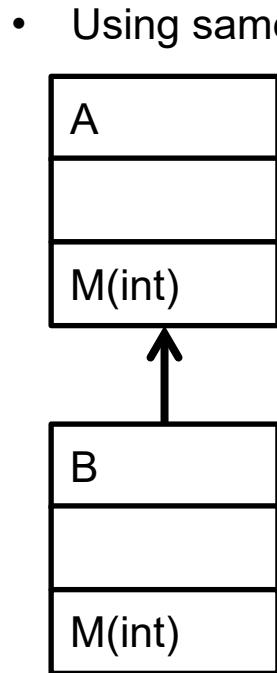
Polymorphism

- Using same name for different services

$$y=F(x)$$

$$z=F(x,y)$$

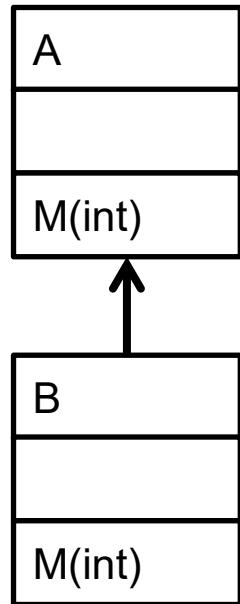
- Polymorphism related to inheritance



`A a;
B b;`

`a.M(x);
b.M(x);`

Polymorphism – Dynamic Objects



`A *pa;
B *pb;`

`pa=new A;
pb=new B;`

`pa->M(x); 1st
pb->M(x); 2nd
pa=pb;
pa->M(x); 1st`

Testing Constructors

- Constructor: A method that is executed when the object is created
- Constructors with no parameters
- Constructor with parameters

Strategy

- Create the object and check the state of the object

Testing Destructors

- Destructors: Always executed when object of class is destroyed
- Static Object: On program termination
- Dynamic Object: Object is disposed

Strategy

- Create the object and check the state of the object

Class Pair

```
Class Pair
{
public:
    void set_x(int);
    int get_x();
    void set_y(int);
    int get_y();
private:
    int x;
    int y;
}
```

Implementation:

```
Void Pair::set_x(int v)
```

```
{  
    if (v>0) x=v;  
}
```

```
Int Pair::get_x()
```

```
{  
    if (x>0) return x;  
    return 0;  
}
```

How do you test this class?

Use of test techniques

- Specification Based Testing
 - BVA
 - EC
 - DT
 - CEG
- Code Based Testing
 - Statement, Branch, Loop
 - McCabe Path, Basis Path
 - CF (Data and/or Control)
- Special Value Testing

Use of special methods

- Methods that view state of object
- Drivers (Input and Output)
- Introduce additional methods
 - Return results of execution of method
 - Set/Get values of some private data
- Test using a sequence of operations (data and methods)

Methods as Units

- This choice reduces OO to procedural unit testing (Method = Procedure)
- Need for stub and driver
 - Availability of other methods
- Look for messages

Classes as Units

- Entire class as unit solves intraclass problem
- Views of class testing
 - Static view: as we read the source code
 - Compile time view: Inheritance occurs
 - Execution view: Abstract classes
- Need for other classes
- Flattening of classes

Method or Class as Unit

- Explore both
- Come up with your reasons for making a choice
- Review examples in the text
- Apply them for the stack() and queue() data structures and their implementations

Currency Converter



INR Amount

Equivalent in

- US Dollar**
- Canadian Dollar**
- Euro**
- Pound**
- Chinese Yuan**

Compute

Clear

Quit



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 9: Agenda

Topic 9.1

**OO Software & OO Software test –
Introduction & Overview**

Topic 9.2

Issues in Testing OO Software

Topic 9.3

OO Unit Testing

Topic 9.4

Examples



Topic 9.1: OO Software & OO Software test – Introduction & Overview

Object Orientation

- Well-conceived objects encapsulate functions and data “that belong together”
- Composition
- Enable – Reuse without modification or additional testing

Object – An example

Dog as an object

State

- Age
- Name
- Color

Behaviour

- Barking
- Hungry
- Sleeping



Source: <http://retrieverman.files.wordpress.com/2011/09/german-shepherd-dog.jpg>

Thinking OO – Various Examples



- Point – Line
- Data Structures – Stack/Queue
- Employee Data
- ...

Levels – A discussion

- What is a unit?
- Implications of strategy of composition
- Inheritance, encapsulation & polymorphism
- Class, GUI, integration and system testing
- Data flow



BITS Pilani

Software Testing Methodologies

Prashant Joshi



Topic 9.2 : Issues in Testing OO Software

Units for OO

- A unit is the smallest software component that can be compiled and executed
- A unit is a software component that would never be assigned to more than one designer to develop

Units for OO

- Class as unit
- Integration – clear goals
 - To check the cooperation of separately tested classes

Implications of Inheritance

- Role of inheritance complicates the choice of classes as units

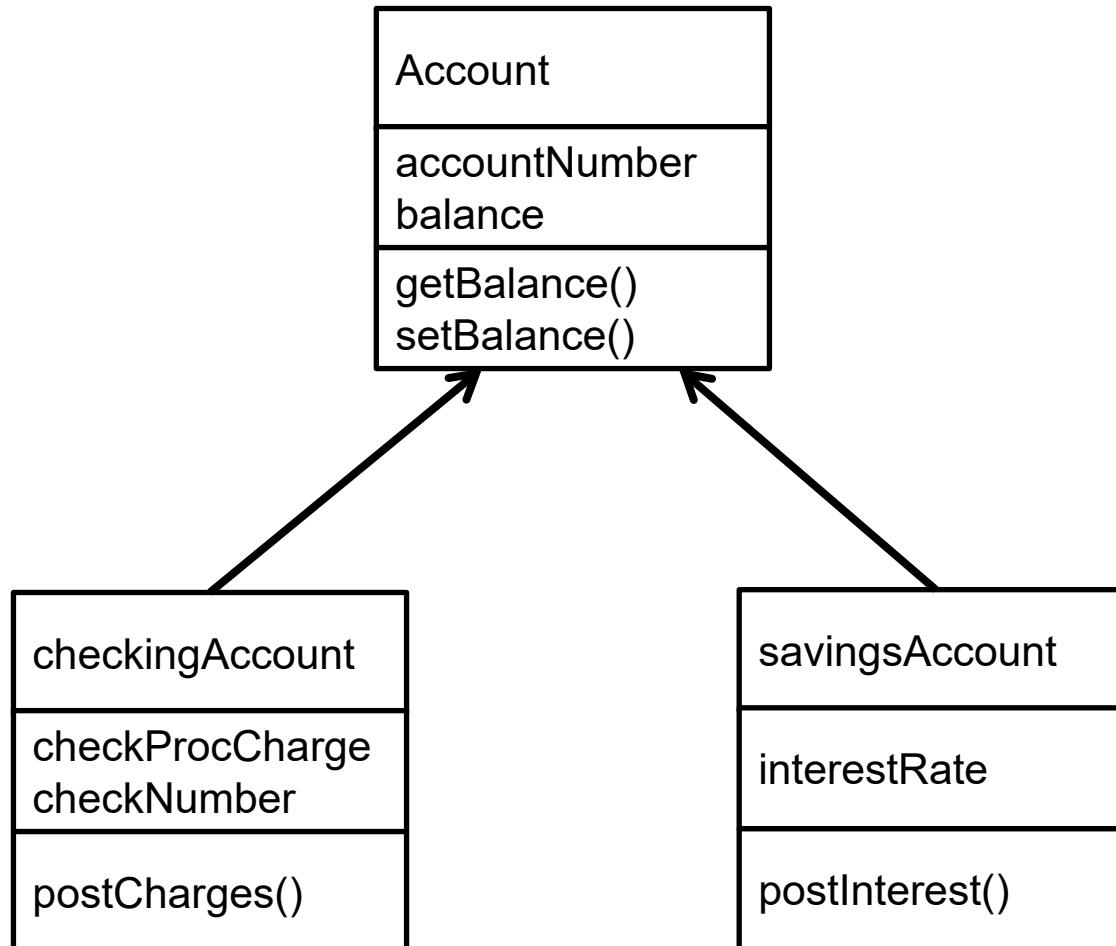
Class Flattening

- Flattening of classes
- A flattened class is an original class expanded to include all the attributes and operations it inherits

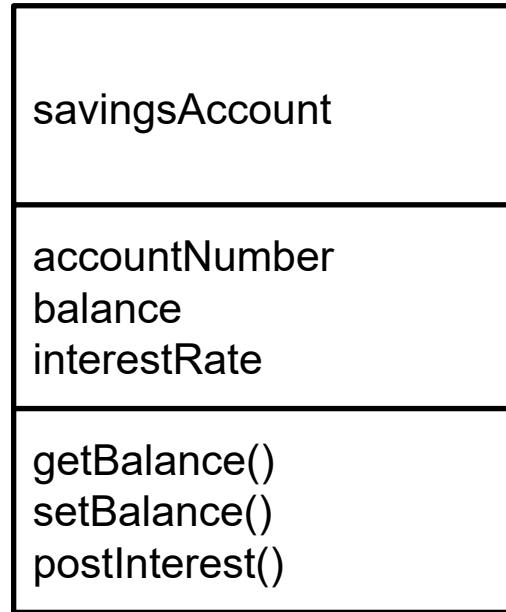
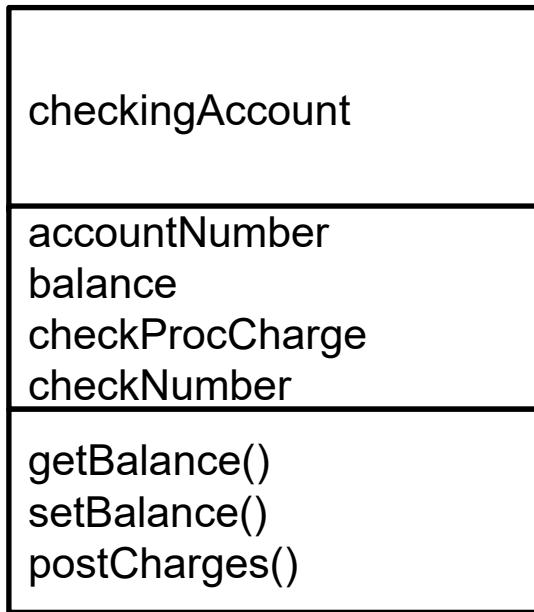
Issues with Flattening

- Uncertainty – flattened class is not part of final system
- Similar issue as instrumented code

Flattening Examples



Flattening Examples



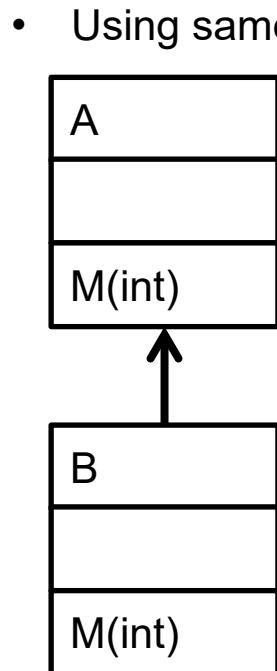
Polymorphism

- Using same name for different services

$$y=F(x)$$

$$z=F(x,y)$$

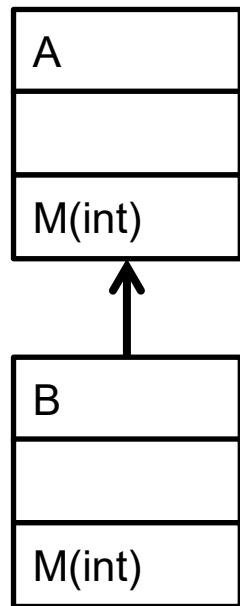
- Polymorphism related to inheritance



A a;
B b;

a.M(x);
b.M(x);

Polymorphism – Dynamic Objects



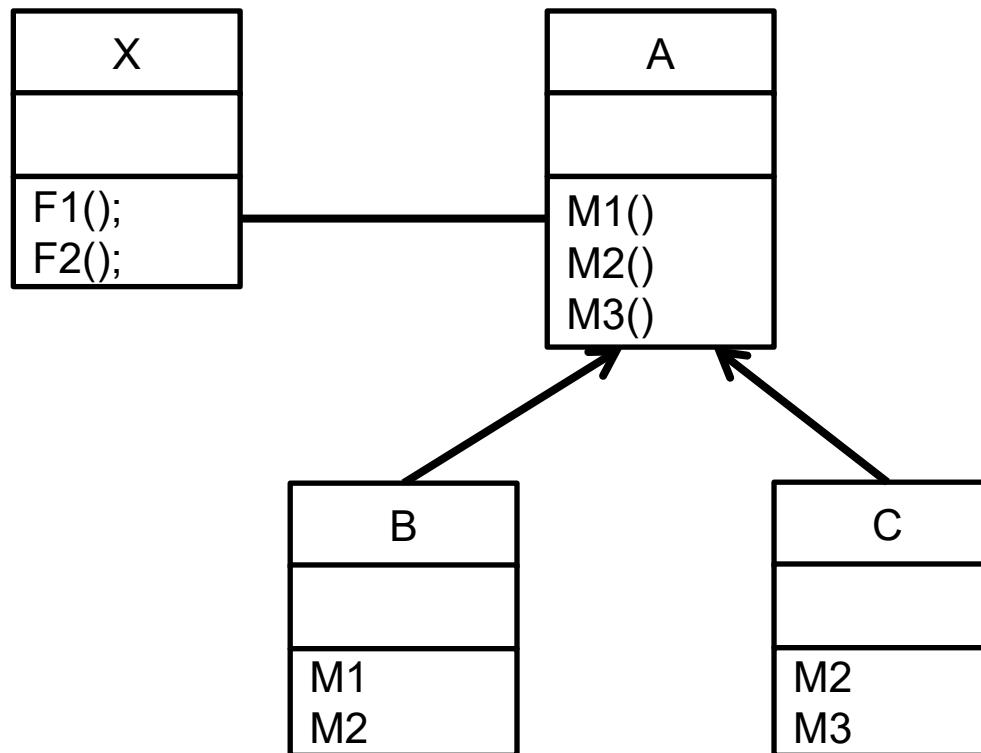
`A *pa;
B *pb;`

`pa=new A;
pb=new B;`

`pa□M(x); 1st
pb□M(x); 2nd
pa=pb;
pa□M(x); 1st`

Testing Polymorphism

Object of class A or B or C (Different bindings)



Testing Polymorphism

Testing class X

Method F1()

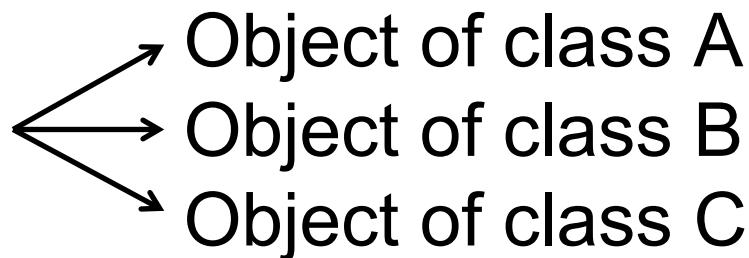
```
void F1()
{
    A *p;
    ...
    p->M1();
    ...
}
```

Testing Polymorphism

Testing class X

Method F1()

```
void F1()
{
    A *p;
    ...
    p->M1();
    ...
}
```



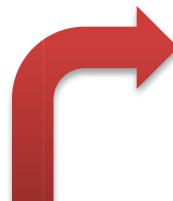
Object of class A
Object of class B
Object of class C

Testing Polymorphism

Testing class X

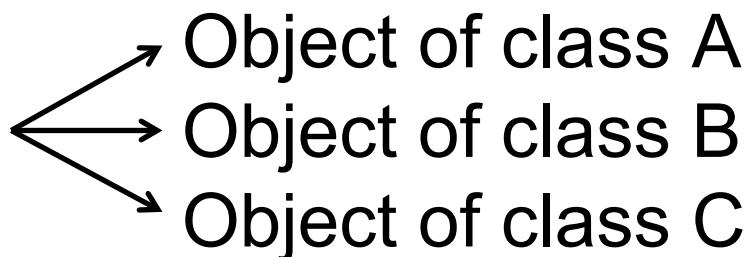
Method F1()

```
void F1()
{
    A *p;
    ...
    p->M1();
    ...
}
```



Thus

TC1: p->M1() on class A
TC2: p->M1() on class B
TC3: p->M1() on class C



Object of class A
Object of class B
Object of class C

Testing Polymorphism

- Concept: Develop a set of test cases for a client (class X) that exercise all client bindings to the polymorphic server. Test every polymorphic call.
- Write the code snippets for to depict the test case (HW)

Testing Constructors & Destructors



- Review and relook at testing Constructors and Destructors
- How?
- Challenges

Testing Constructors

- Constructor: A method that is executed when the object is created
- Constructors with no parameters
- Constructor with parameters

Strategy

- Create the object and check the state of the object

Constructors with no parameters

1. Create an Object

```
Y *p;
```

```
p = new Y
```

2. Check the results

```
p->checkstate()
```

```
delete p
```

Constructors with parameters

1. Create an Object

```
Y *p;
```

```
Input(a, b) // enter data
```

```
p = new Y(a, b)
```

2. Check the results

```
p->checkstate()
```

```
delete p
```

Testing Destructors

- Destructors: Always executed when object of class is destroyed
- Static Object: On program termination
- Dynamic Object: Object is disposed

Strategy

- Create the object and check the state of the object

Testing Destructors

Create and Destroy

```
Y *p;
```

```
p = new Y
```

```
...
```

```
delete p // Object is destroyed
```

Use checkstate() in the destructor



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 9.3 : OO Unit Testing

Object Orientation

- Well-conceived objects encapsulate functions and data “that belong together”
- Reuse without modification or additional testing

Units for OO – definition

- A unit is the smallest software component that can be compiled and executed
- A unit is a software component that would never be assigned to more than one designer to develop

Units for OO

- Class as unit
- Integration – clear goals
 - To check the cooperation of separately tested classes

Typical Class

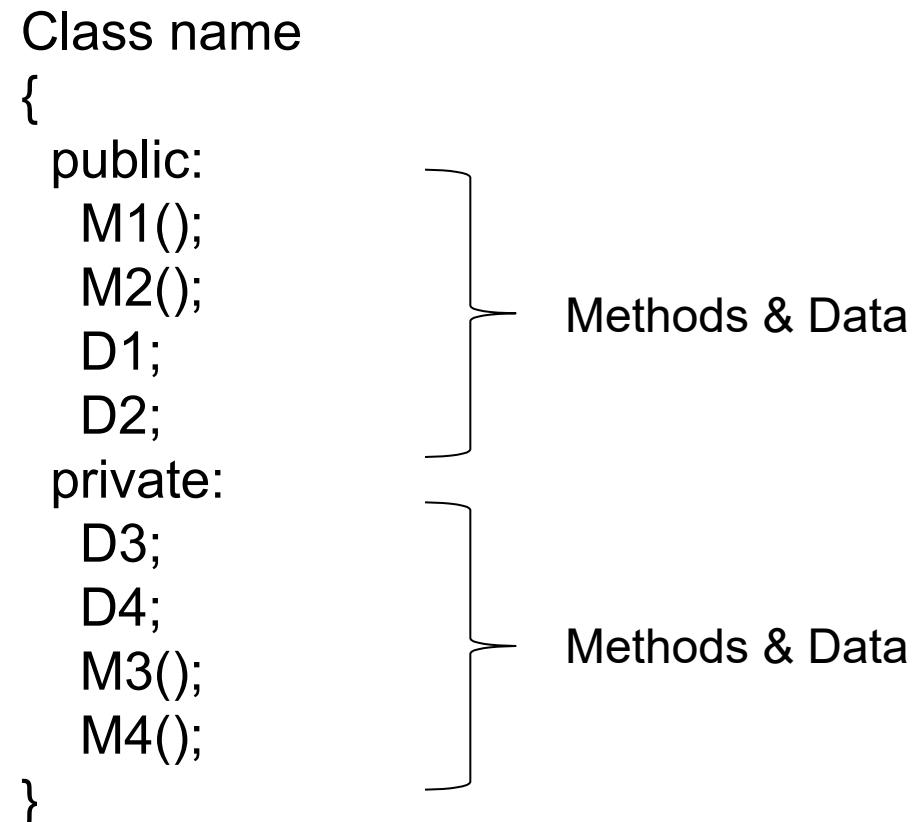
- A Class is a user defined data type
- Class has
 - Interface
 - Operations
 - Data

Typical Class

- A Class is a user defined data type
- Class has

- Interface
- Operations
- Data

```
Class name
{
    public:
        M1();
        M2();
        D1;
        D2;
    private:
        D3;
        D4;
        M3();
        M4();
}
```



Methods & Data

Methods & Data

Class Pair

```
Class Pair
{
public:
    void set_x(int);
    int get_x();
    void set_y(int);
    int get_y();
private:
    int x;
    int y;
}
```

Implementation:

```
Void Pair::set_x(int v)
```

```
{  
    if (v>0) x=v;  
}
```

```
Int Pair::get_x()
```

```
{  
    if (x>0) return x;  
    return 0;  
}
```

Class Pair

```
Class Pair
{
public:
    void set_x(int);
    int get_x();
    void set_y(int);
    int get_y();
private:
    int x;
    int y;
}
```

Implementation:

```
Void Pair::set_x(int v)
```

```
{  
    if (v>0) x=v;  
}
```

```
Int Pair::get_x()
```

```
{  
    if (x>0) return x;  
    return 0;  
}
```

How do you test this class?

Use of test techniques

- Specification Based Testing
 - BVA
 - EC
 - DT
 - CEG
- Code Based Testing
 - Statement, Branch, Loop
 - McCabe Path, Basis Path
 - CF (Data and/or Control)
- Special Value Testing

Use of special methods

- Methods that view state of object
- Drivers (Input and Output)
- Introduce additional methods
 - Return results of execution of method
 - Set/Get values of some private data
- Test using a sequence of operations (data and methods)

Develop these for class Pair and Stack

Methods as Units

- This choice reduces OO to procedural unit testing (Method = Procedure)
- Need for stub and driver
 - Availability of other methods
- Look for messages

Classes as Units

- Entire class as unit solves intraclass problem
- Views of class testing
 - Static view: as we read the source code
 - Compile time view: Inheritance occurs
 - Execution view: Abstract classes
- Need for other classes
- Flattening of classes

Method or Class as Unit

- Explore both
- Come up with your reasons for making a choice
- Review examples in the text
- Apply them for the stack() and queue() data structures and their implementations



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 9.4 : Examples

Typical Examples

- Queue
- Stack
- Automated Teller Machine
- Vending Machine
- Washing Machine
- UI of a Mobile Phone
- UI of Settop Box

Currency Converter

INR Amount

Equivalent in

- US Dollar**
- Canadian Dollar**
- Euro**
- Pound**
- Chinese Yuan**

Compute

Clear

Quit

Chapter References

- T1 Chapter 15
 - Review of examples is key to understanding the aspects
 - Lecture discussion
- Review the examples
 - OO Calendar
 - Currency Converter Application
- References
- Testing Object Oriented Systems: models, patterns and tools, Robert V Binder, Addison Wesley



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 9 Self Study

- SS9.1 Write a program in your chosen language (C++/Java) to sort a set of strings in alphabetical order (ascending or descending). Use OO methods. Design test cases for your work.
- SS9.2 Review the high level design of a UI framework like QT/GTK. Write a review on use of OO and testing techniques



SS 9.1 String sort

9.1 Self Study

To explore:

- Test techniques for OO
- Apply techniques for testing OO software

Study Work:

- You are required to write the program in your chosen language (C++/Java). Implement your own sorting algorithm.
- Create the test cases for the OO program using the techniques learnt
- Test your program using the test cases
- Analyse the results of the test run and comment on the effectiveness of the techniques



BITS Pilani

Software Testing Methodologies

Prashant Joshi



SS 9.2 Object Oriented UI frameworks study

9.2 Self Study

To explore:

- Application of testing techniques to OO software
- List the limitations that you come across

Study Work:

- Take up either GTK or QT as frameworks to study. Summarise their design. Explore the application of testing techniques to test the framework as well as applications developed using the framework
- Compare and contrast the effectiveness of the techniques to test the System



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 8.1 Systems & Systems of Systems

8.1 Self Study

To explore:

- Systems and Systems of Systems around us

Study Work:

- Review systems and system of systems around us
- Write down their characteristics and working (in terms of use cases)
- Further, analyse use of techniques learnt so far for designing test cases
- Document your findings in form of a whitepaper (IEEE format recommended)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 10: Agenda

Module 10: Object Oriented Testing (2/2)

Topic 10.1

OO Integration Testing

Topic 10.2

OO System Testing

Topic 10.3

OO – GUI Testing

Topic 10.4

Examples



Topic 10.1: OO Integration Testing

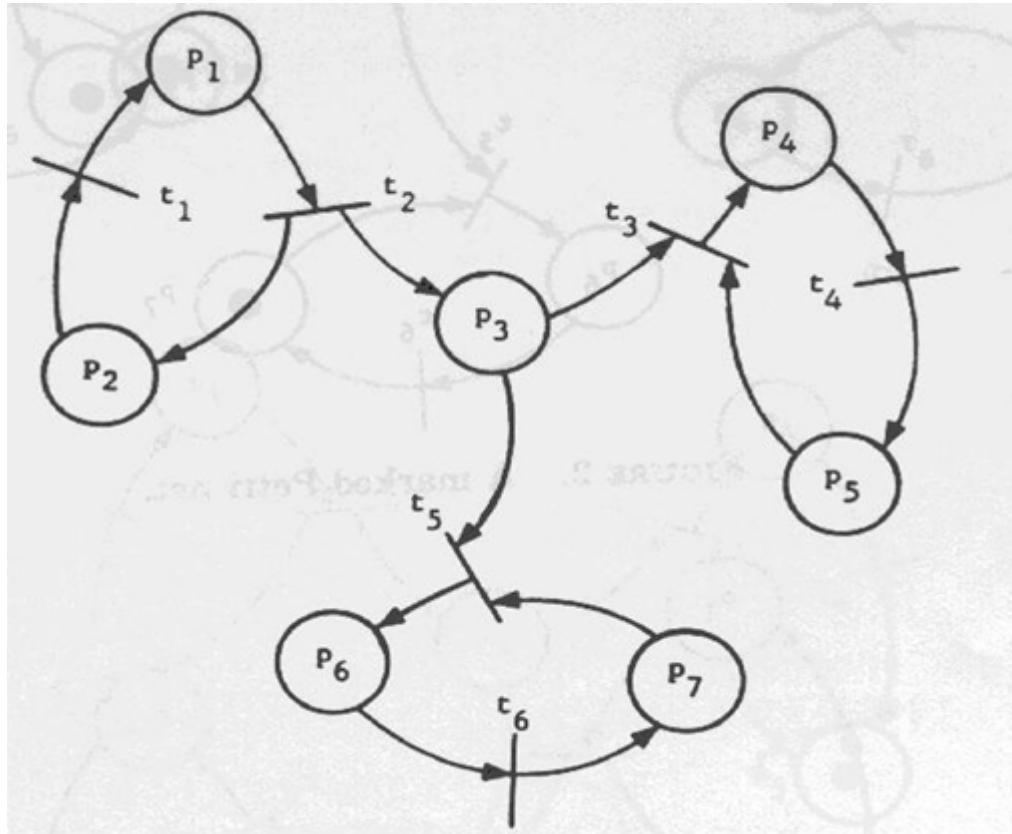
Petri Nets

- Major use:
 - Modeling of systems of events in which it is possible for some events to occur concurrently, but there are constraints on the occurrences, precedence, or frequency of these occurrences

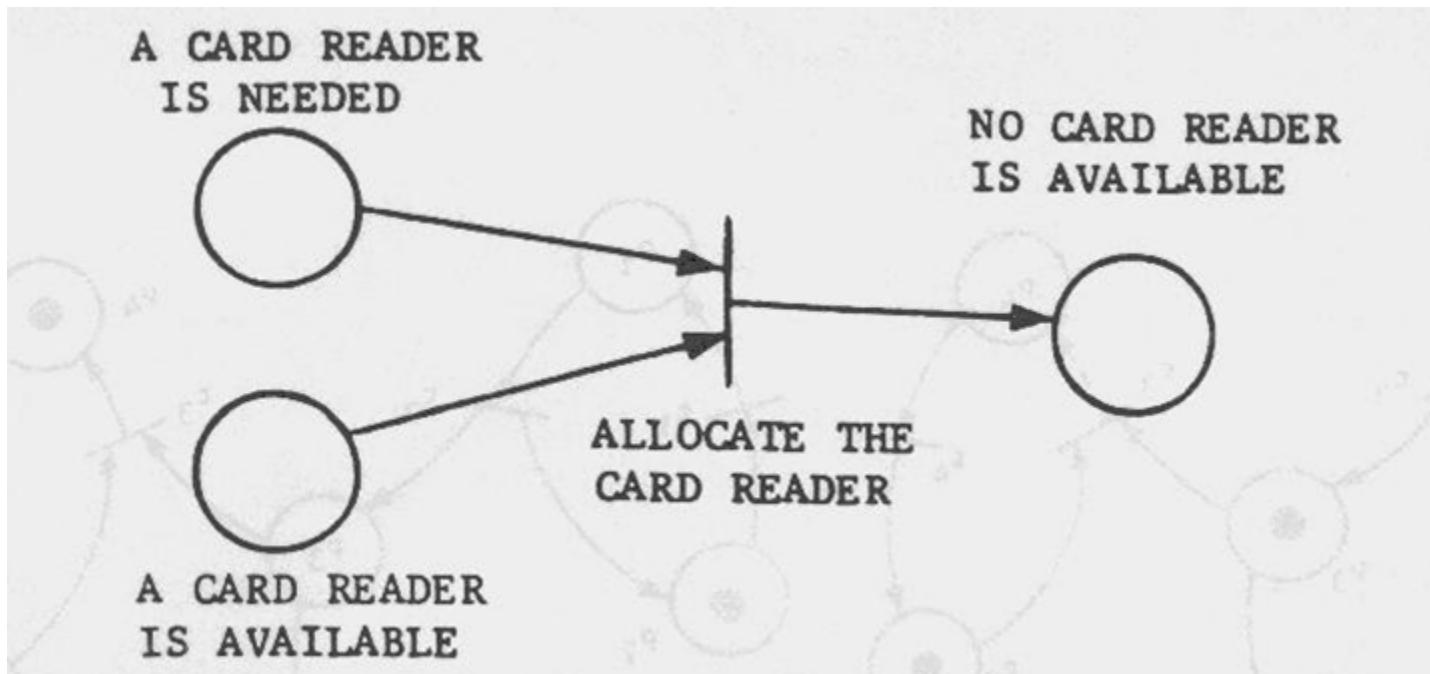
Petri Nets as a Graph

- Models static properties of a system
- Graph contains 2 types of nodes
 - Circles (Places)
 - Bars (Transitions)
- Petri net has dynamic properties that result from its execution
 - Markers (Tokens)
 - Tokens are moved by the firing of transitions of the net.

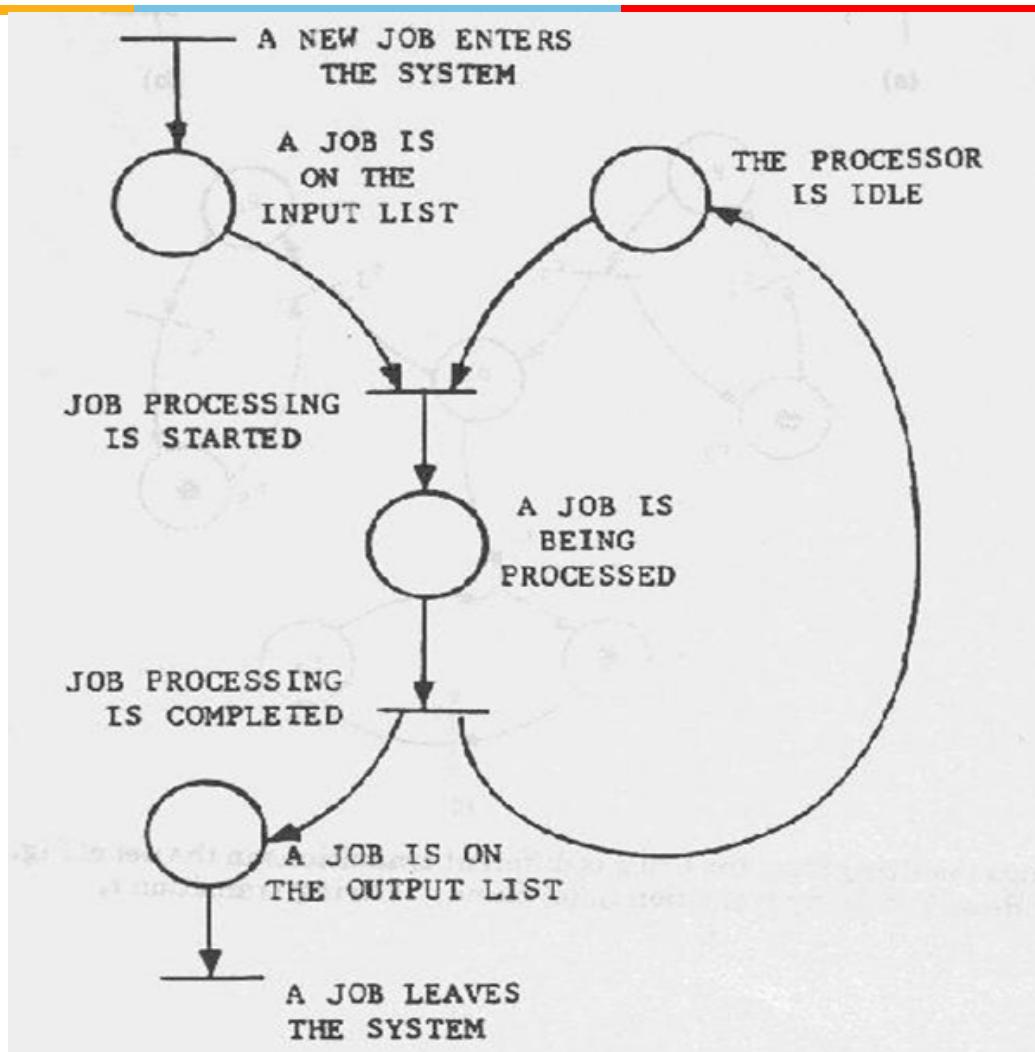
Graph representation



A Simple Model



Model Example

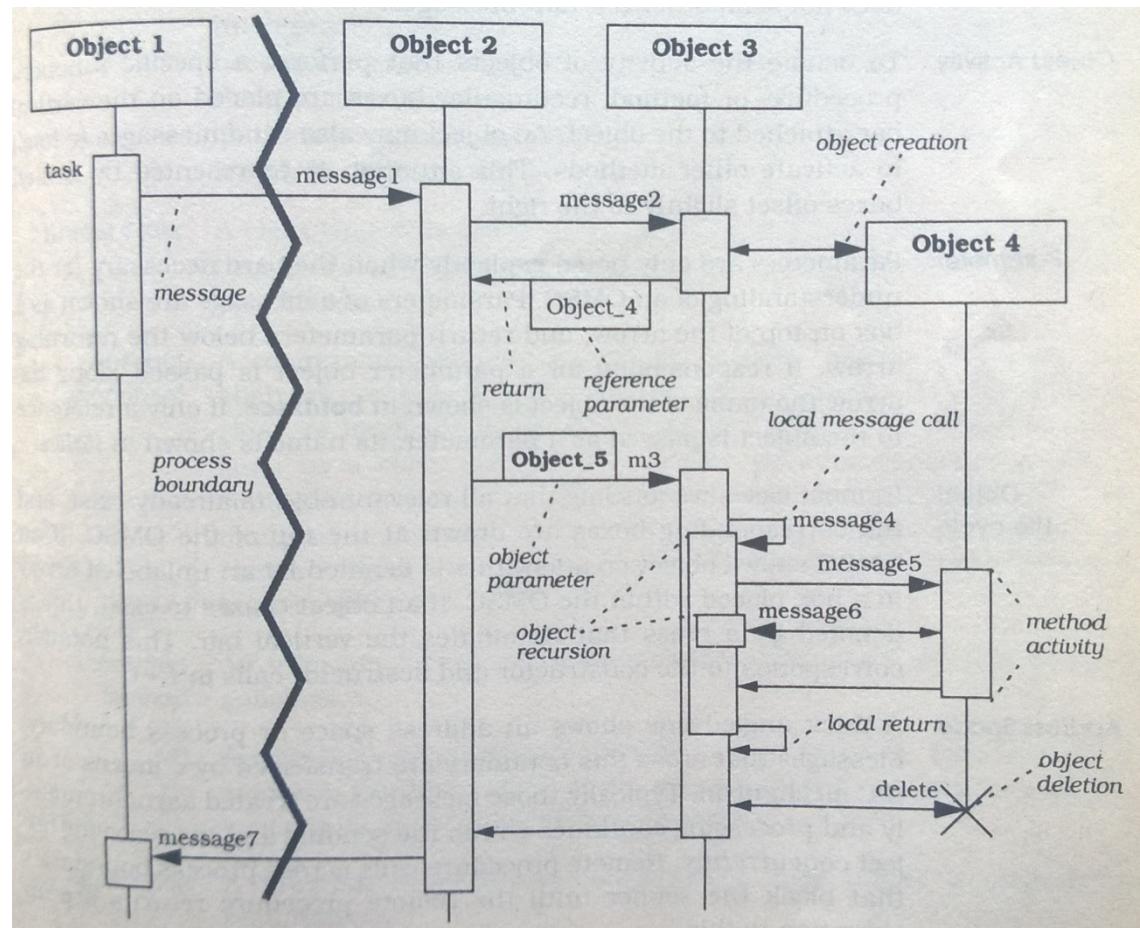


OO Integration Testing

- Flattened to Hierarchy
- Test methods to be removed
- Polymorphism – Test in each polymorphic context

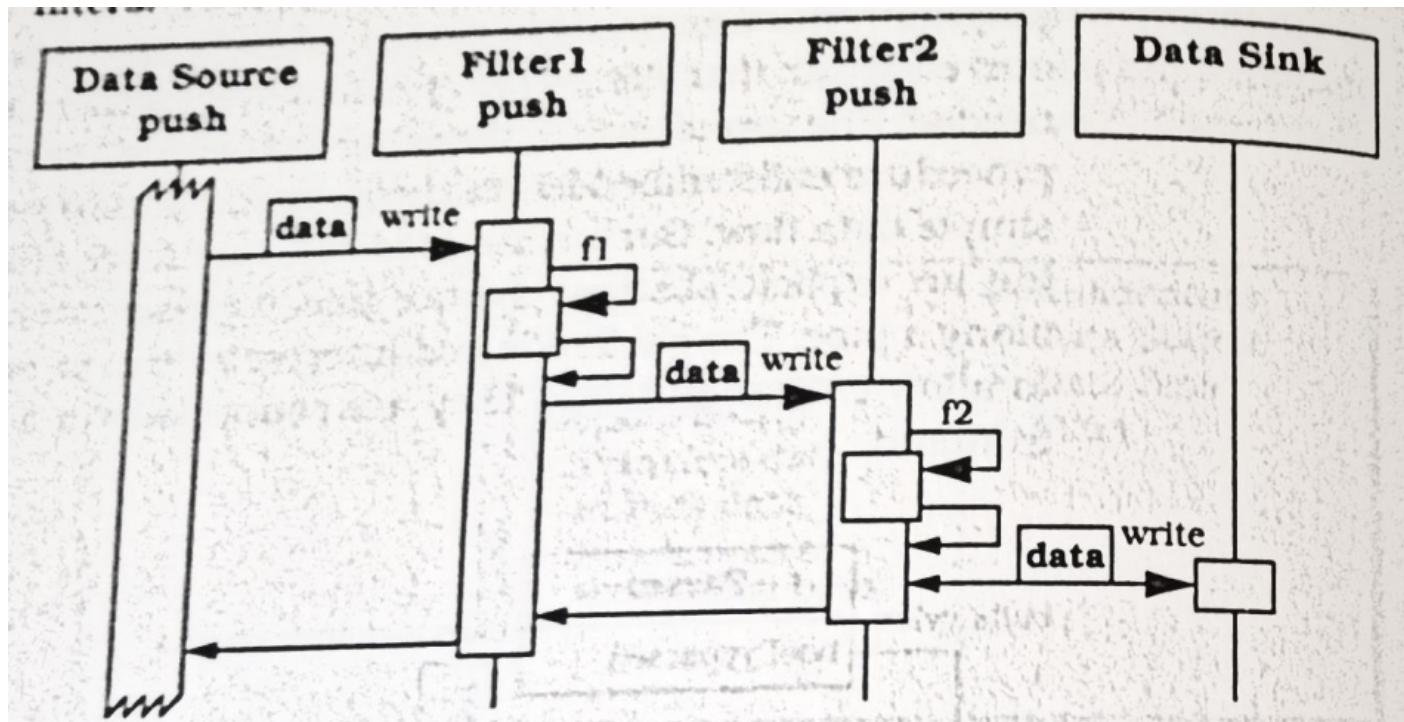
Message Sequence Charts – Dynamic View

Adapt MSC to demonstrate object or component interaction among the participants of a pattern



Example

- Scenario for Pipes & Filters
 - Push pipeline
 - Filter activity started by writing data to the filters



OO Integration

- MM Paths (Sequence of method execution linked by messages)
- Use of Data Flow



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 10.2: OO System Testing

System Testing

- System testing is concerned with the behaviour of a whole system. The majority of the functional failures should already have been identified during unit and integration testing
- System testing is usually considered appropriate for comparing the system to the non-functional system requirements, such as security, speed, accuracy, and reliability. External interfaces to other applications, utilities, hardware devices, or the operating environment are also evaluated at this level.

System Testing

- It is essential to define what the “system” is made up of
 - A system for one team/organisation may mean a module/part of a larger system
- Generally system level is treated as closest to everyday experience i.e. system (Product) as seen in the hands of the user
- In system testing apart from finding faults the goal is to demonstrate
 - System works
 - System is reliable
 - System is durable
 - Quality of system for defined parameters
- Use of specification based testing techniques to design test cases

System Testing – Aspects

- Thread
 - Thread Possibilities
 - Thread Definitions
 - Atomic Thread Function

System Requirements – Basic Concepts



- Data
 - Information used and created by the system
- Actions
 - Transform, data transform, control transform, process, activity, task, method, service
- Devices
 - Source and destination of system level inputs and outputs
- Events
 - System level I/O; occurs on port device; inputs and outputs of actions
- Threads
 - Model of a system – interactions among data, events and action

OO System Test

- Test Engineer treats system as Black Box!
- Look for input and output events
- OO or Procedural Implementation
- Requirements model
- Use of Petri nets

Focus of Our Testing

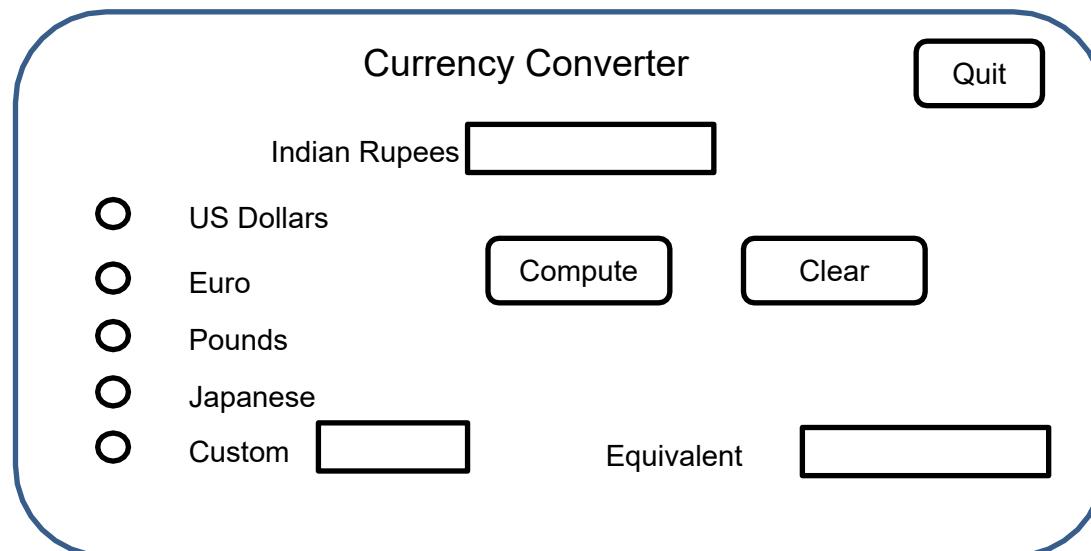
- Requirements
- System Functions
- Presentation Layer
- The way the user or customer sees it!

System Functions

- Use of description by customer/user in general terms
- User Stories Use Cases
- Gives rise to system functions
 - Evident (Input currency amount)
 - Hidden (Maintain ex-or relationship among countries)
 - Frill (Display of country flags)

Presentation Layer

- Sketching of UI
- Relating User Experience to the user stories and use cases (Making it simple and easy to use)
- Screen transition and Navigation



Use Case Based Testing

- High level Use Cases
 - Essential Use Cases
 - Expanded Essential Use Cases
-
- Each level bring in more refinement in what is done as part of the use case
 - Refine only as much it is required

Example: Use Case

HLUC

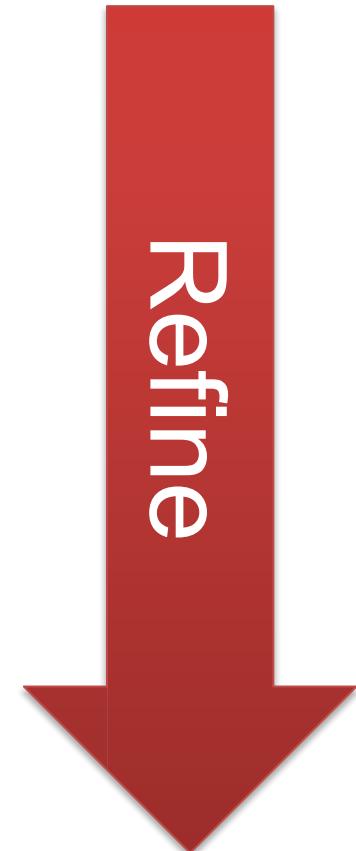
- User Starts the currency converter application in Windows

EUC

- Add an event sequence
- Input: User starts the application by using Run command, double click the application icon
- Output: The CC application GUI appears on the screen

EEUC

- Add next level details
- Details like: Which input box or area has focus. Specifics on static or dynamic data shown on the screen. Any details on memory or specifics of post-condition to be checked.



State Based

- State Based Testing
- Use of Extended Finite State Machine



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 10.3: OO – GUI Testing

Currency Converter System

Currency Converter Quit

Indian Rupees

US Dollars Compute Clear

Euro

Pounds

Japanese

Custom Equivalent

Top areas of focus

- Main characteristic of GUI Testing: Event Driven
 - Users can cause any of several events in any order
- Buttons have functions; and they can be tested
- Focus: test the event driven nature: System Testing of GUI
 - Also at level of unit one can test the function of buttons
- Use of Finite State Machines and Extended Finite State Machines

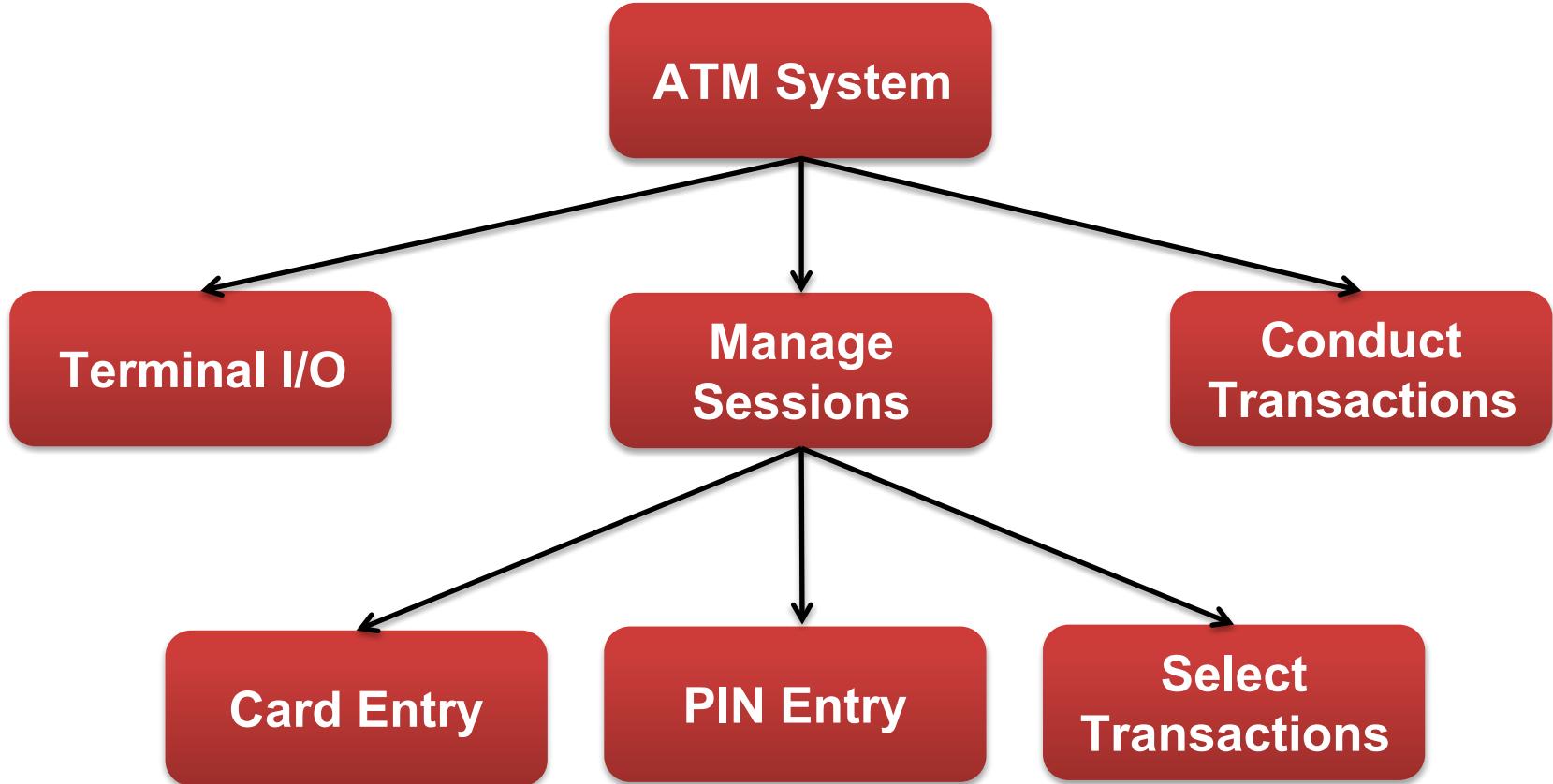
Testing a GUI Application

- Identify all the user input events
- Identify all the system output events
 - Externally visible & observable
- Use the System Level State Machine (State Chart)
 - Use of artificial events (Idea is to simplify the system level state machine)

Unit & Integration Testing

- Example of currency converter program
- Unit Level Testing (Level: Open for discussion)
 - Use of Input and Output for that “unit”
 - System level unit testing! (This poses some issues)
 - Test Driver is preferred for unit testing (Ensures testing of input, output and/or computation)
 - Method or Class as Unit (Remember out discussion on choosing either)
- Integration Level Testing
 - With unit level thoroughly tested; is integration required?
 - Definition of “integration level”

Automated Teller Machine



Source: T1: Figure 12.2



Software Testing Methodologies

BITS Pilani

Prashant Joshi

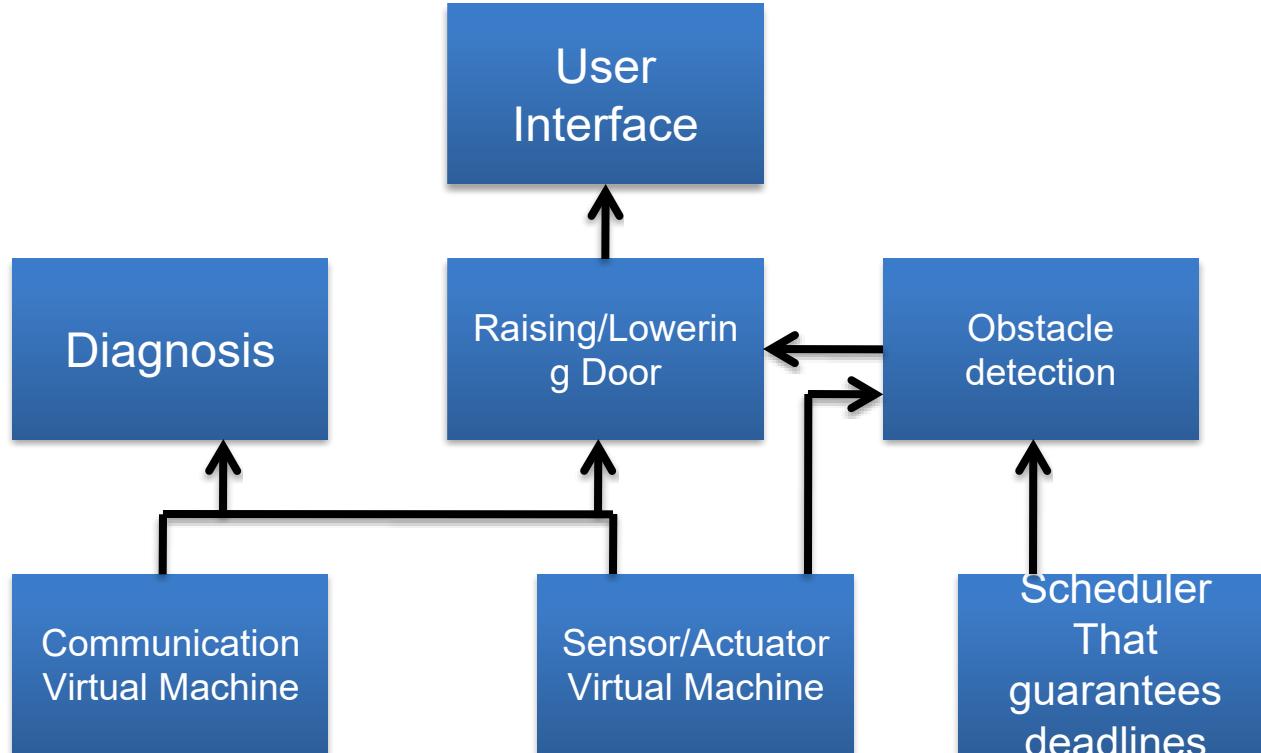


Topic 10.4: Examples

Garage Door

- The device controls for opening and closing the door are different for the various product lines. They may include controls from within a home information system. The product architecture for a specific set of controls should be directly derivable from the product line architecture
- The processor used in different processes will differ. The product architecture for each specific processor should be directly derivable from the product line architecture
- If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) within 0.1 second
- The garage door opener should be accessible for diagnosis and administration from within the home information system using product specific diagnosis protocol. It should be possible to directly product an architecture that reflects this protocol

Garage Door





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Higher Levels of Testing

- Integration Testing
- Sub-system Testing
- System Testing
- Product Testing
- *And More*

Integration Testing

- Integration testing is the process of verifying the interaction between software components. Classical integration testing strategies, such as top-down or bottom-up, are used with traditional, hierarchically structured software
- Modern systematic integration strategies are rather architecture-drive, which implies integrating the software components or subsystems based on identified functional threads.

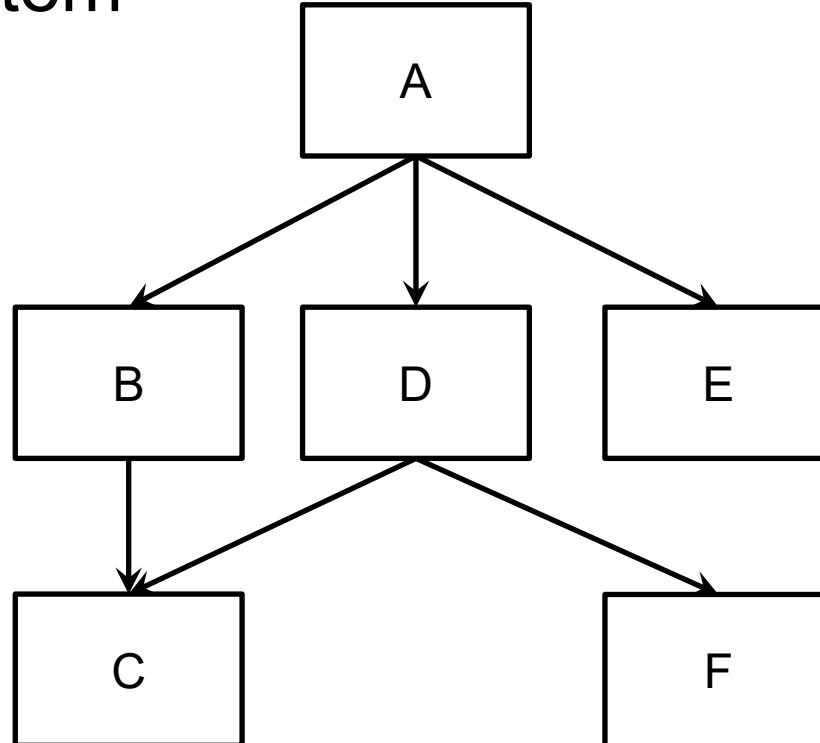
Source: SWEBOK 2004

Integration Testing

- Integration of Software
- Individually test “units” (components or modules or sub-systems)
- Structural
- Behavioural
- Alternative ways
 - Process driven
 - Customer commitment driven
 - Phased product release

Integration of Software

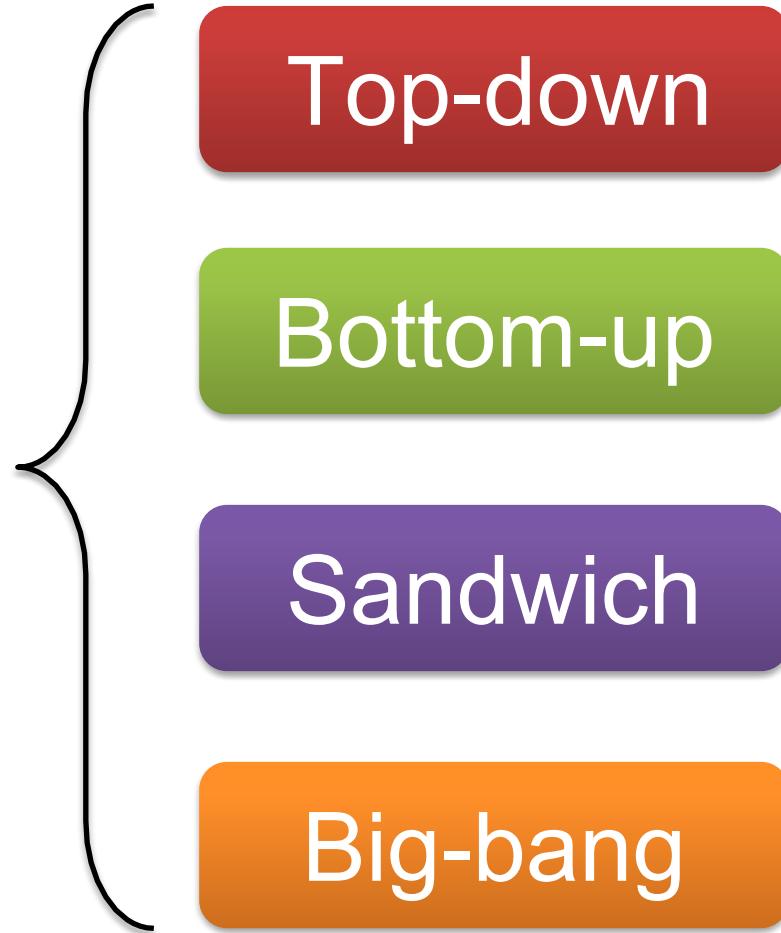
- Integration is concerned with integration of individual components, called “units”, to form the entire system



Decomposition Based Integration



Decomposition
Based Integration



Pros and Cons

- Need of Drivers
- Need of Stubs
- Number of Integration Cycles/Sessions
 - Sessions = nodes – leaves + edges
 - Number of stubs or drivers required
- Number of integration test cases and their runs
- Detection of defects
 - Early
 - Separation of defective module/component

Management driven processes versus engineering and development driven process

Comparison between DBI

Category	BU	TD	BB	S
Drivers	Yes	No	No	In-part
Stubs	No	Yes	No	In-part
Early Working version of the system	Late	Early	Late	Early
Early detection of interface errors	Early	Early	Late	Early
Parallel Testing <i>Note: Individual testing of drivers and stubs will need to be considered separately</i>	Medium	Low	High	Medium

Call Graph-Based Integration



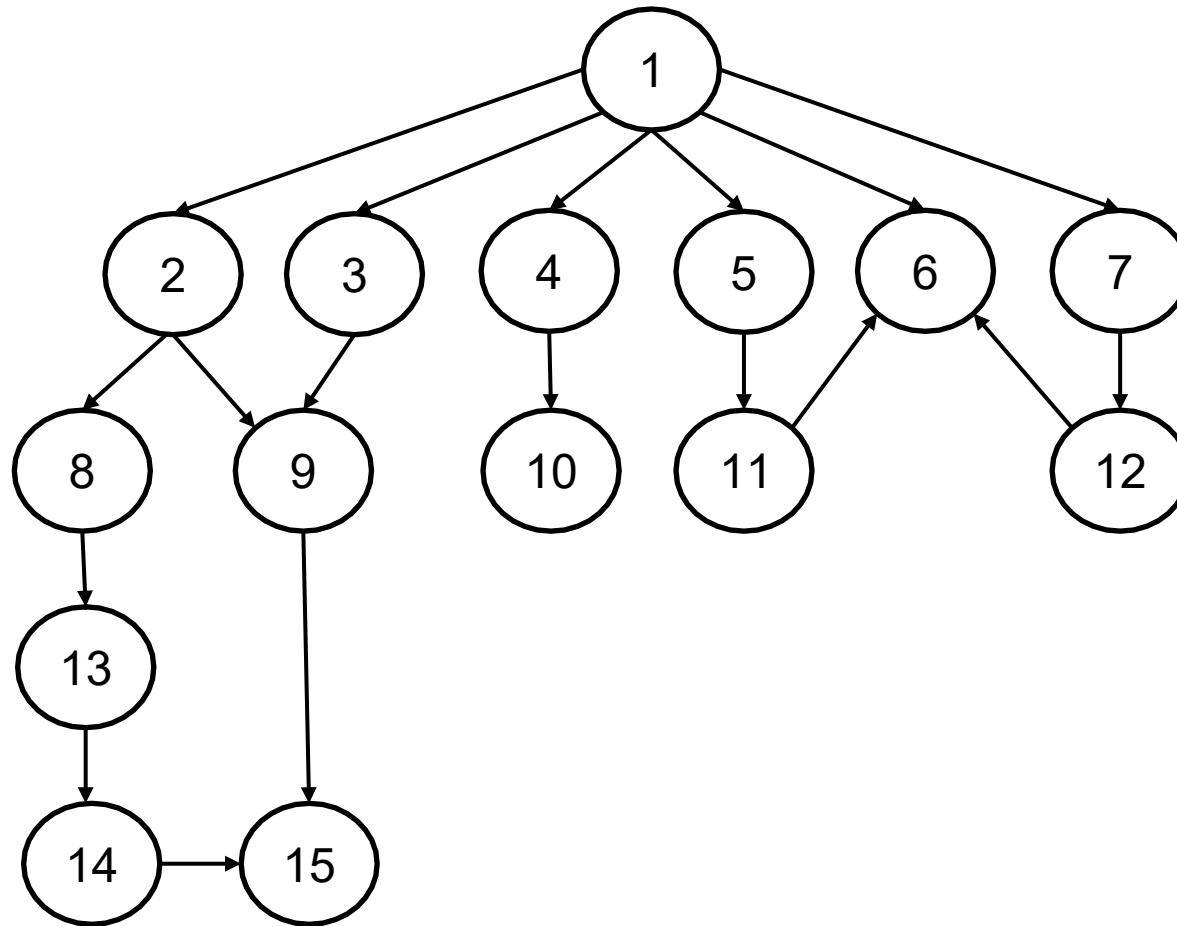
Call Graph-Based
Integration

Pair-Wise
Neighbourhood

Path-Based Integration

- Desirable Structural and functional testing
- Express testing in terms of behavioural threads
- Focus on interaction (instead of interfaces)
 - Co-functioning
 - Interfaces are structural
 - Interactions is behavioural

A Sample Program



Refer: Page 239 of T1



Software Testing Methodologies

BITS Pilani

Prashant Joshi

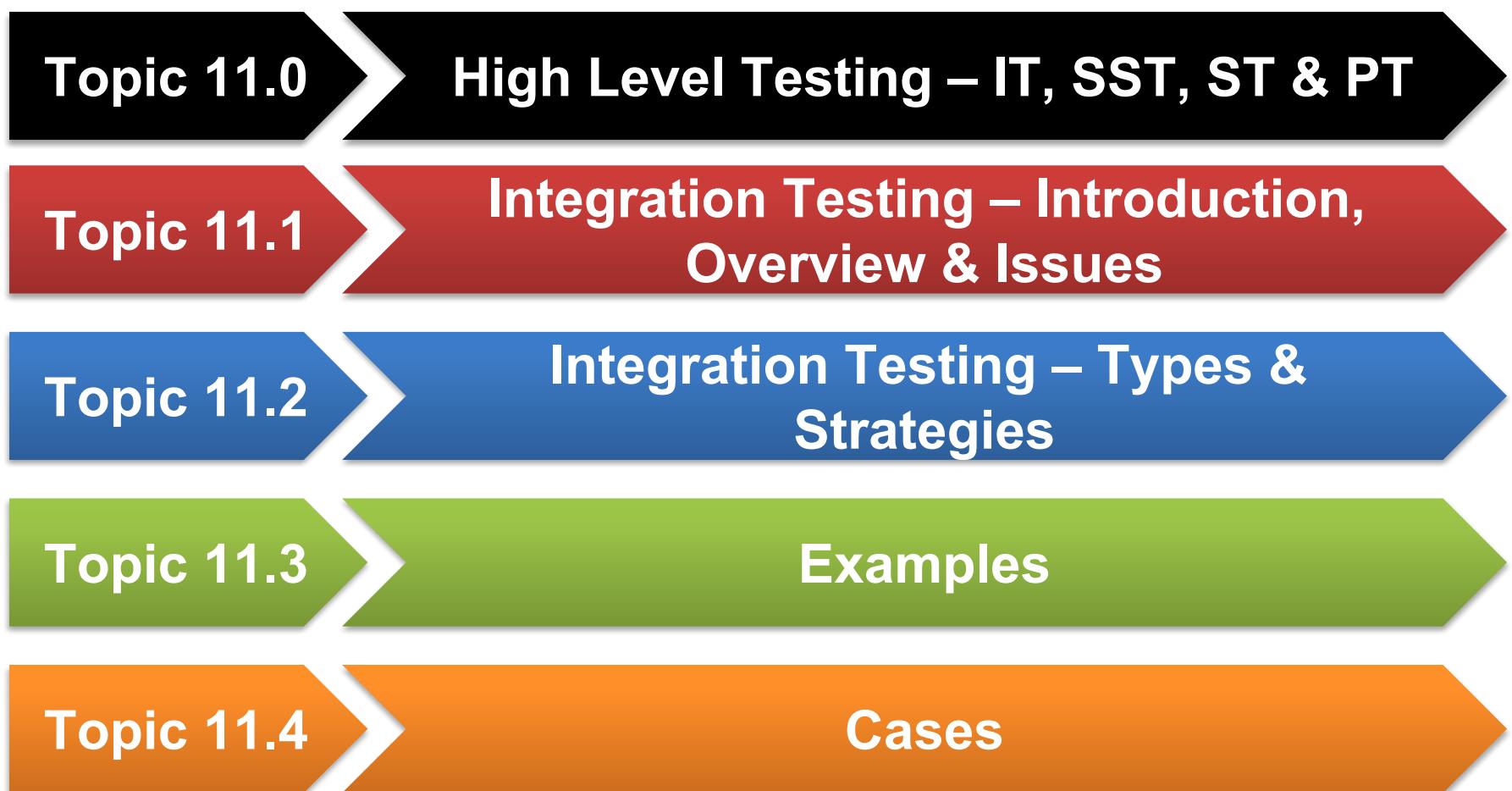


Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 11: Agenda





Topic 11.0 High Level Testing – IT, SST, ST & PT

Higher Levels of Testing

- Integration Testing
- Sub-system Testing
- System Testing
- Product Testing
- *And More*



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 11.1: Integration Testing – Introduction, Overview & Issues

Integration Testing

- Integration testing is the process of verifying the interaction between software components. Classical integration testing strategies, such as top-down or bottom-up, are used with traditional, hierarchically structured software
- Modern systematic integration strategies are rather architecture-drive, which implies integrating the software components or subsystems based on identified functional threads.

Source: SWEBOK 2004

Insights

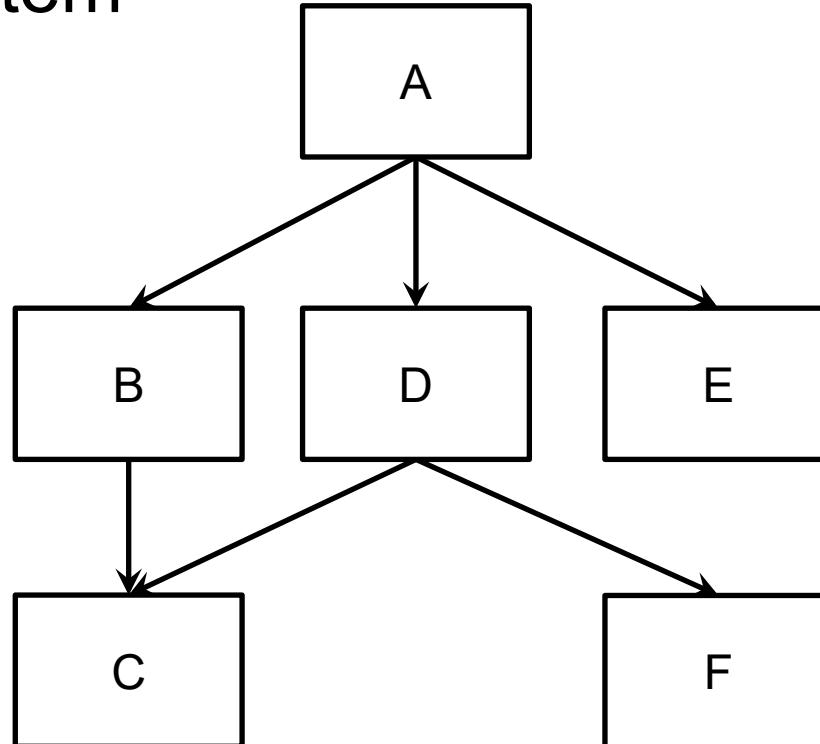
- Structural
- Behavioural

Integration Testing

- Integration of Software
- Individually test “units” (components or modules or sub-systems)
- Structural
- Behavioural
- Alternative ways
 - Process driven
 - Customer commitment driven
 - Phased product release

Integration of Software

- Integration is concerned with integration of individual components, called “units”, to form the entire system





Software Testing Methodologies

BITS Pilani

Prashant Joshi

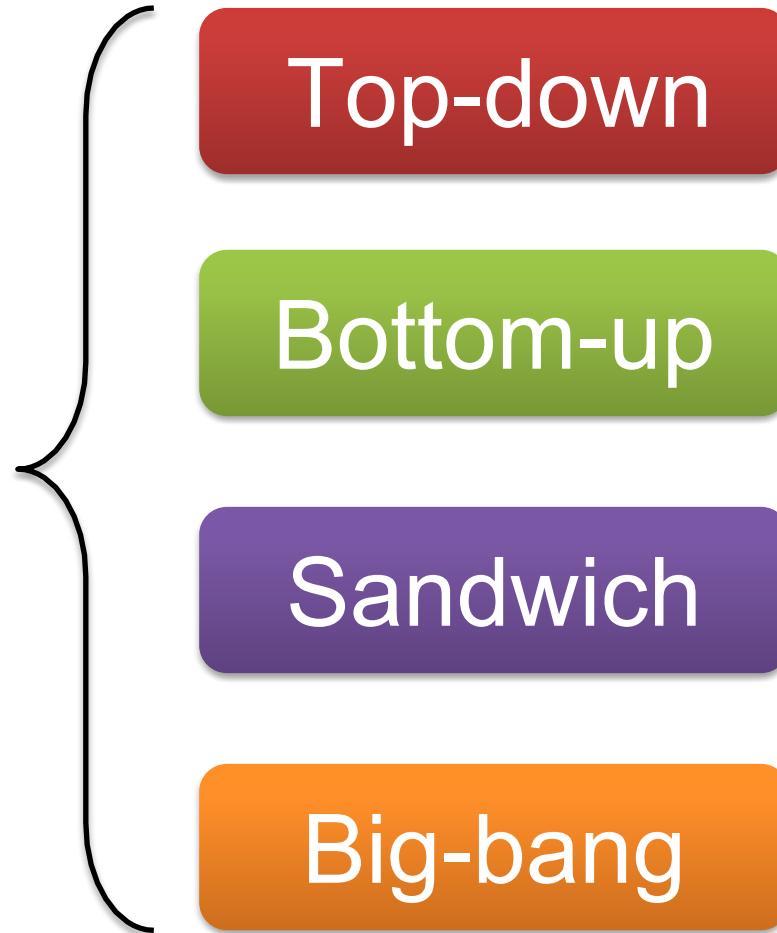


Topic 11.2 : Integration Testing – Types & Strategies

Decomposition Based Integration

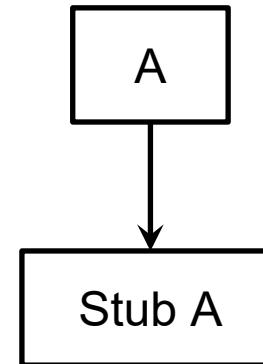
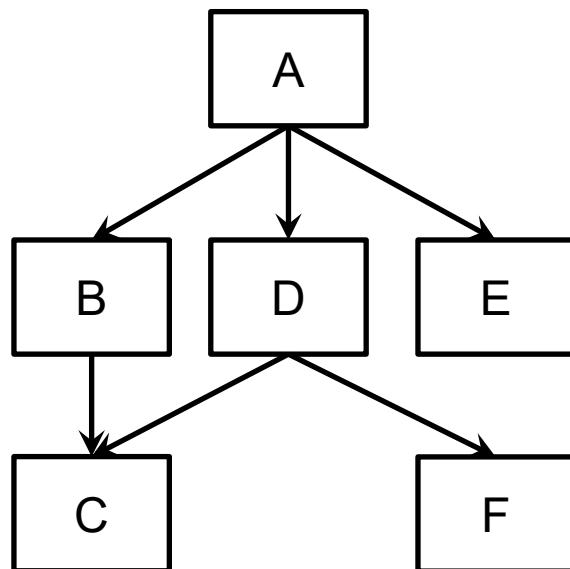


Decomposition
Based Integration



Top Down

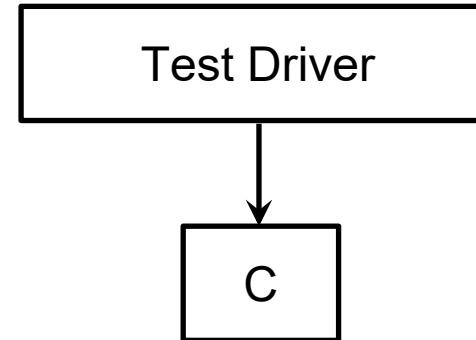
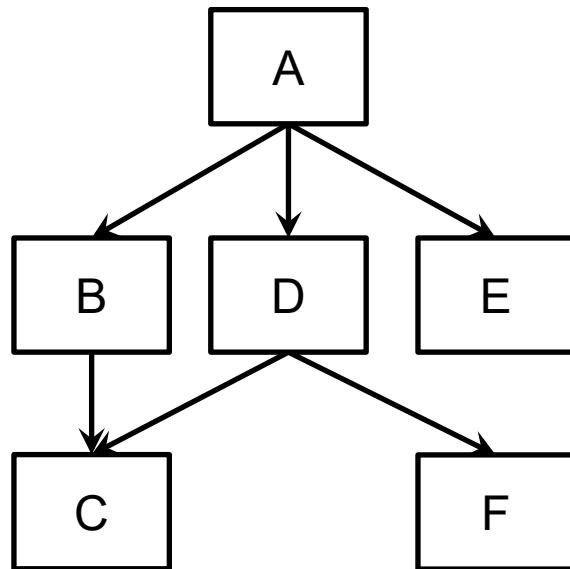
- A system is integrated Top to Bottom
- Integration Sequence: A B D E C F



A stub is a component that is used to simulate the system

Bottom Up

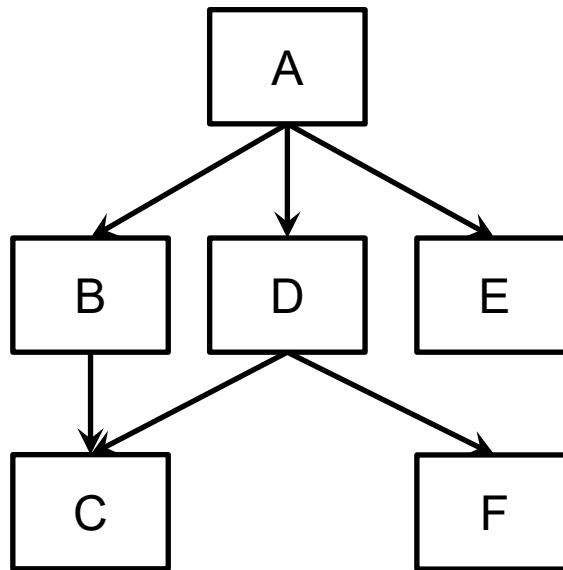
- A system is integrated Bottom to Top
- Bottom Units: C, D and F
- Integration Sequence: C D F B E A



A test driver is used to test (execute) the unit

Sandwich

- Combination of BU and TD



Big Bang

- Each unit is implemented (and separately tested)
- Integrate all units together to form the entire system
- Test the entire(integrated) system

Pros and Cons

- Need of Drivers
- Need of Stubs
- Number of Integration Cycles/Sessions
 - Sessions = nodes – leaves + edges
 - Number of stubs or drivers required
- Number of integration test cases and their runs
- Detection of defects
 - Early
 - Separation of defective module/component

Management driven processes versus engineering and development driven process

Comparison between DBI

Category	BU	TD	BB	S
Drivers	Yes	No	No	In-part
Stubs	No	Yes	No	In-part
Early Working version of the system	Late	Early	Late	Early
Early detection of interface errors	Early	Early	Late	Early
Parallel Testing <i>Note: Individual testing of drivers and stubs will need to be considered separately</i>	Medium	Low	High	Medium

Call Graph-Based Integration

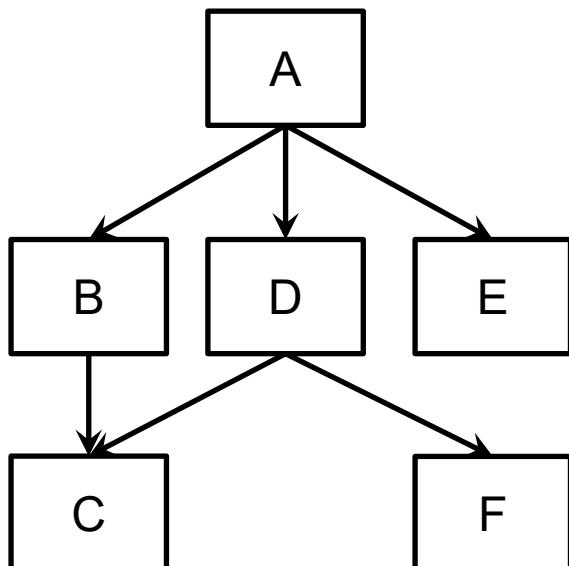


Call Graph-Based
Integration

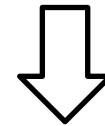
Pair-Wise
Neighbourhood

Neighbourhood

- One edge away
- Predecessor and Successor



Interior nodes = nodes – (source nodes + sink nodes)
Neighbourhoods = interior nodes + source nodes



Neighbourhoods = nodes – sink nodes

Pros and Cons

- Move away from purely structural basis to behavioural basis
- Number of Integration Cycles/Sessions
- Number of integration test cases and their runs
- Detection of defects
 - Early
 - Separation of defective module/component is at times a challenge

Path-Based Integration

- Desirable Structural and functional testing
- Express testing in terms of behavioural threads
- Focus on interaction (instead of interfaces)
 - Co-functioning
 - Interfaces are structural
 - Interactions is behavioural



Software Testing Methodologies

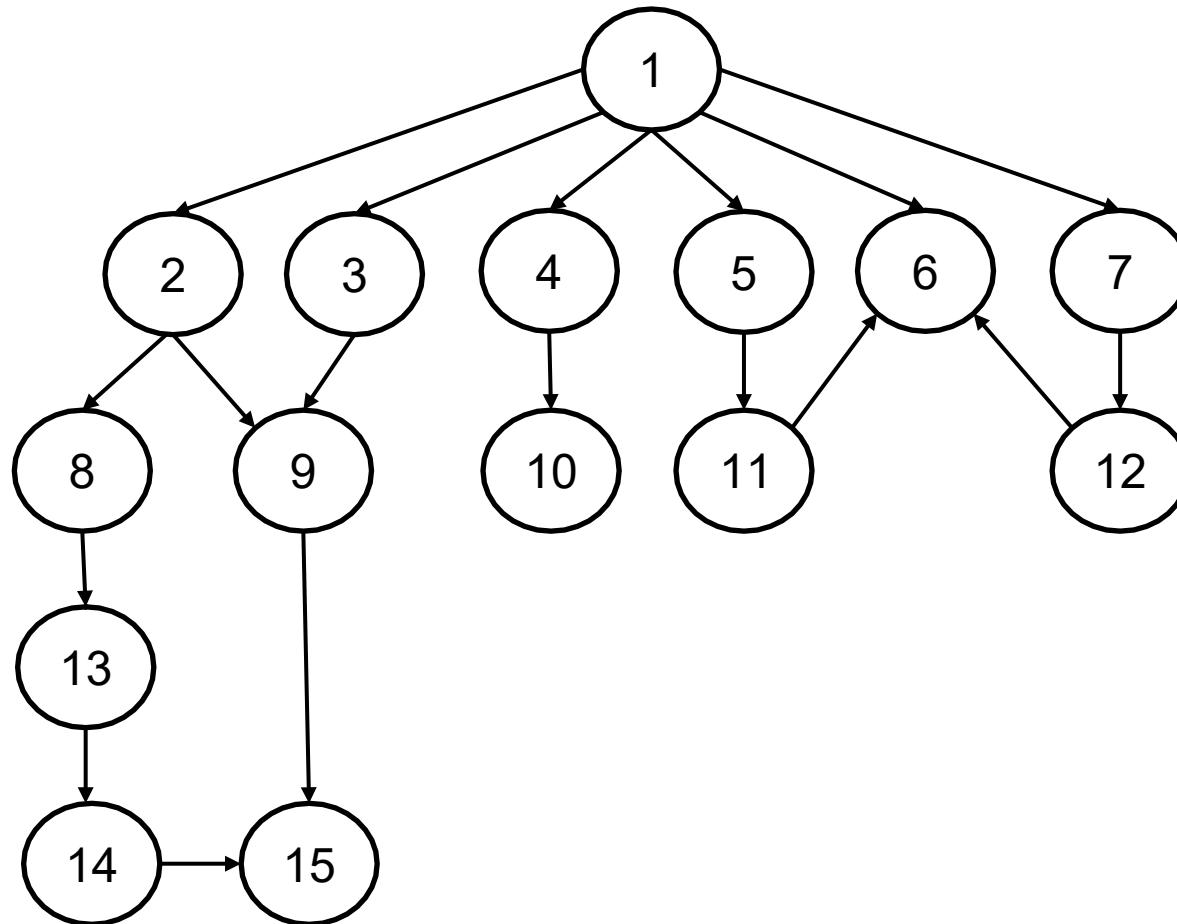
BITS Pilani

Prashant Joshi



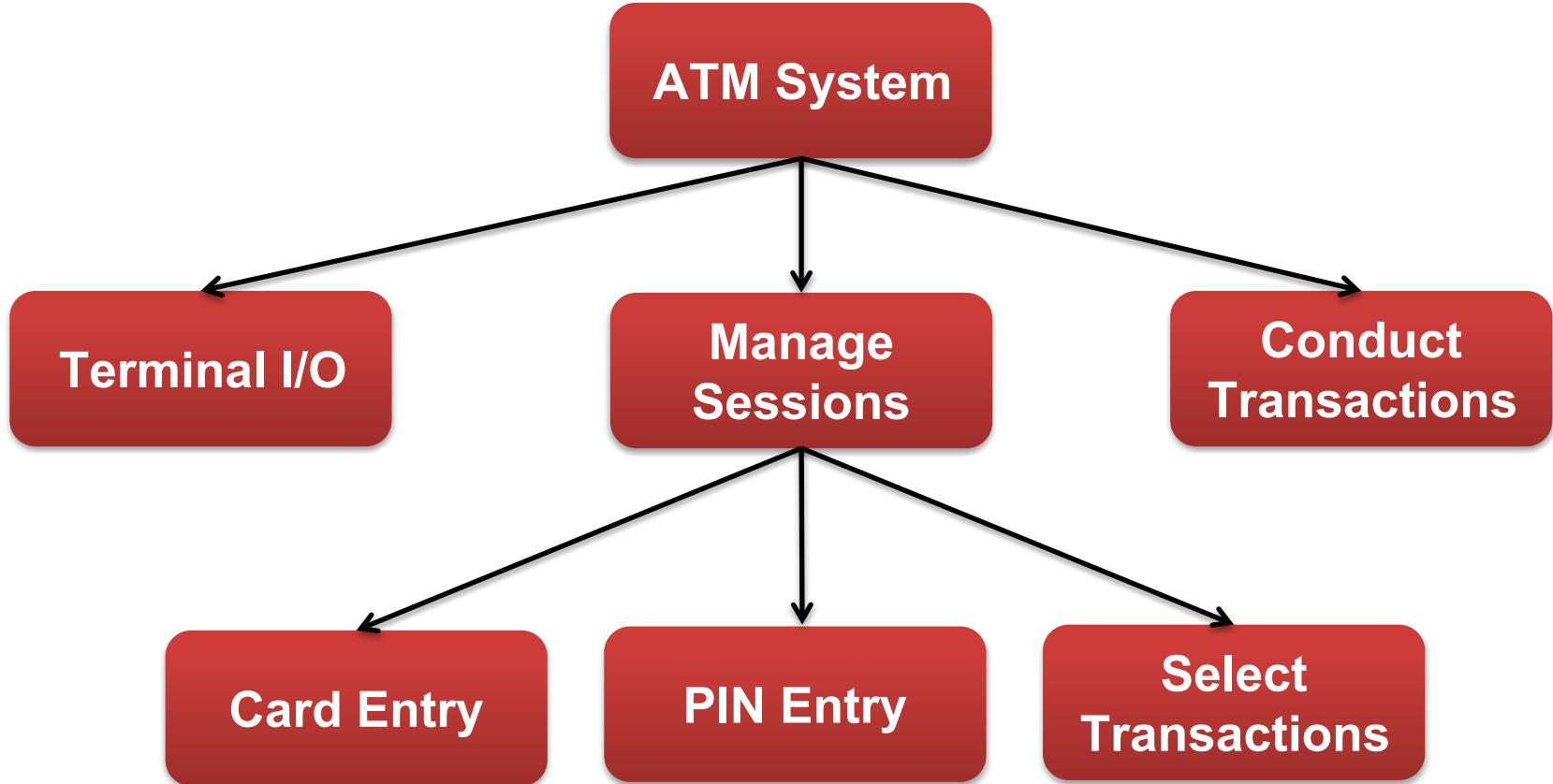
Topic 11.3 : Examples

A Sample Program



Refer: Page 239 of T1

Simple ATM



Source: T1: Figure 12.2



Software Testing Methodologies

BITS Pilani

Prashant Joshi

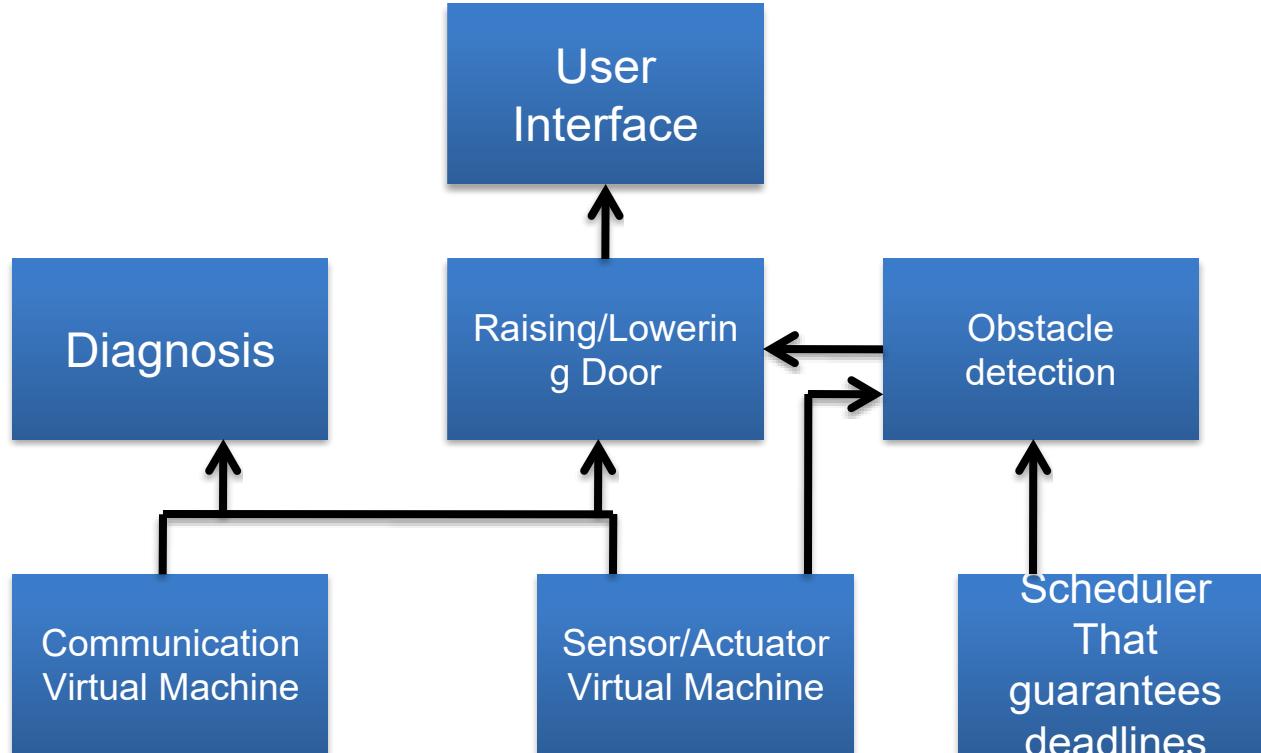


Topic 11.4 : Cases

Garage Door

- The device controls for opening and closing the door are different for the various product lines. They may include controls from within a home information system. The product architecture for a specific set of controls should be directly derivable from the product line architecture
- The processor used in different processes will differ. The product architecture for each specific processor should be directly derivable from the product line architecture
- If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) within 0.1 second
- The garage door opener should be accessible for diagnosis and administration from within the home information system using product specific diagnosis protocol. It should be possible to directly product an architecture that reflects this protocol

Garage Door

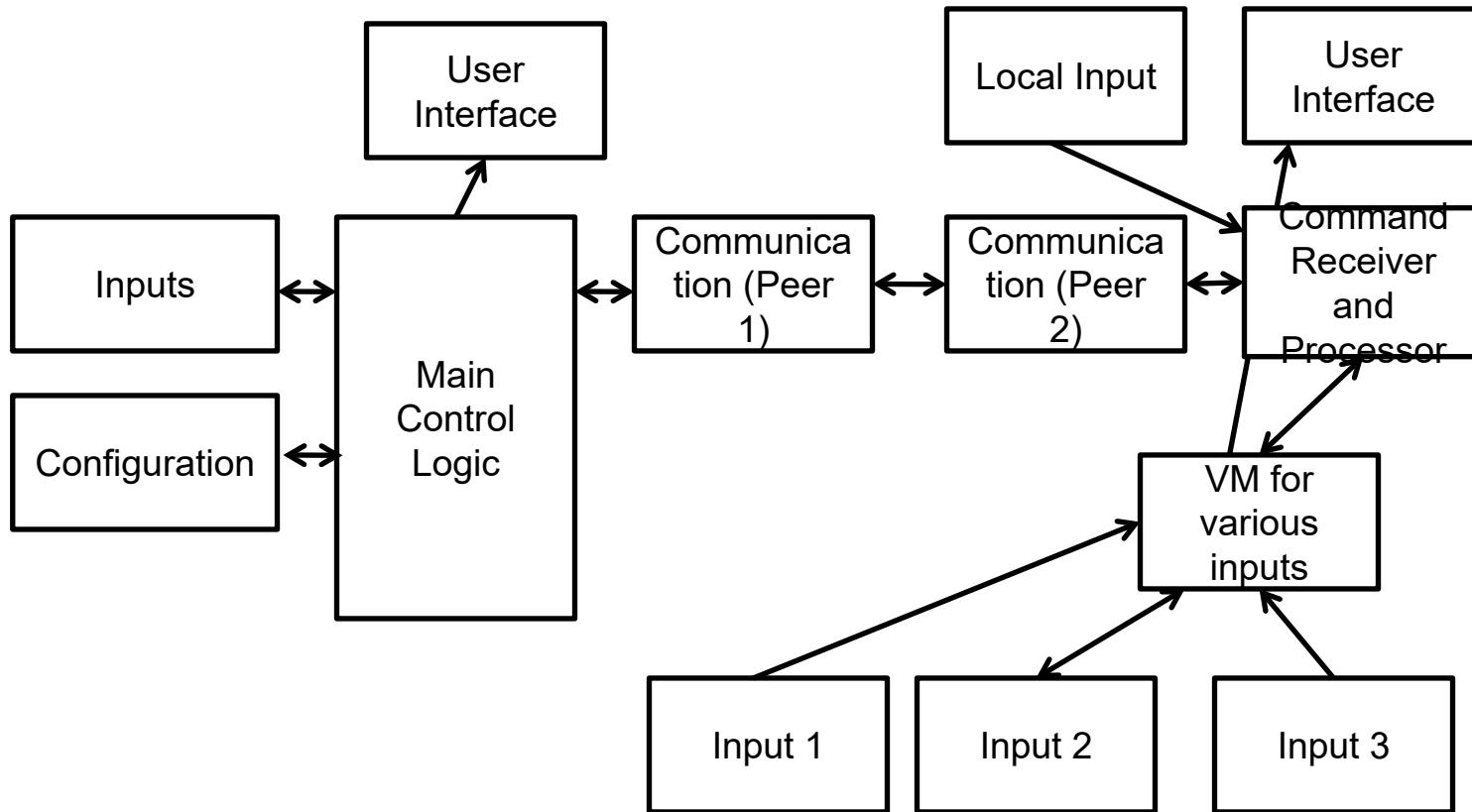


Building Lighting Control System



- Lighting control system for a apartment complex
- Emergency lights are UPS powered
- Corridor lights are timed and based on ambient light
- Garden and Architectural lights are timed

Building Lighting System





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 11 Self Study

- SS11.1 Take up a program (like currency converter) along with a GUI. Develop integration strategies and compare them or various attributes. Comment on use of various testing techniques for integration testing



SS 11.1 Integration Strategies

11.1 Self Study

To explore:

- Various strategies of integration and their effectiveness for testing software
- Use of testing techniques for integration testing

Study Work:

- You will need a modular program (either design and develop your self or use from open source) like currency converter
- Based on the modules; review and apply various integration strategies. Compare and contrast.
- Review various testing techniques to accomplish integration testing



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 9.2 Object Oriented UI frameworks study

9.2 Self Study

To explore:

- Application of testing techniques to OO software
- List the limitations that you come across

Study Work:

- Take up either GTK or QT as frameworks to study. Summarise their design. Explore the application of testing techniques to test the framework as well as applications developed using the framework
- Compare and contrast the effectiveness of the techniques to test the System



Software Testing Methodologies

BITS Pilani

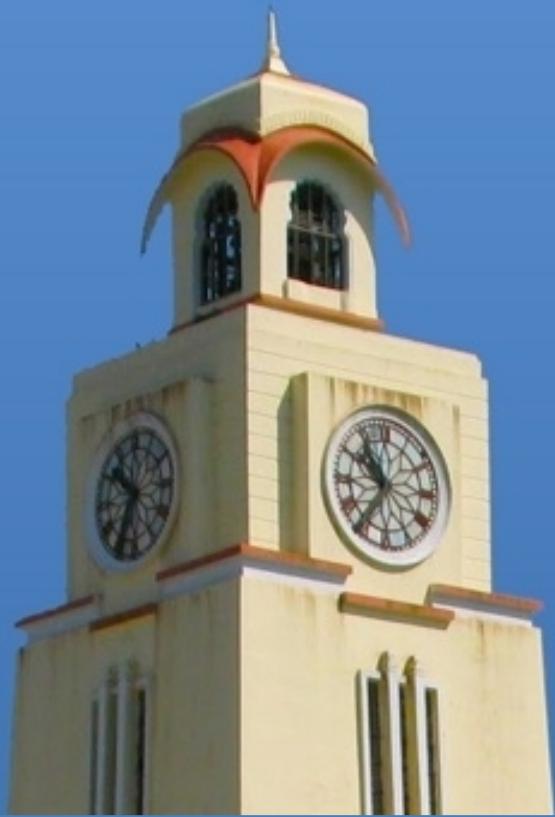
Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 8.1 Systems & Systems of Systems

8.1 Self Study

To explore:

- Systems and Systems of Systems around us

Study Work:

- Review systems and system of systems around us
- Write down their characteristics and working (in terms of use cases)
- Further, analyse use of techniques learnt so far for designing test cases
- Document your findings in form of a whitepaper (IEEE format recommended)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

System Testing

- System testing is concerned with the behaviour of a whole system. The majority of the functional failures should already have been identified during unit and integration testing
- System testing is usually considered appropriate for comparing the system to the non-functional system requirements, such as security, speed, accuracy, and reliability. External interfaces to other applications, utilities, hardware devices, or the operating environment are also evaluated at this level.

System Testing

- It is essential to define what the “system” is made up of
 - A system for one team/organisation may mean a module/part of a larger system
- Generally system level is treated as closest to everyday experience i.e. system (Product) as seen in the hands of the user
- In system testing apart from finding faults the goal is to demonstrate
 - System works
 - System is reliable
 - System is durable
 - Quality of system for defined parameters
- Use of specification based testing techniques to design test cases

System Testing – Aspects

- Thread
 - Thread Possibilities
 - Thread Definitions
 - Atomic Thread Function

System Requirements – Basic Concepts



- **Data**
 - Information used and created by the system
- **Actions**
 - Transform, data transform, control transform, process, activity, task, method, service
- **Devices**
 - Source and destination of system level inputs and outputs
- **Events**
 - System level I/O; occurs on port device; inputs and outputs of actions
- **Threads**
 - Model of a system – interactions among data, events and action

Concepts

- Relationship among basic concepts
 - Interrelation among data, actions, devices, events and threads
- Modeling with Basic Concepts
 - Structural
 - Behavioural
 - Contextual

Structural Strategies for Thread Testing



- Bottom Up Threads
- Node and Edge Coverage Metrics

Functional Strategies for Thread Testing



- Event-based Thread Testing
- Port-based Thread Testing
- Data-based Thread Testing

Testing – In action

- Process View
- Progression Vs. Regression

Test Plan

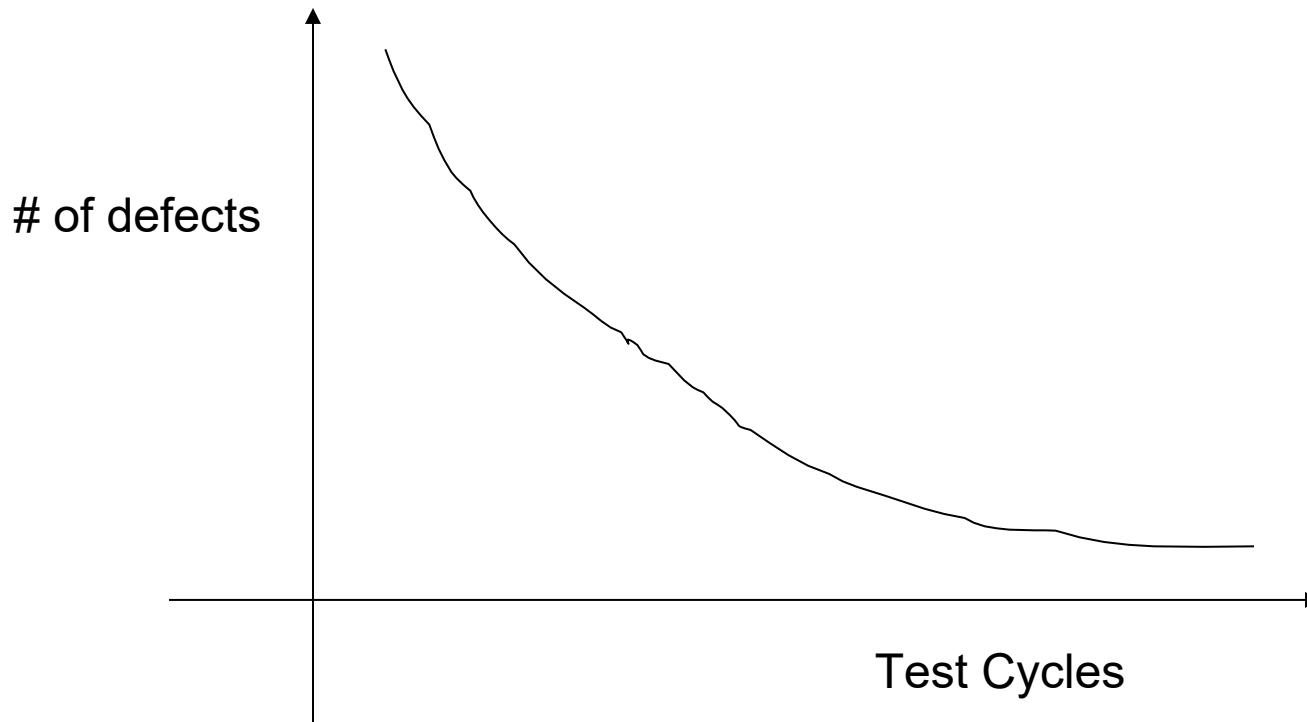
- IEEE Test Plan (Review the paper)
- Test Cycles
- Test Cases per cycle
- Test Execution Report
- Test Automation
- Test Effort Estimation
 - Design
 - Development
 - Execution

Reliability Analysis

- Reliability theory is concerned with determining the probability, that a system, possibly consisting of many components will function.
- Overall, System components can be treated in configurations
 - Series System: System will function if all components are functioning
 - Parallel System: System will function if one of the component is functioning. (Provided the ultimate aim is to have the same component working).
 - Composite System: (Series + Parallel) in varying configurations

Reliability Analysis

- This study gives us the mean life time of a system
- Analysis requires us to look at the system when failed components are subjected to repair





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 12: Agenda

Module 12: System Testing

Topic 12.1

System Testing- Introduction,
Overview & Issues

Topic 12.2

System Testing – Types, Techniques &
Strategies

Topic 12.3

Examples



Topic 12.1: System Testing- Introduction, Overview & Issues

System Testing

- System testing is concerned with the behaviour of a whole system. The majority of the functional failures should already have been identified during unit and integration testing
- System testing is usually considered appropriate for comparing the system to the non-functional system requirements, such as security, speed, accuracy, and reliability. External interfaces to other applications, utilities, hardware devices, or the operating environment are also evaluated at this level.

System Testing

- It is essential to define what the “system” is made up of
 - A system for one team/organisation may mean a module/part of a larger system
- Generally system level is treated as closest to everyday experience i.e. system (Product) as seen in the hands of the user
- In system testing apart from finding faults the goal is to demonstrate
 - System works
 - System is reliable
 - System is durable
 - Quality of system for defined parameters
- Use of specification based testing techniques to design test cases

System Testing – Aspects

- Thread
 - Thread Possibilities
 - Thread Definitions
 - Atomic Thread Function

System Requirements – Basic Concepts



- Data
 - Information used and created by the system
- Actions
 - Transform, data transform, control transform, process, activity, task, method, service
- Devices
 - Source and destination of system level inputs and outputs
- Events
 - System level I/O; occurs on port device; inputs and outputs of actions
- Threads
 - Model of a system – interactions among data, events and action



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 12.2: System Testing – Types, Techniques & Strategies

Concepts

- Relationship among basic concepts
 - Interrelation among data, actions, devices, events and threads
- Modeling with Basic Concepts
 - Structural
 - Behavioural
 - Contextual

Structural Strategies for Thread Testing



- Bottom Up Threads
- Node and Edge Coverage Metrics

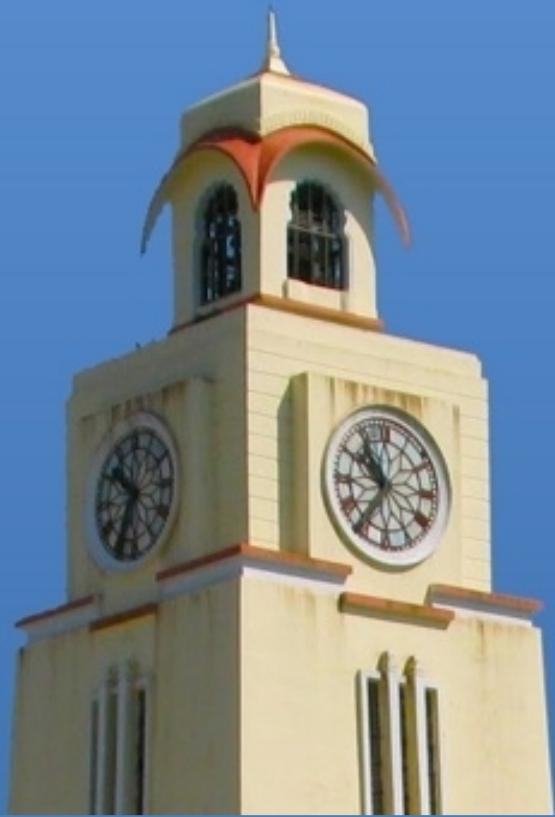
Functional Strategies for Thread Testing



- Event-based Thread Testing
- Port-based Thread Testing
- Data-based Thread Testing

Testing – In action

- Process View
- Progression Vs. Regression



Test Plan

Test Plan

- IEEE Test Plan (Review the paper)
- Test Cycles
- Test Cases per cycle
- Test Execution Report
- Test Automation
- Test Effort Estimation
 - Design
 - Development
 - Execution

Reliability Analysis

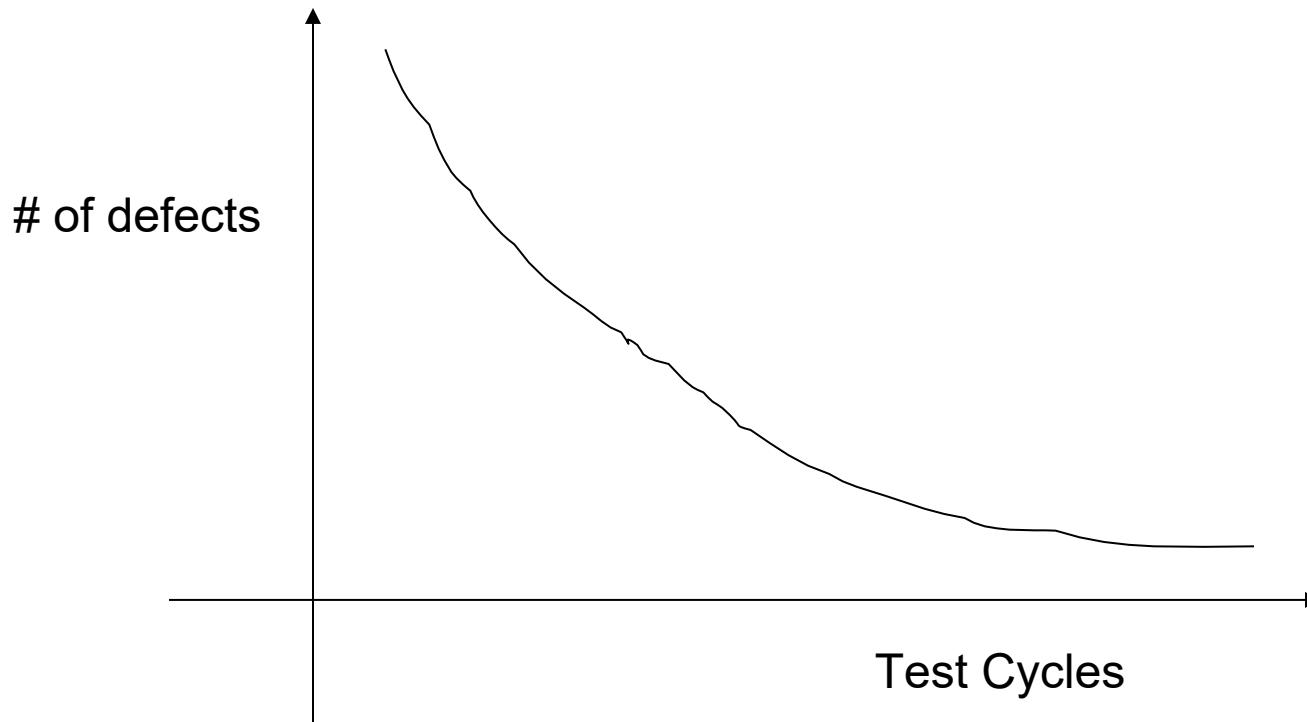
- Reliability theory is concerned with determining the probability, that a system, possibly consisting of many components will function.
- Overall, System components can be treated in configurations
 - Series System: System will function if all components are functioning
 - Parallel System: System will function if one of the component is functioning. (Provided the ultimate aim is to have the same component working).
 - Composite System: (Series + Parallel) in varying configurations

Reliability Analysis

- Each component will function with some known probability (independent of each other)
- Upper and lower bounds of probability
- A component may show a dynamic behaviour over time i.e. it functions for a random length of time and after which it fails
- Amount of time a system functions is related to the distribution of a component lifetimes
- In particular, it turns out that if the amount of time that a component functions has an increasing failure rate on average distribution, then so does the distribution of system lifetimes

Reliability Analysis

- This study gives us the mean life time of a system
- Analysis requires us to look at the system when failed components are subjected to repair



Reliability Analysis

- Mean Time to Failure (MTTF) = Total Testing time/# of defects
$$MTTF = T/d$$
- Failure Rate

$$F_a = \frac{1}{MTTF}$$

- Mean Time to Repair (MTTR) = Sum(|Time of detection of fault – time of closure|)/Total # of defects

$$MTTR = \sum (T_d - T_c)/d$$

- Availability

$$A = MTTF / (MTTR + MTTF)$$

Reliability Analysis

- Problem: A SW component was tested for a period of 3 months. The total number of defects found were 300. On an average the time required to fix the defect was 2.5 days. Find the availability of the system
- Solution:

$$MTTF = (3 * 30 * 8) / 300 = 2.4$$

$$F_a = 1 / 2.4 = 0.4166$$

$$MTTR = (2.5 * 8) = 20$$

$$\text{Availability } A = MTTF / (MTTF + MTTR) = \\ 0.107 \sim 10.7\%$$

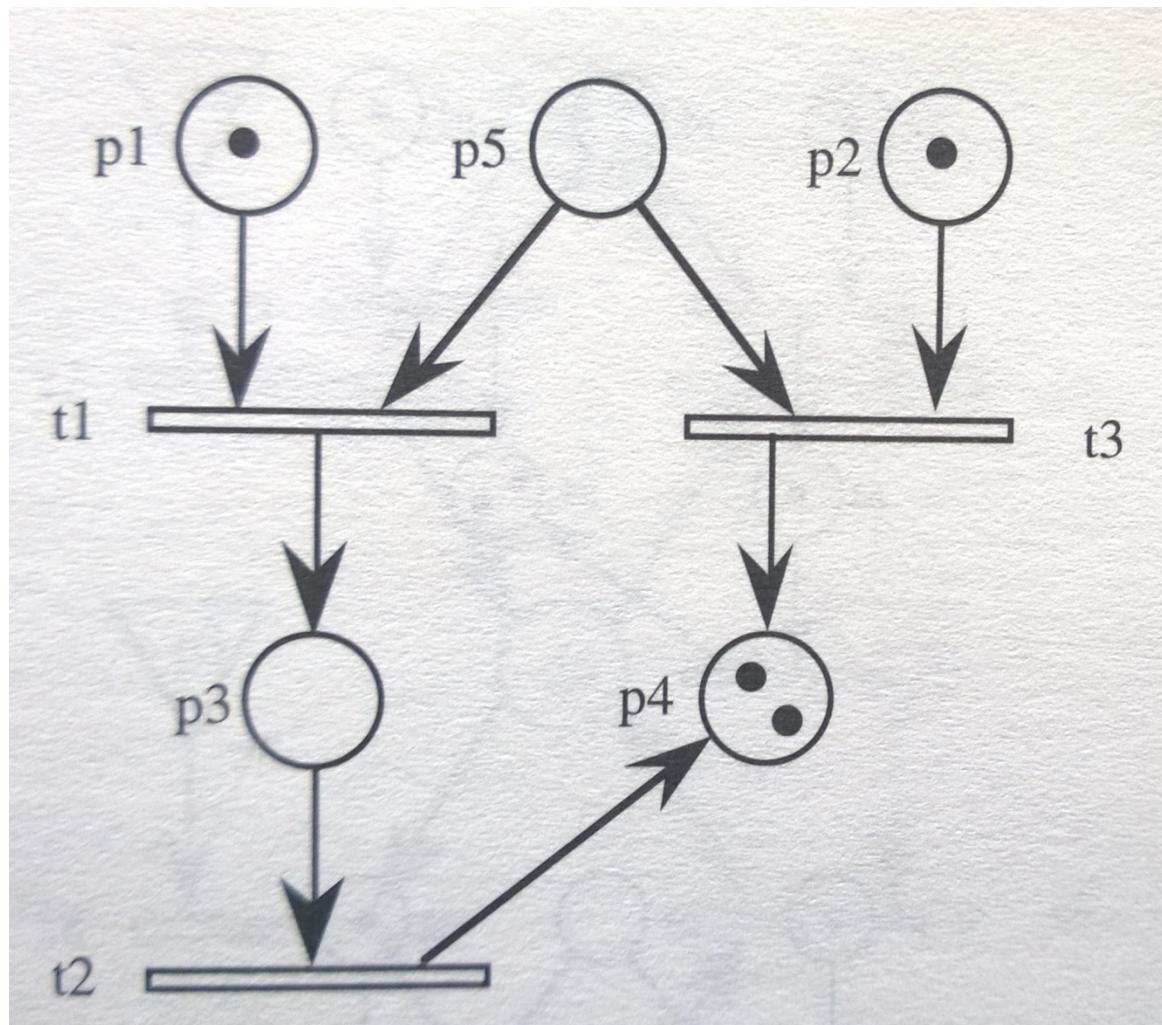


Petri Nets & System Test

Petri Net

- Carl Adam Petri 1963
- Accepted Model for protocol and other application involving concurrency and distributed processing
- Form of directed graph: a bipartite directed graph
- A bipartite graph has two sets of nodes V_1 and V_2 and a set of edges E , with the restriction that every edge has its initial node on one of the sets V_1 , V_2 and its terminal node in the other set
- In Petri net one set is Places and the other is Transitions

A Marked Petri Net



Event Driven Petri Net

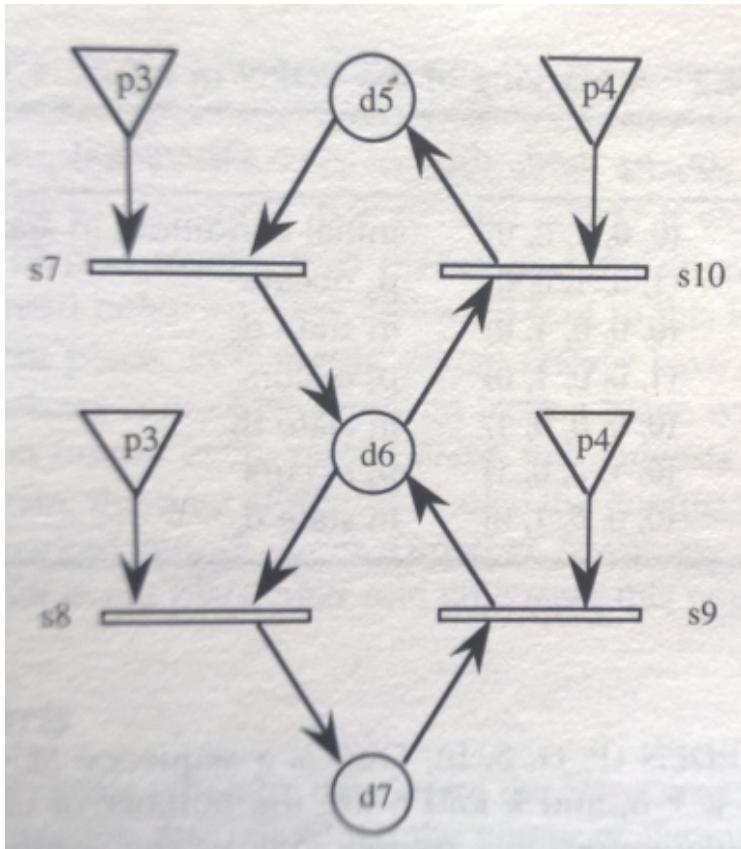
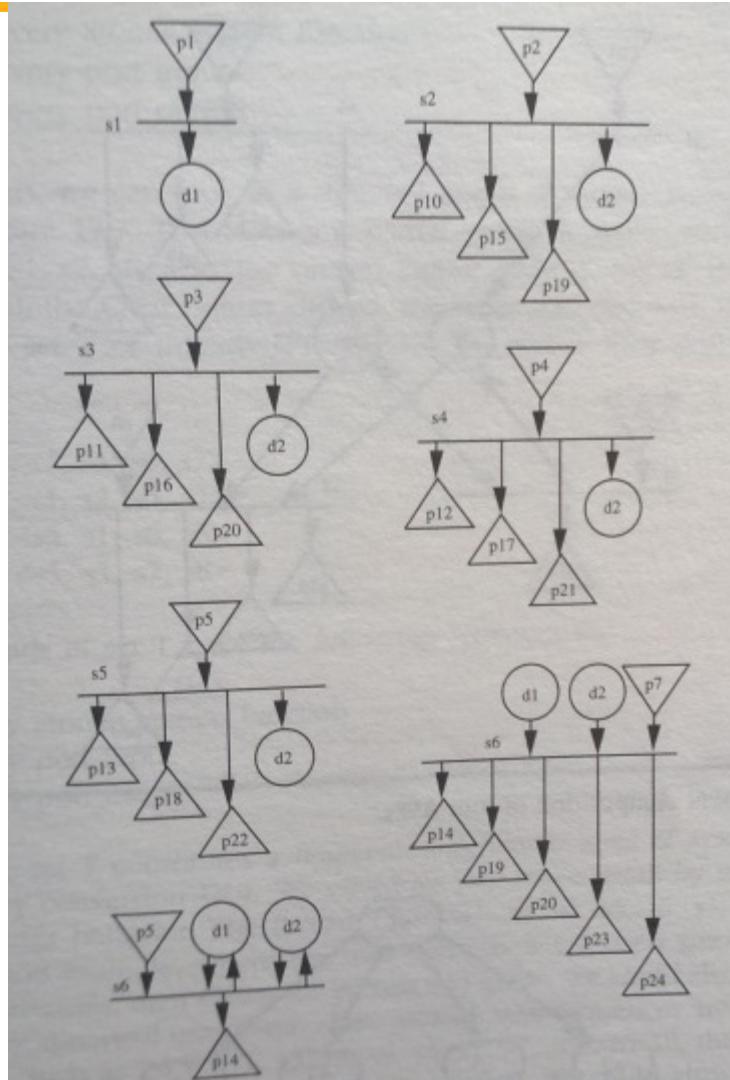


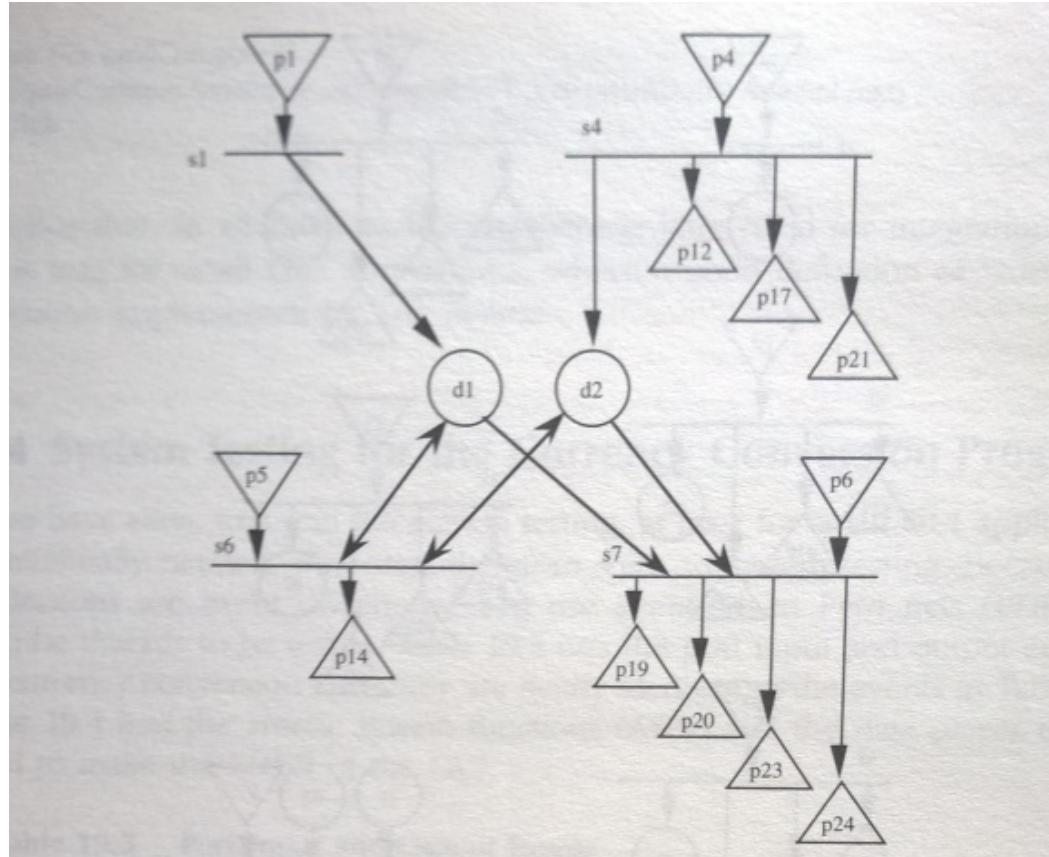
Table 4.1 EDPN Elements in Figure 4.10

Element	Type	Description
p_3	port input event	rotate dial clockwise
p_4	port input event	rotate dial counterclockwise
d_5	data place	dial at position 1
d_6	data place	dial at position 2
d_7	data place	dial at position 3
s_7	transition	state transition: d_5 to d_6
s_8	transition	state transition: d_6 to d_7
s_9	transition	state transition: d_7 to d_6
s_{10}	transition	state transition: d_6 to d_5

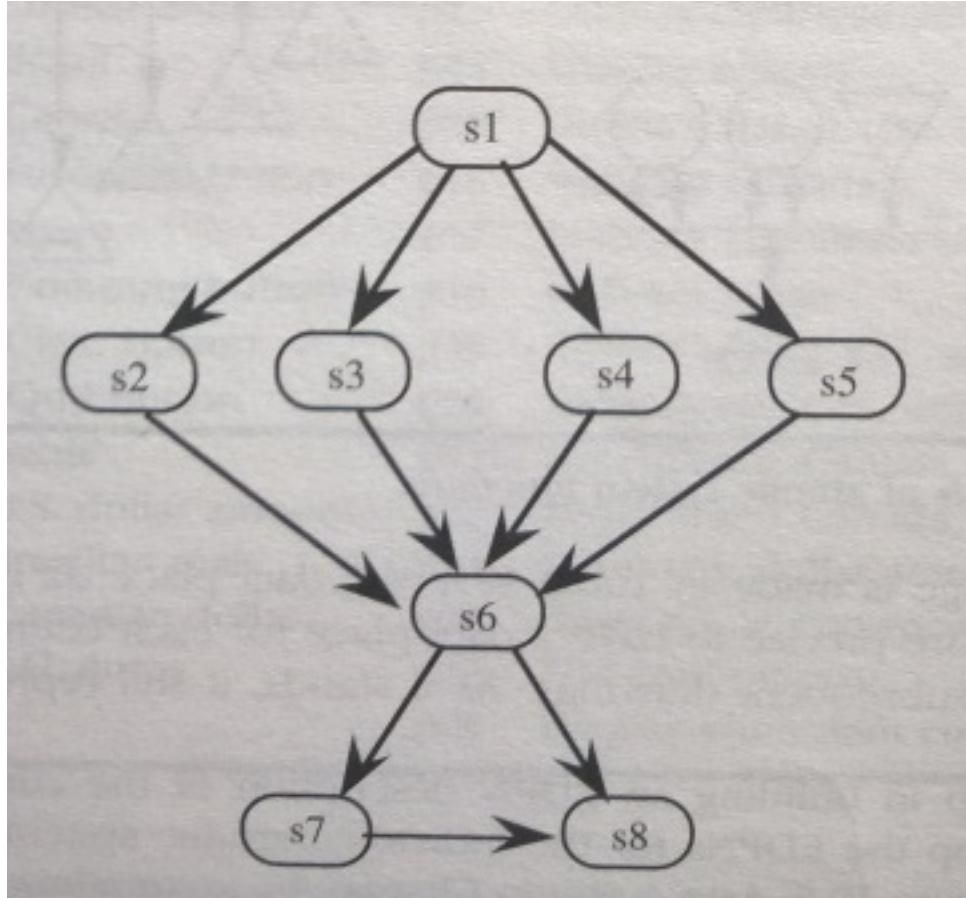
System Testing Currency Converter Program



EPDN Composition of Four ASFs



Directed Graph of ASF sequences





Software Testing Methodologies

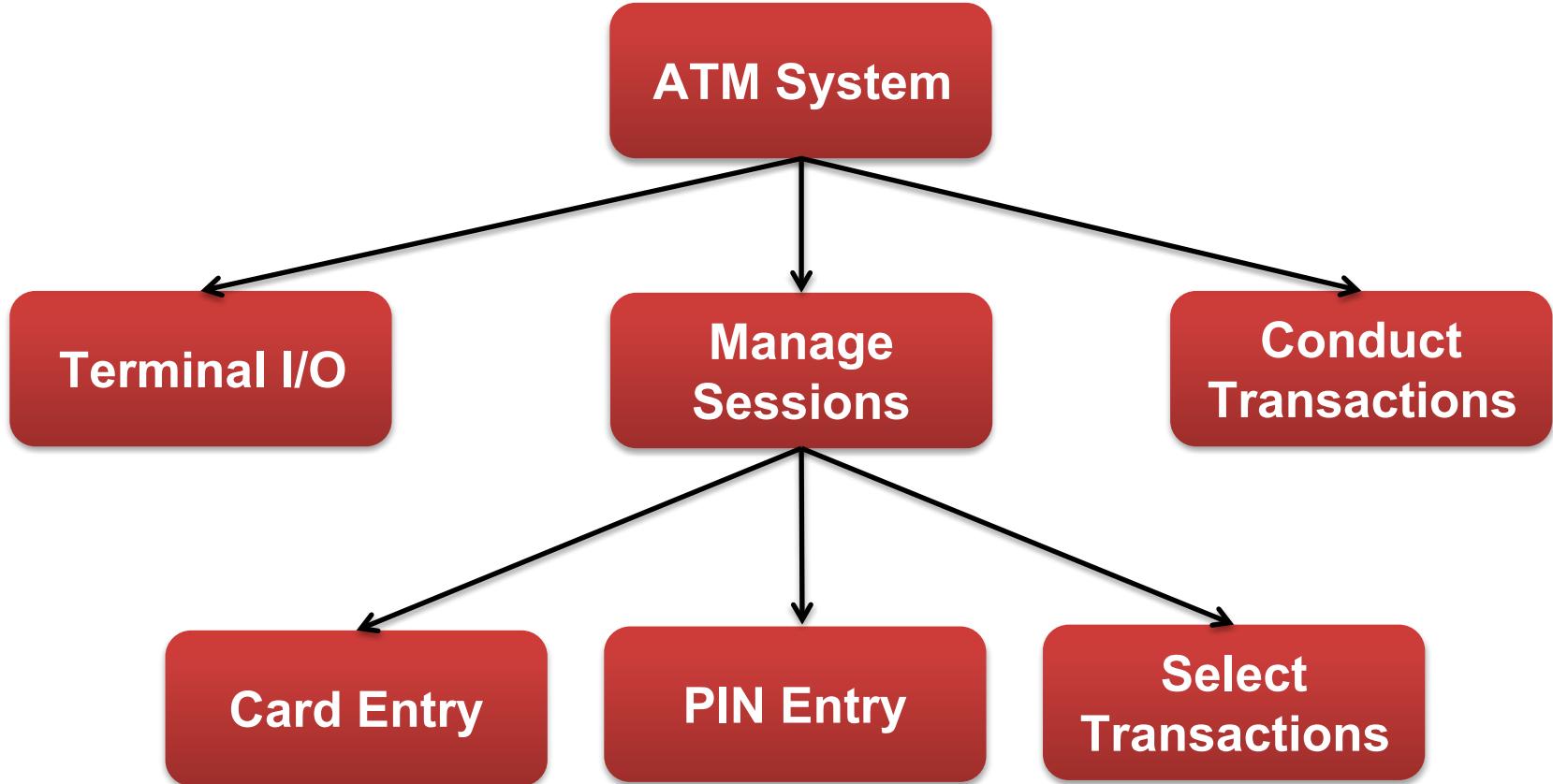
BITS Pilani

Prashant Joshi



Topic 12.3: Examples

Automated Teller Machine

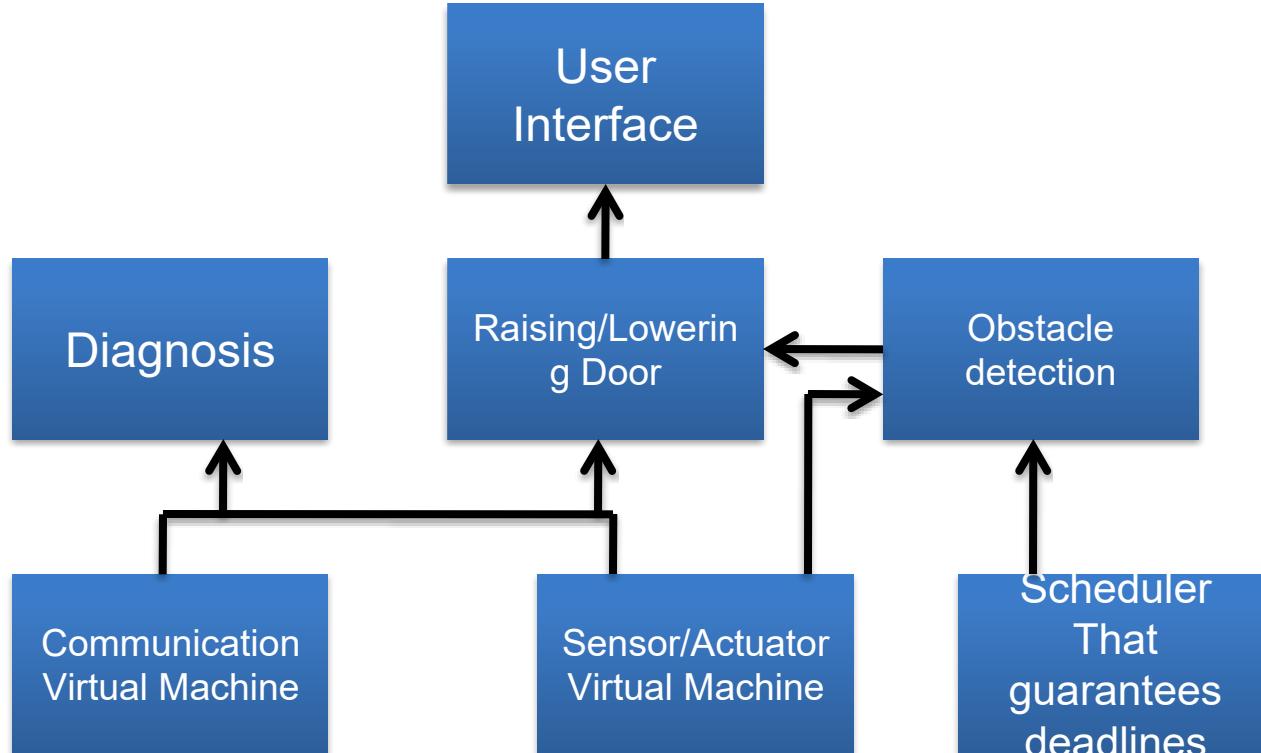


Source: T1: Figure 12.2

Garage Door

- The device controls for opening and closing the door are different for the various product lines. They may include controls from within a home information system. The product architecture for a specific set of controls should be directly derivable from the product line architecture
- The processor used in different processes will differ. The product architecture for each specific processor should be directly derivable from the product line architecture
- If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) within 0.1 second
- The garage door opener should be accessible for diagnosis and administration from within the home information system using product specific diagnosis protocol. It should be possible to directly product an architecture that reflects this protocol

Garage Door

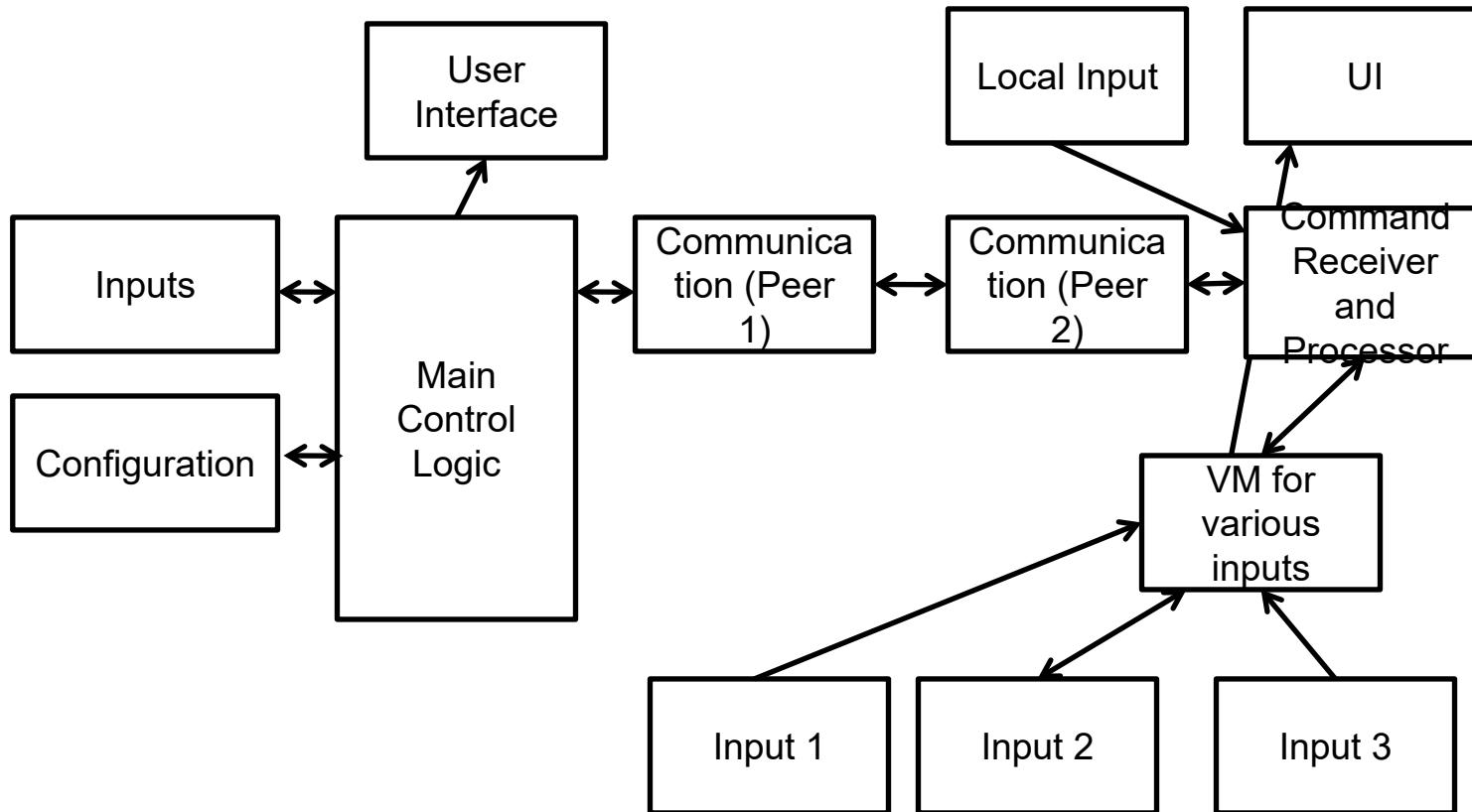


Building Lighting Control System



- Lighting control system for a apartment complex
- Emergency lights are UPS powered
- Corridor lights are timed and based on ambient light
- Garden and Architectural lights are timed

Building Lighting System





BITS Pilani

Software Testing Methodologies

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Life-Cycle & Model View

innovate

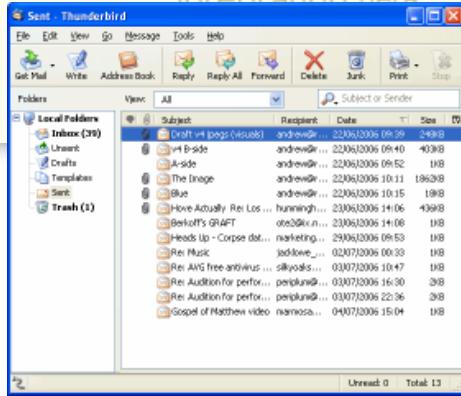
achieve

lead

- Software Development Life Cycles
- Model Based Testing
- Testing at all levels

Usability

Product
SubSystem
Alpha Unit
Component
Module
Scalability



Building Blocks – A View

SW Products
SW Test Tools

Test Research

Test Techniques Application

Functional, Behavioral, IOT, Usability, Integration, Unit, Performance, Stress

Test Techniques Development

EC, BVA, DT, CEG, McCabe, OATS, Data Flow, Control Flow

Math

Set theory
Discrete
Combinatorial

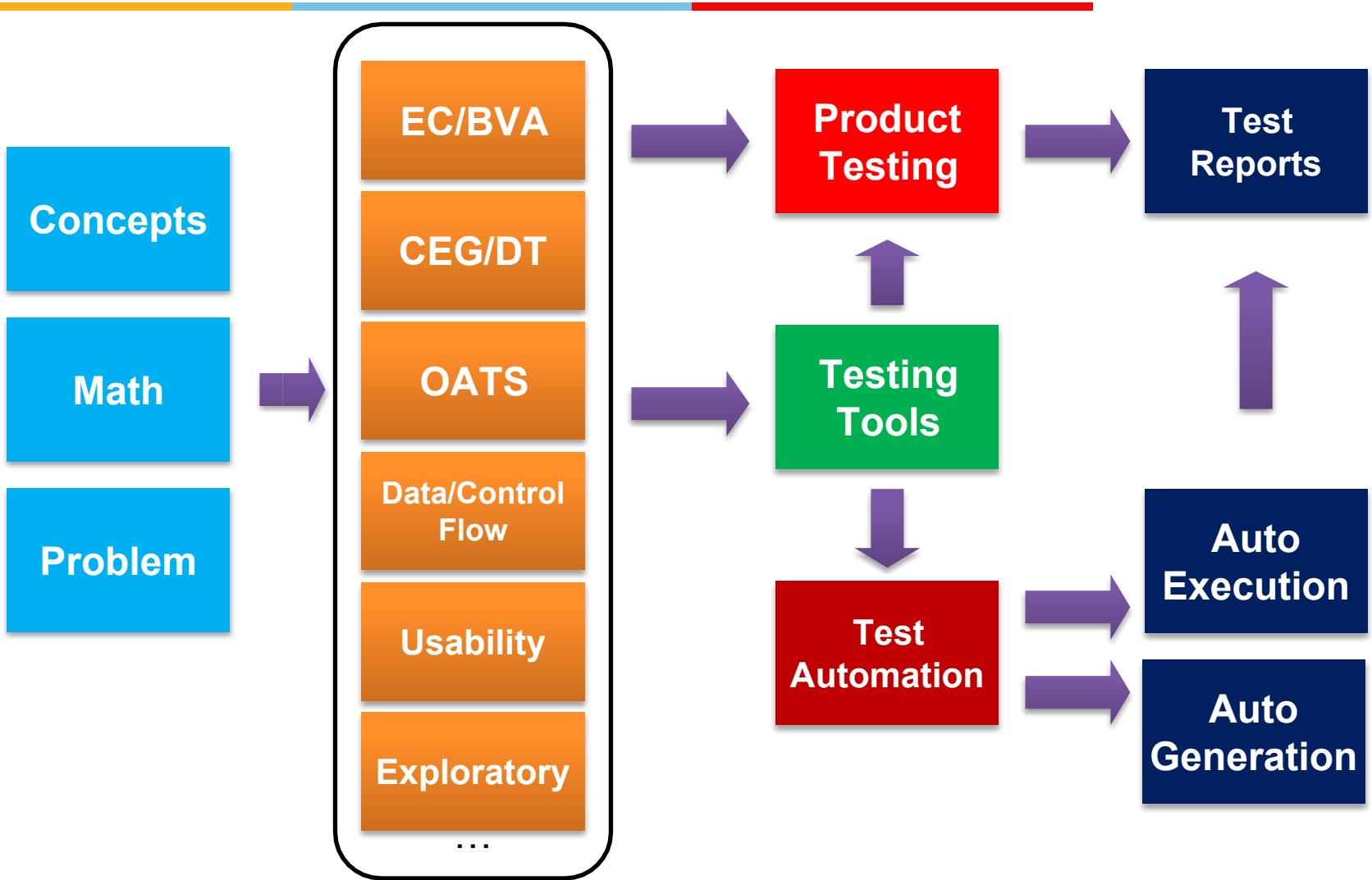
Concepts

SW Arch, Design, Data Structures

Problem

Code coverage
Quality
Zero Defects

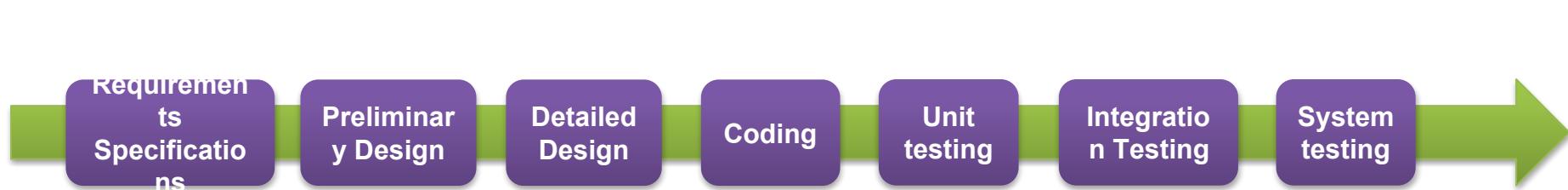
Progression



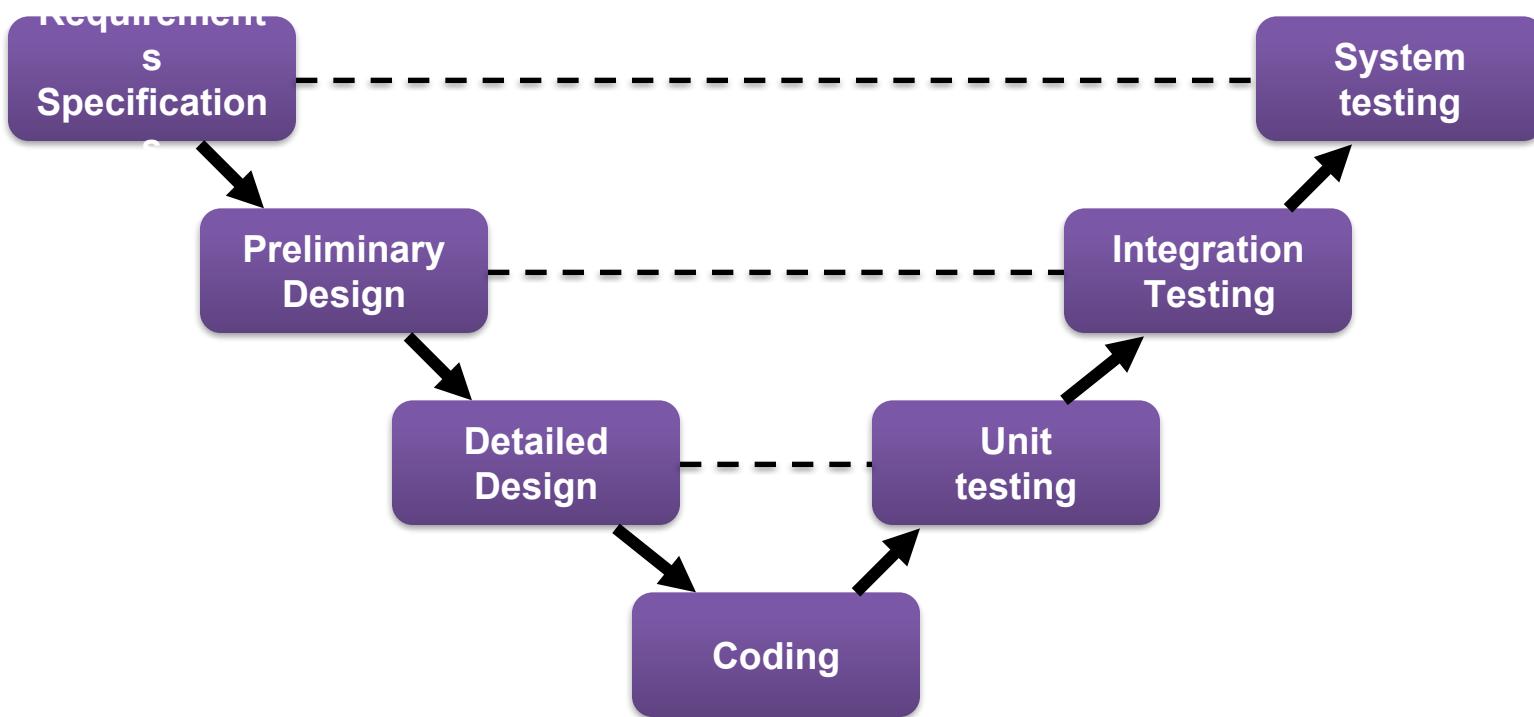
Test Process – Test Activities

- Test Planning
- Test case generation
- Test environment development
- Test Execution
- Test results evaluation
- Problem reporting/Test log
- Defect tracking

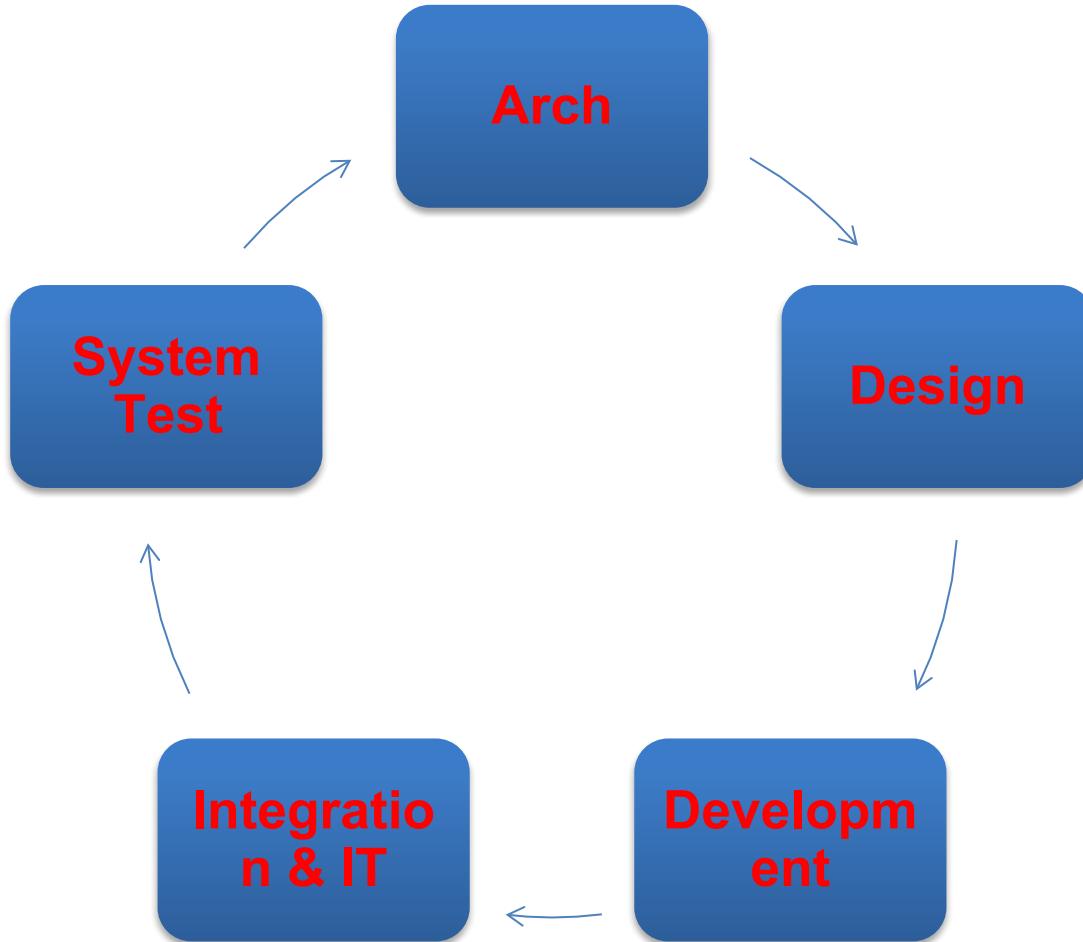
The Waterfall



The Waterfall

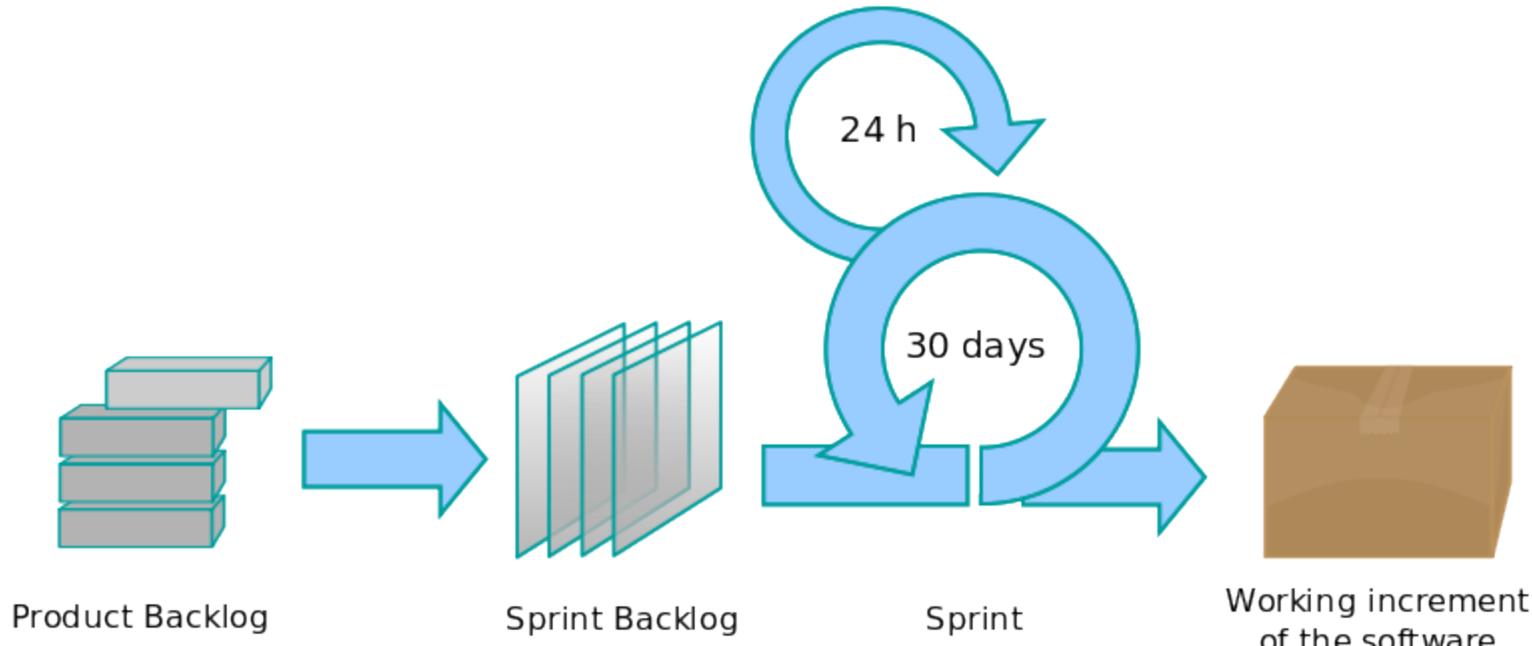


Iterative



- A View
- A Debate

Scrum



Reference: SCRUM Development Process by Ken Scheweber



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 13: Agenda

Topic 13.0

Life Cycle – Perspectives & Focus

Topic 13.1

Life-Cycle Based Testing – Overview & Perspective

Topic 13.2

Life-Cycles – Waterfall, Iterative & Agile

Topic 13.3

Implications and Issues, Strategies & models

Topic 13.4

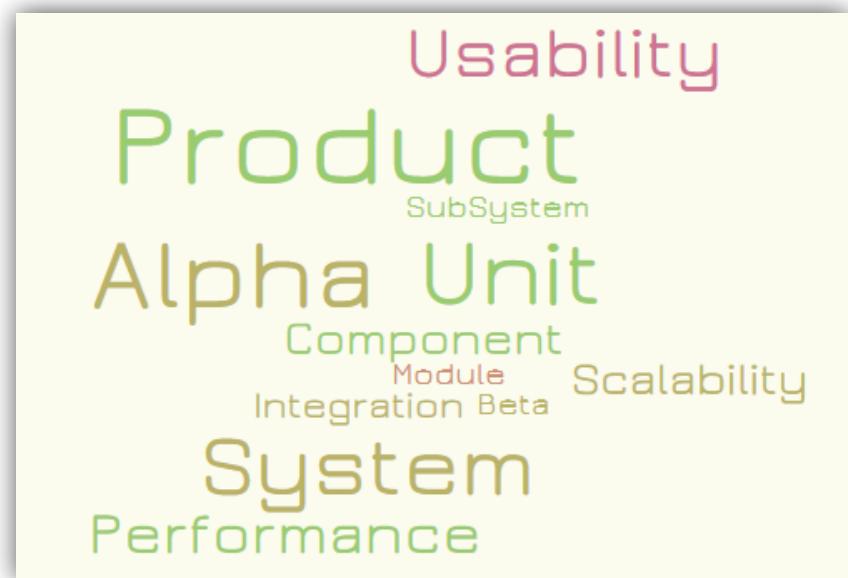
Example and Case



Topic 13.0 Perspectives & Focus

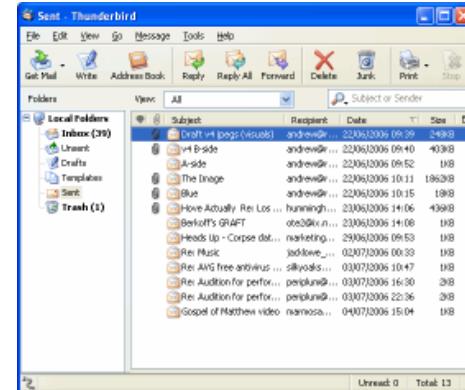
Life-Cycle & Model View

- Software Development Life Cycles
- Model Based Testing
- Testing at all levels



Review & Remember

- Levels of Testing
 - Targets
 - Unit
 - Integration
 - System
 - Objectives
 - Acceptance
 - Installation
 - Conformance
 - Functional
 - Correctness
 - Regression
 - Performance
 - Stress
 - ...



2004

Review & Remember

- Test Techniques (Our course, Our Focus)
 - Specification-based
 - Code-based
- Based on Nature of Application
 - OO testing
 - GUI
 - ...
- Selecting and Combining Techniques
 - Functional and structural
 - Deterministic vs. Random

Ref: SWEBOK 2004

Review & Remember

- Test Related Measures
 - Evaluation of Program under test
 - Evaluation of the Tests Performed
- Test Process
 - Practical Considerations
 - Test Activities

Ref: SWEBOK 2004



BITS Pilani

Software Testing Methodologies

Prashant Joshi



Topic 13.1: Life-Cycle Based Testing

– Overview & Perspective

Building Blocks – A View

SW Products
SW Test Tools

Test Research

Test Techniques Application

Functional, Behavioral, IOT, Usability, Integration, Unit, Performance, Stress

Test Techniques Development

EC, BVA, DT, CEG, McCabe, OATS, Data Flow, Control Flow

Math

Set theory
Discrete
Combinatorial

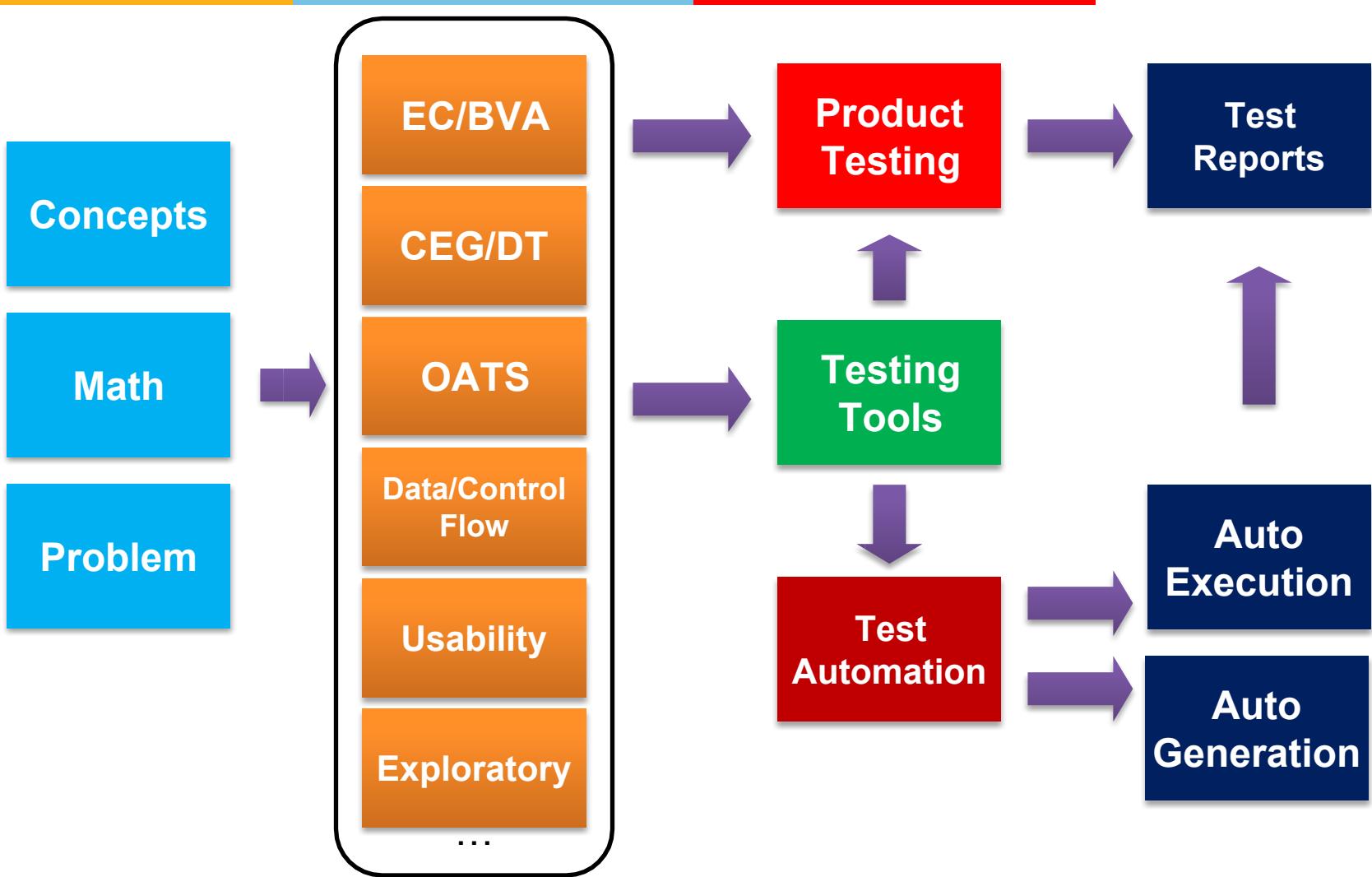
Concepts

SW Arch, Design, Data Structures

Problem

Code coverage
Quality
Zero Defects

Progression



Test Process – Test Activities

- Test Planning
- Test case generation
- Test environment development
- Test Execution
- Test results evaluation
- Problem reporting/Test log
- Defect tracking

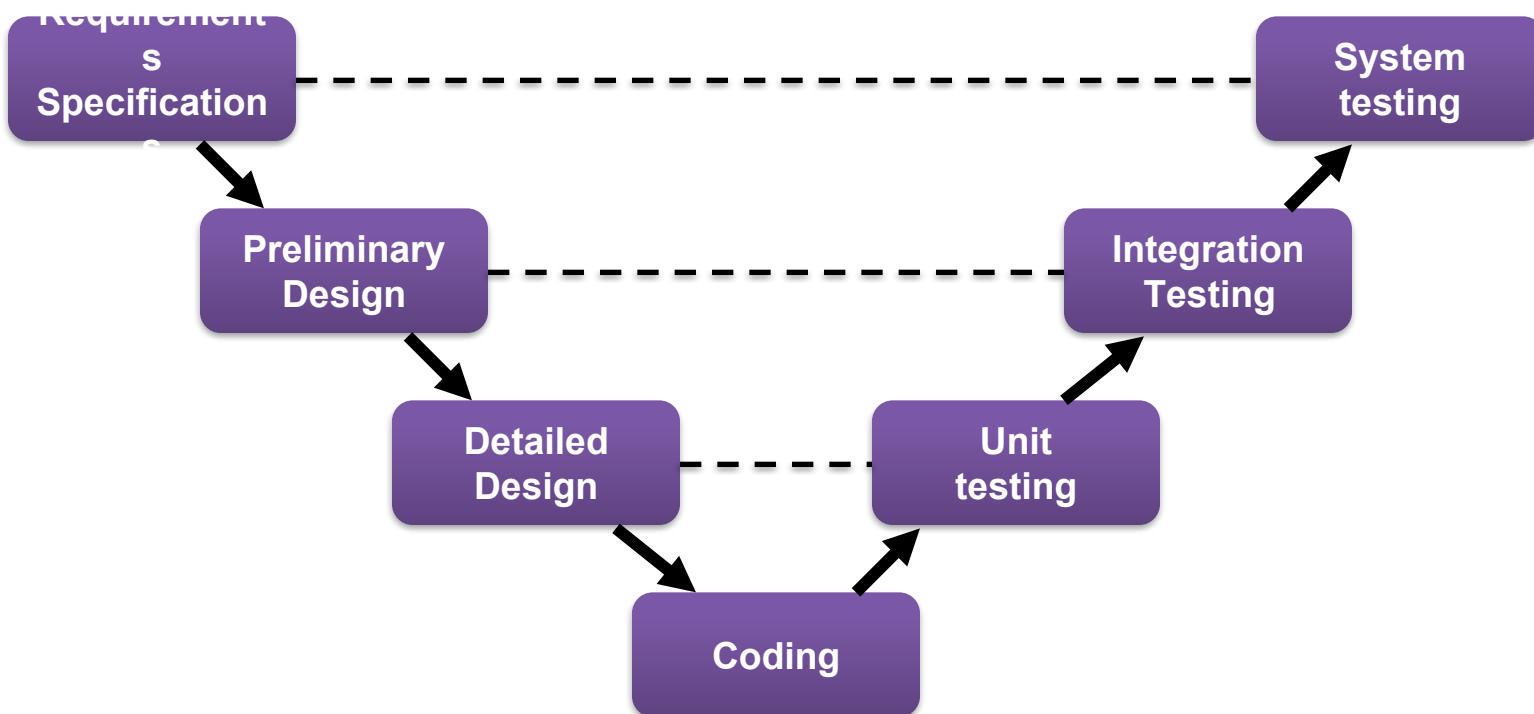
Software Development Life Cycles



Life Cycle Models

- Waterfall
- V-model
- Prototyping
- Waterfall – spin-offs
 - Evolutionary
 - Incremental Model
 - Spiral

The V Model



Please refer to the paper on the taxila

Focus Areas

- People
- Process
- Product
- Technology
- Research
- Business
- Innovation
- Engineering



Software Testing Methodologies

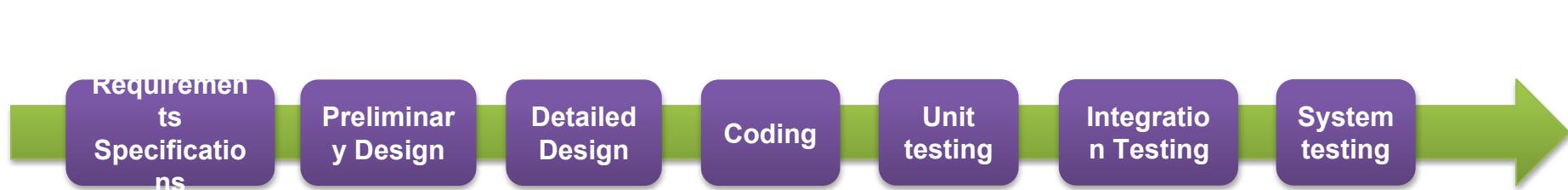
BITS Pilani

Prashant Joshi

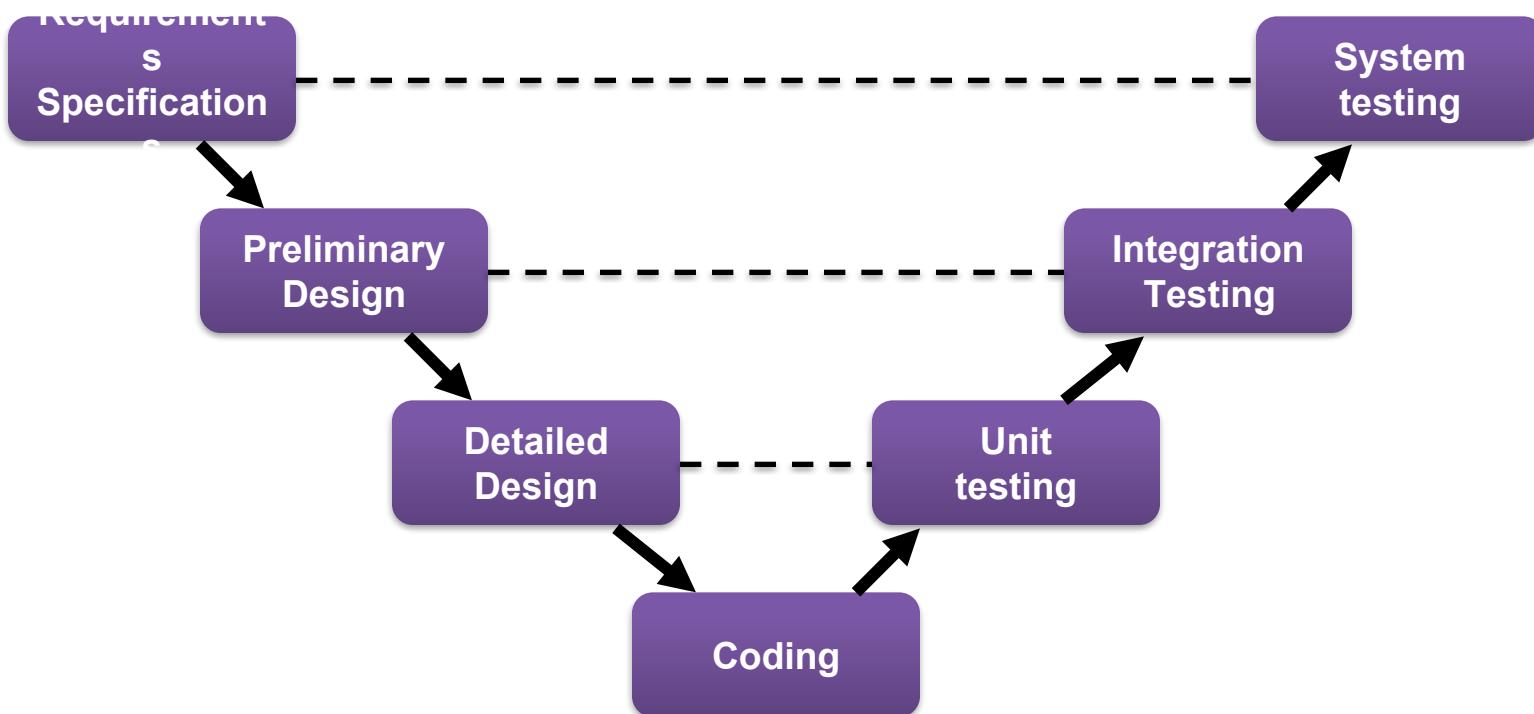


Topic 13.2 : Life-Cycles – Waterfall, Iterative & Agile

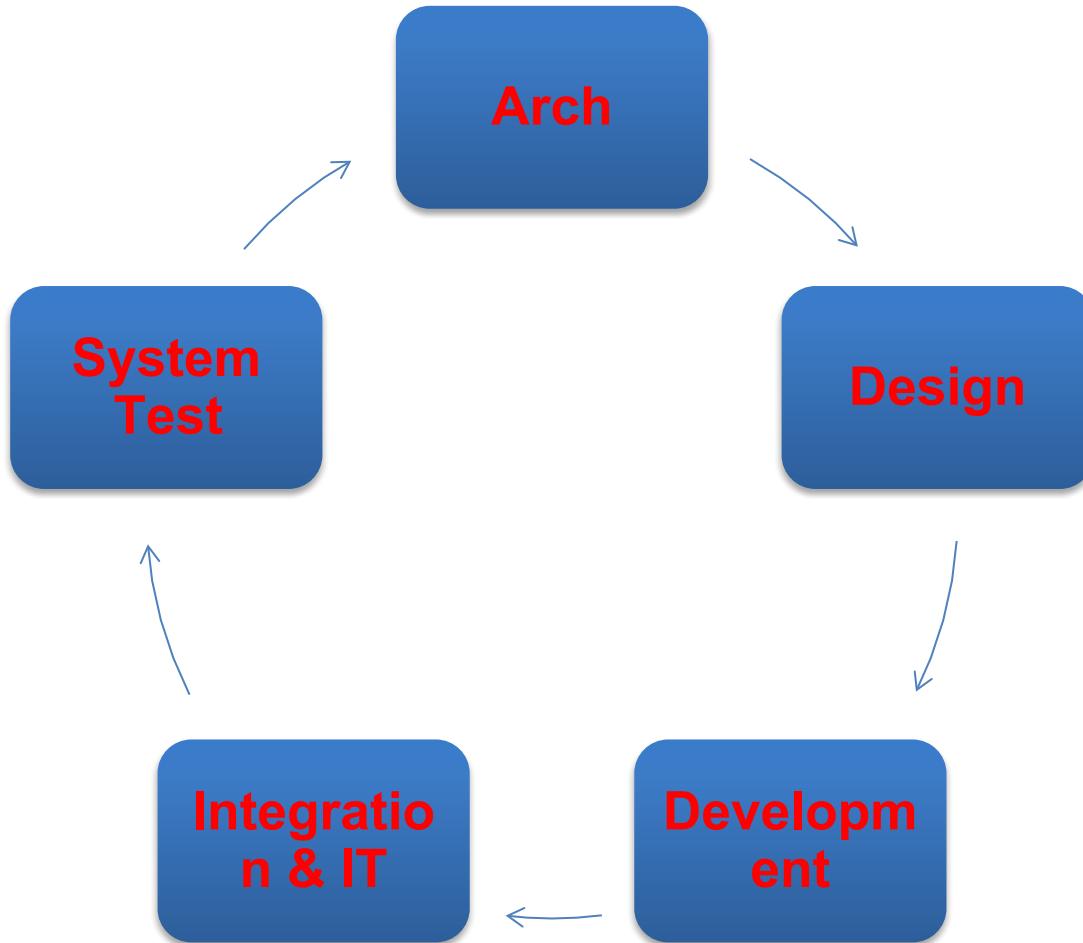
The Waterfall



The Waterfall



Iterative



- A View
- A Debate

AGILE Manifesto



We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Source: <http://agilemanifesto.org/>

Principles Behind Agile Manifesto

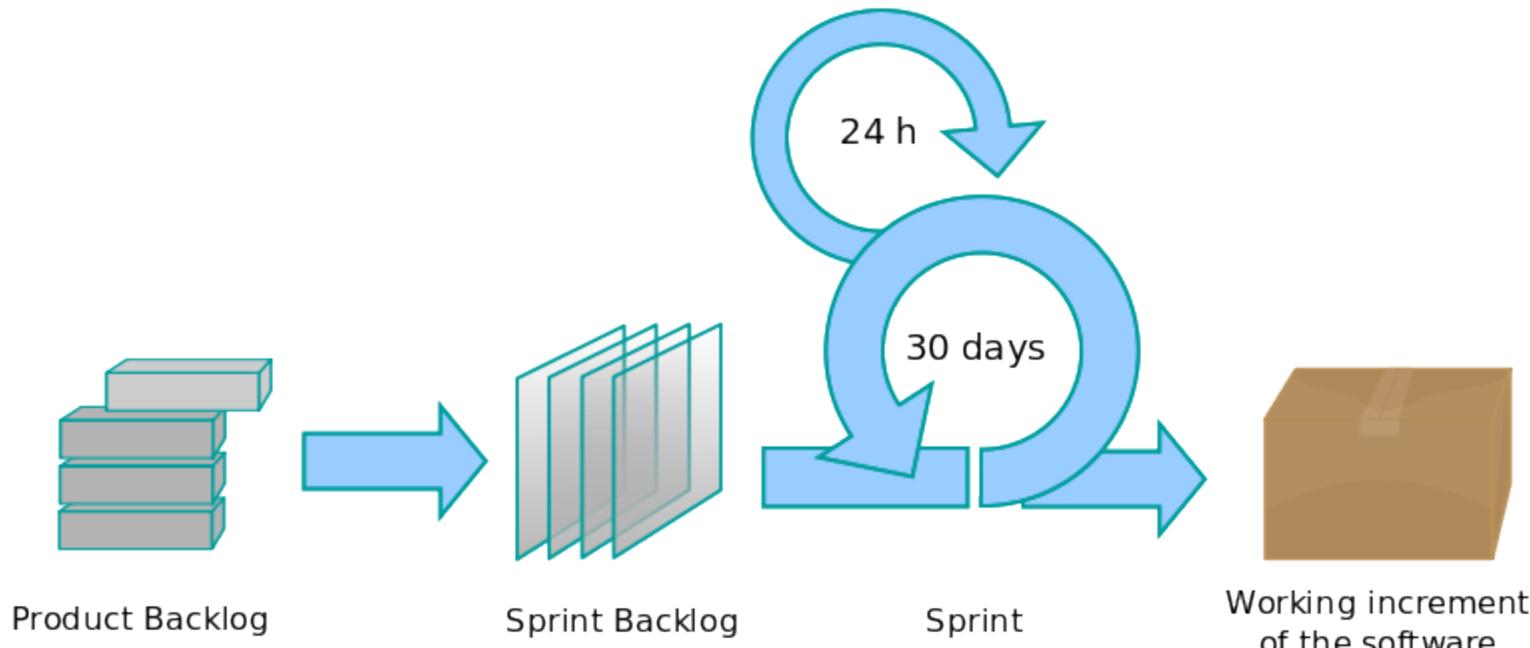


We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Source: <http://agilemanifesto.org/>

Scrum



Reference: SCRUM Development Process by Ken Scheweber



Software Testing Methodologies

BITS Pilani

Prashant Joshi



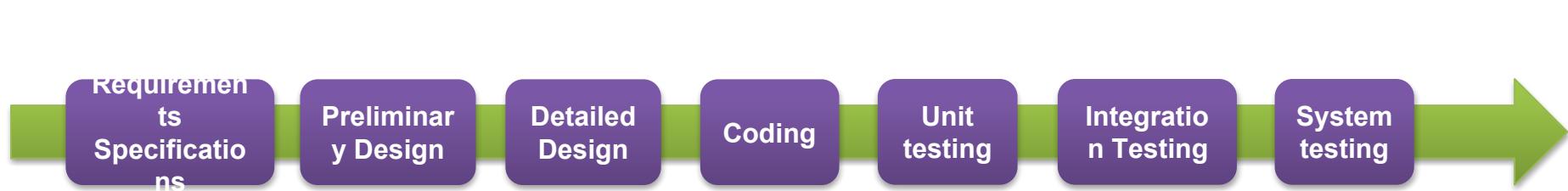
Topic 13.3 : Implications and Issues, Strategies & models

Issues & Implications with SDLCs

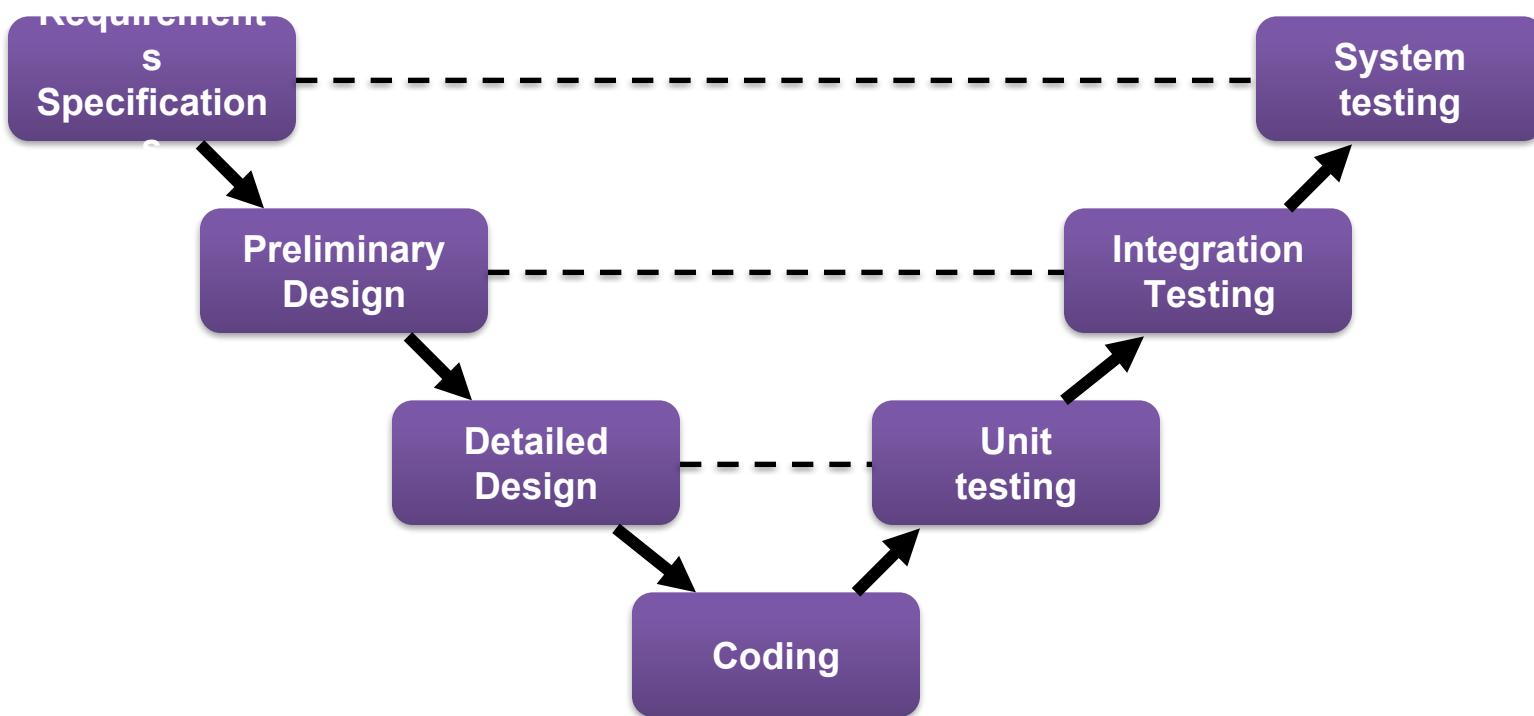


- Waterfall
- V Model
- Iterative
- Agile

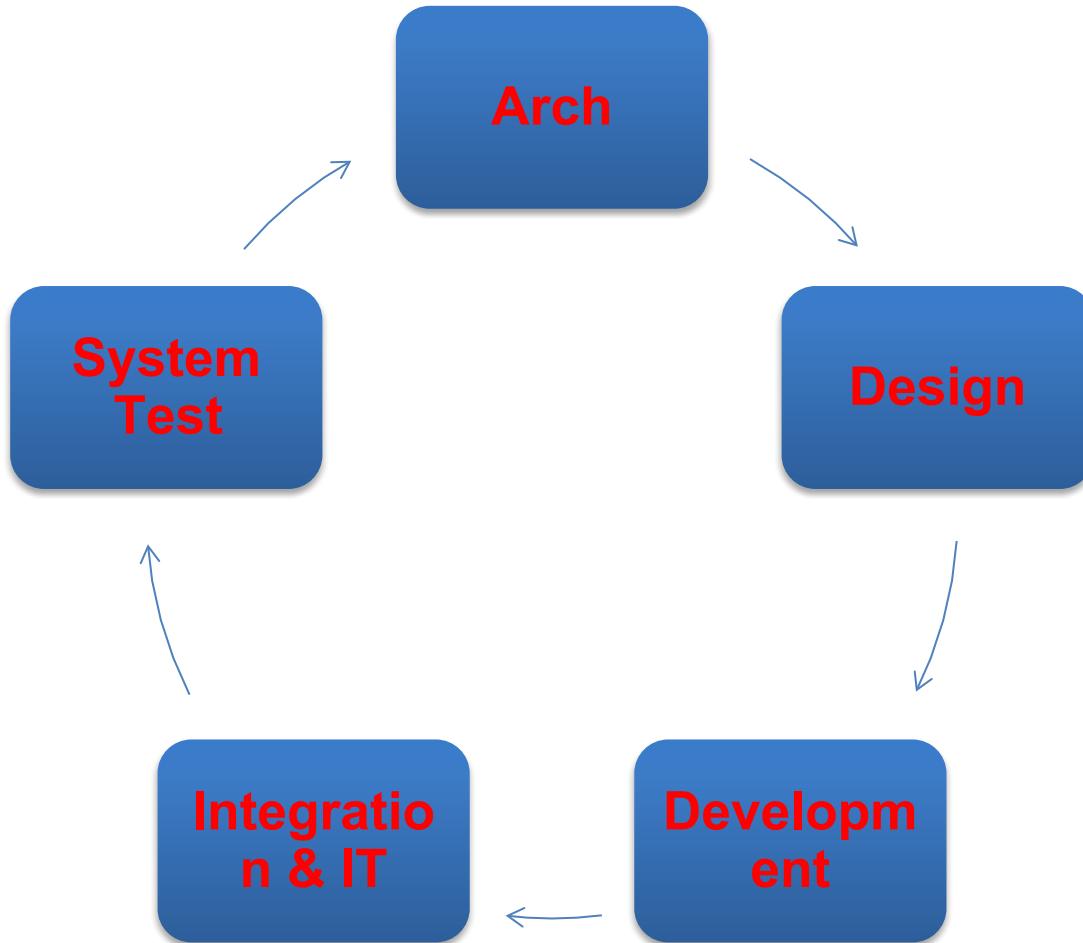
The Waterfall



The Waterfall

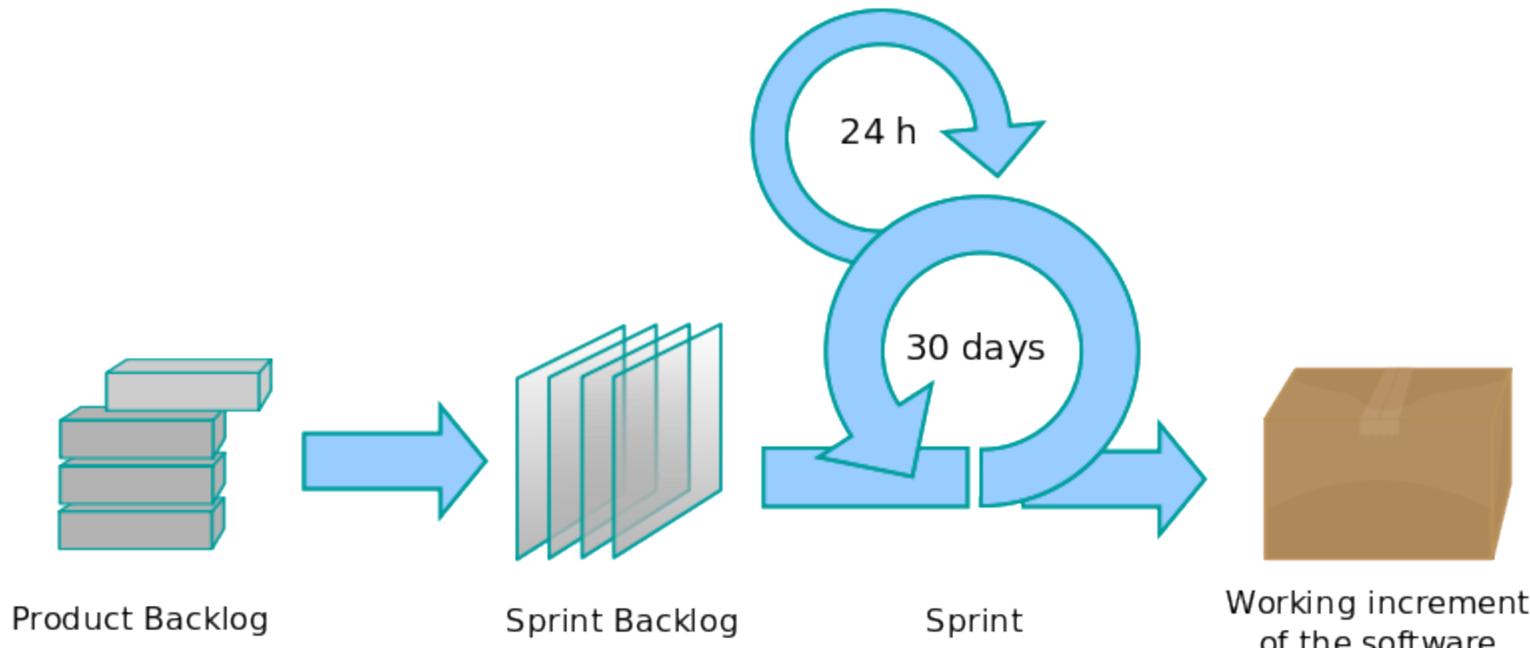


Iterative



- A View
- A Debate

Scrum



Reference: SCRUM Development Process by Ken Scheweber



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 13.4 : Example and Case

Garage Door

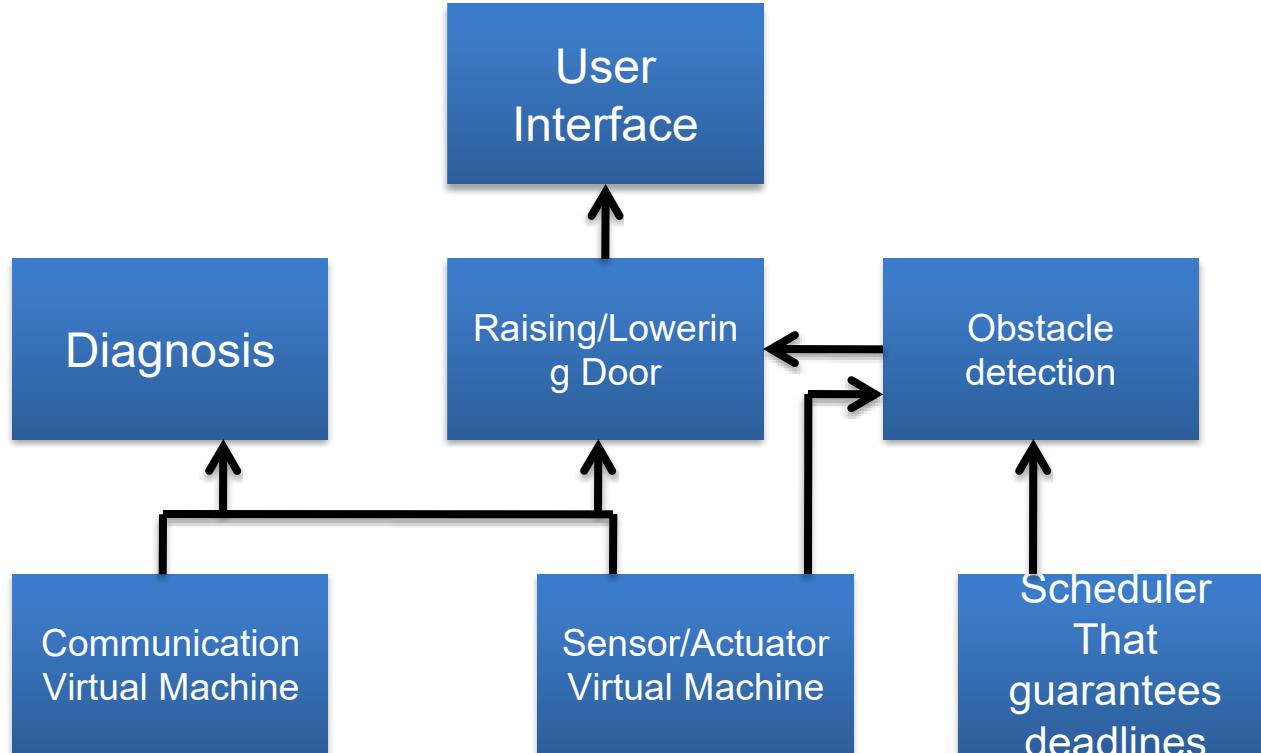
- The device controls for opening and closing the door are different for the various product lines. They may include controls from within a home information system. The product architecture for a specific set of controls should be directly derivable from the product line architecture
- The processor used in different processes will differ. The product architecture for each specific processor should be directly derivable from the product line architecture
- If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) within 0.1 second
- The garage door opener should be accessible for diagnosis and administration from within the home information system using product specific diagnosis protocol. It should be possible to directly product an architecture that reflects this protocol

Building Lighting Control System

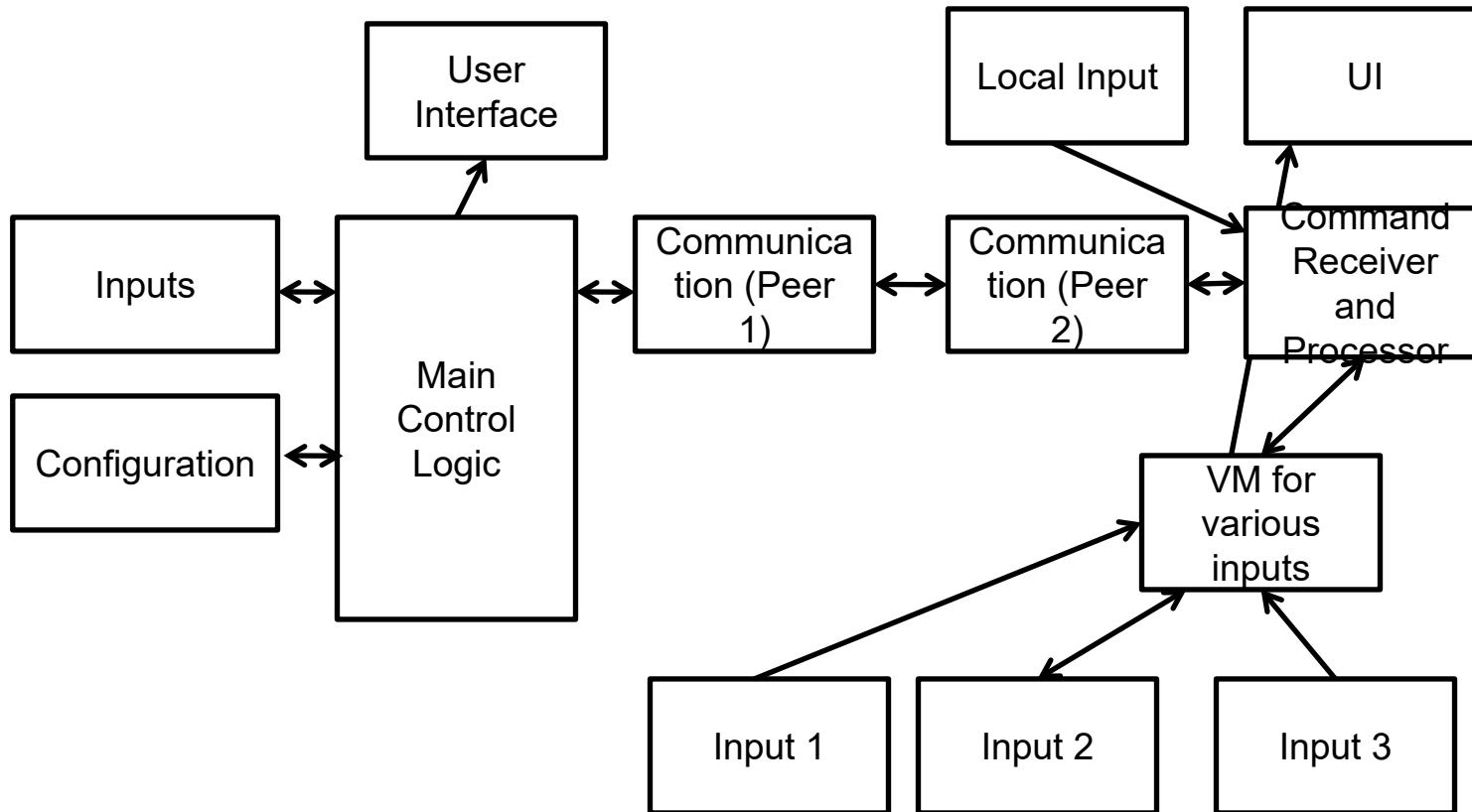


- Lighting control system for a apartment complex
- Emergency lights are UPS powered
- Corridor lights are timed and based on ambient light
- Garden and Architectural lights are timed

Garage Door



Building Lighting System





Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 13 Self Study

- SS13.1 Research the Agile Methodologies and find out various way in which developed software is expected to be tested. Find out and discuss the specific mechanisms and tools which are required for those approaches.



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 8.1 Systems & Systems of Systems

8.1 Self Study

To explore:

- Systems and Systems of Systems around us

Study Work:

- Review systems and system of systems around us
- Write down their characteristics and working (in terms of use cases)
- Further, analyse use of techniques learnt so far for designing test cases
- Document your findings in form of a whitepaper (IEEE format recommended)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 14: Agenda

Module 14: Test Adequacy & Enhancement

Topic 14.1

Test Adequacy – Need & Overview

Topic 14.2

Test Adequacy Assessment – Data Flow

Topic 8.3

Test Adequacy Assessment – Control Flow

Topic 8.3

Examples



Topic 14.1: Test Adequacy – Need & Overview

-
- Is my testing adequate?
 - Is my testing complete?
 - Is my testing good enough?
 - Have I tested my program thoroughly?
 - Is testing covering all that needs to be covered

- Is my testing adequate?
 - Is my testing complete?
 - Is my testing good enough?
 - Is testing covering all that needs to be covered
- What do you think?**

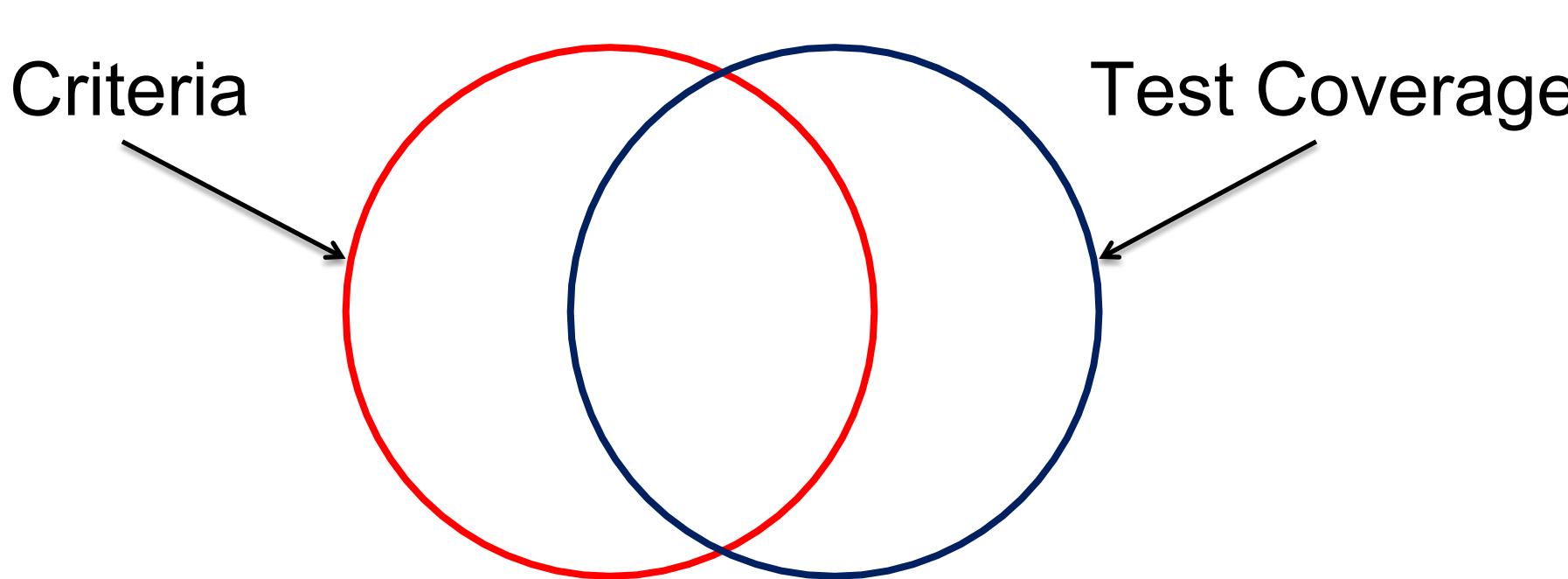
Some more questions

- What is test adequacy?
- What are the measures of test adequacy?
- If not adequate, How do we enhance tests?
- Achievable goals
 - Feasibility
 - Error Detection

Test Adequacy

- For us the term **adequate** means
 - Thorough
 - Good or Great
 - Completeor any other term you would like to suggest
- A test set is considered adequate if it satisfies a specific criteria
- A test set T is adequate to test program P , iff it satisfies the criteria C

Visual Representation



- **Establish the criteria**
- Use appropriate instrumentation
- Measure the coverage
- Enhance as required

Measurement of Test Adequacy



- For each adequacy criterion C, we derive a finite set known as the coverage domain and denoted by C_e
- A criterion C is a **white-box test adequacy** criterion if the corresponding coverage domain C_e depends solely on program P under test
- A criterion C is a **black-box test adequacy** criterion if the corresponding coverage C_e depends solely on requirements R for the program P under test

Test Enhancements using Measurements of Adequacy



- Increase the possibility of detecting any uncovered errors
- Adequacy does not guarantee an error-free program
 - Inadequate is a cause of worry for sure
- Identification of the deficiency helps in enhancement of the inadequate test set

Measure Enhance Adequacy

Measurement of Test Adequacy



- Measure Test Adequacy of T
- Measure if all elements of Ce are indeed covered by T
 - All covered: Adequate coverage
 - All not covered: Inadequate coverage
- k elements are covered out of n element
- Coverage is measured by k/n

Infeasibility and Test Adequacy



- An element of a coverage domain e.g. a path, might be infeasible to cover. This implies that there exists no test case that covers such an element
- It is quite possible that in spite of detecting the inadequacy an enhancement may not be possible.

Error Detection and Test Enhancement



- Test Enhancement To determine test cases that test the untested parts of the program
- Test Enhancement Uncover new errors of the program by exercising the new areas of the program

Test Adequacy & Execution

- Number of times program is executed with certain inputs will enable newer error detection
- Single or multiple execution matter
- Same input across multiple runs can detect the error which in a single execution may not be possible to detect



BITS Pilani

Software Testing Methodologies

Prashant Joshi



Topic 14.2: Test Adequacy Assessment – Data Flow

Test Adequacy – Data Flow

- The d, u and k
- DU Pairs (and other pairs)
- c-use & p-use

Infeasible Flow

- Nodes are not reachable
- Hence cannot determine the Data flow
- No test case or input may satisfy the path for the data flow of a variable



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 14.3: Test Adequacy Assessment – Control Flow

Test Adequacy – Control Flow

- Statement and Block Coverage
- Conditions and Decisions
- Multiple Condition Coverage

Infeasible Flow

- Unreachable code – unreachable Control Flow
- Uncovers “dead code”



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 14.4: Examples

Suggested Examples

- Review the sample code in T2 Chapter 7 in appropriate sections



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 15: Agenda

Module 15: Test Case Minimization, Prioritization & Optimization (1/2)

Topic 15.1

Need & Motivation

Topic 15.2

Techniques

Topic 15.3

Regression Testing – Test Selection
(Execution Trace, Dynamic Slicing)



Topic 15.1: Need & Motivation

Need & Motivation

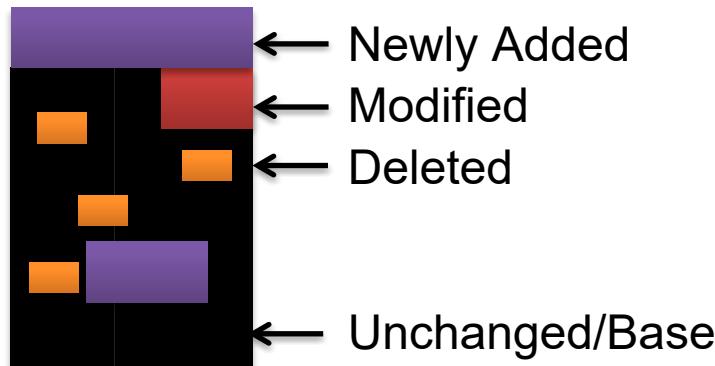
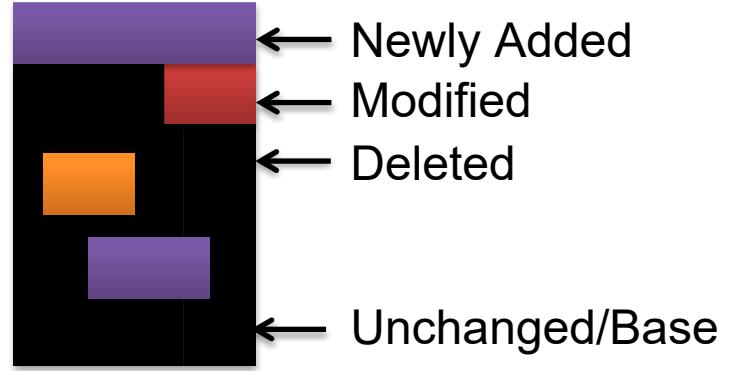
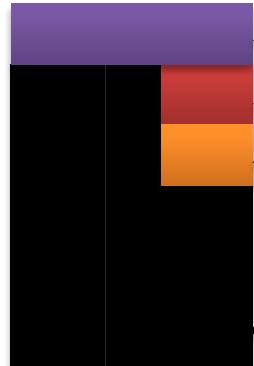
- All systems change over a period of time
 - Upgrades
 - Defect fixes
 - Roadmap – Platforms, change in technology
- On change
 - Regression testing takes front seat
 - Product = Base + Change
- Ensure
 - Base works as it should
 - New features and fixes work as well
 - System as a whole is of high quality

Regression Testing

- A program that is modified for reasons such as correction, addition of features etc., is often retested to ensure that the unchanged parts of the program continue to work correctly. Such testing is commonly referred to as regression testing

Regress means to return to previous,
usually worst, state

A Minefield!



A
new
form!

Definitions - Optimisation

- Dictionary: an act, process, or methodology of making something (as a design, system, or decision) as fully perfect, functional, or effective as possible
- In mathematics, computer science, or management science, mathematical optimization is the selection of a best element from some set of available alternatives

Definitions - Minimisation

- Dictionary: to make (something bad or not wanted) as small as possible
- In mathematics, computer science, or management science, mathematical optimization is the selection of a least set of elements from the set of available alternatives

Our View

- Optimization
- Minimization
- Prioritization

Motivations & Constraints

- Resource
 - Effort
 - Staff
 - Time
 - Budget
- Management
 - Do more with less
 - Best use of resources
- Technical
 - Get the most efficacy



BITS Pilani

Software Testing Methodologies

Prashant Joshi



Topic 15.2: Techniques

Regression Test Process

- Selection, Prioritization and Minimization
- Test Setup
- Test Sequencing
- Test Execution
- Output Comparison

Test Selection

- Test All
- Random Selection
- Selecting Modification traversing tests
- Test Minimization
- Test prioritization

Test Minimization

- Reduce the set T
- Discard Redundant Test Cases
- Criteria
 - Code Coverage
 - Requirements Coverage
 - A form of data or control flow
 - **ANY OTHER**

Test Prioritization

- Select test cases based on priority
- Discard None
- Prioritization Criteria
 - Requirements Coverage
 - Code Coverage
 - Desired Metric

Regression Test Selection

- Algorithms for test selection
- Tools for test selection

Test Optimization

- Selection of best element
- Remember the “best of the breed”
- Criteria Based
- Technique Based
 - OATS (Orthogonal Array)
 - Combinatorial
 - Choice of Values in EC
 - Special Value



Software Testing Methodologies

BITS Pilani

Prashant Joshi



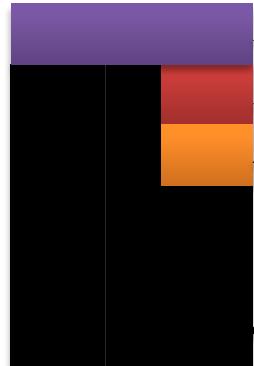
Topic 15.3: Regression Testing – Test Selection (Execution Trace/Dynamic Slicing)

Regression Testing

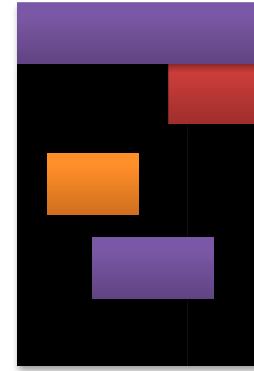
- A program that is modified for reasons such as correction, addition of features etc., is often retested to ensure that the unchanged parts of the program continue to work correctly. Such testing is commonly referred to as regression testing

Regress means to return to previous,
usually worst, state

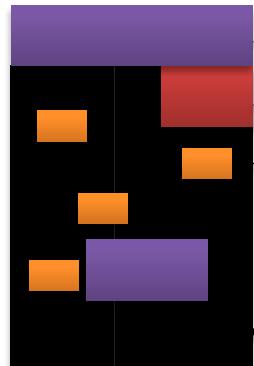
A Minefield!



← Newly Added
← Modified
← Deleted
← Unchanged/Base



← Newly Added
← Modified
← Deleted
← Unchanged/Base



← Newly Added
← Modified
← Deleted
← Unchanged/Base



A
new
form!

Use of Execution Trace

- Obtain the Execution Trace
- Selection of Regression Tests
- CFG
- PDG
- DU Pairs

Use of Dynamic Slicing

- Ensuring that the affected variable is part of the execution
 - Limitations
 - Better than Execution Trace
-
- CFG
 - Execution Trace
 - Dependence Graphs
-



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Test Minimisation

- The Example Code
- Review the Data Set
- Review the Test Set
- Only Control Flow and Data Flow may not be enough
- Redundancy removal with care! Utmost care!

Test Prioritisation

- Ranking
- No Discarding
 - Ranking by Context (Example: During development or Regression)
 - Absolute Ranking (At all times through out the product development, release & maintenance)

Test Design and Selection

- Pairwise Design – Binary Factors
- Pairwise Design – Multi-valued Factors
- Orthogonal Arrays

Pairwise – Binary

X1 Y1 Z1
X1 Y1 Z2
X1 Y2 Z1
X1 Y2 Z2
X2 Y1 Z1
X2 Y1 Z2
X2 Y2 Z1
X2 Y2 Z2

X1 Y1
X1 Y2
X1 Z1
X1 Z2
X2 Y1
X2 Y2
X2 Z1
X2 Z2
Y1 Z1
Y1 Z2
Y2 Z1
Y2 Z2

Pairwise – Mixed- Valued Factors



Block	Row	F1	F2	F3
1	1	1	1	1
	2	1	2	2
	3	1	3	3
2	1	2	1	2
	2	2	2	3
	3	2	3	1
3	1	3	1	3
	2	3	2	1
	3	3	3	2

Orthogonal Arrays

Run	F1	F2	F2
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Run	F1	F2	F2	F4
1	1	1	1	1
2	1	2	2	3
3	1	3	3	2
4	2	1	2	2
5	2	2	3	1
6	2	3	1	3
7	3	1	3	3
8	3	2	1	2
9	3	3	2	1

Examples

- Cross Platform Application
- Web Browser on various platforms



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi

Module 16: Agenda

Module 16: Test Case Minimisation, Prioritisation & Optimisation

Topic 16.1

Minimisation, Prioritisation & Optimisation Techniques

Topic 6.2

Test Selection Algorithms

Topic 16.3

Examples



Topic 16.1: Minimisation, Prioritisation & Optimisation Techniques

Test Minimisation

- The Example Code
- Review the Data Set
- Review the Test Set
- Only Control Flow and Data Flow may not be enough
- Redundancy removal with care! Utmost care!

Test Prioritisation

- Ranking
- No Discarding
 - Ranking by Context (Example: During development or Regression)
 - Absolute Ranking (At all times through out the product development, release & maintenance)



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 16.2: Test Selection Algorithms

Test Design and Selection

- Pairwise Design – Binary Factors
- Pairwise Design – Multi-valued Factors
- Orthogonal Arrays

Pairwise – Binary

X1 Y1 Z1
X1 Y1 Z2
X1 Y2 Z1
X1 Y2 Z2
X2 Y1 Z1
X2 Y1 Z2
X2 Y2 Z1
X2 Y2 Z2

X1 Y1
X1 Y2
X1 Z1
X1 Z2
X2 Y1
X2 Y2
X2 Z1
X2 Z2
Y1 Z1
Y1 Z2
Y2 Z1
Y2 Z2

Pairwise – Mixed- Valued Factors



Block	Row	F1	F2	F3
1	1	1	1	1
	2	1	2	2
	3	1	3	3
2	1	2	1	2
	2	2	2	3
	3	2	3	1
3	1	3	1	3
	2	3	2	1
	3	3	3	2

Orthogonal Arrays

Run	F1	F2	F2
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Run	F1	F2	F2	F4
1	1	1	1	1
2	1	2	2	3
3	1	3	3	2
4	2	1	2	2
5	2	2	3	1
6	2	3	1	3
7	3	1	3	3
8	3	2	1	2
9	3	3	2	1



Software Testing Methodologies

BITS Pilani

Prashant Joshi



Topic 16.3: Examples

Examples

- Cross Platform Application
- Web Browser on various platforms



Software Testing Methodologies

BITS Pilani

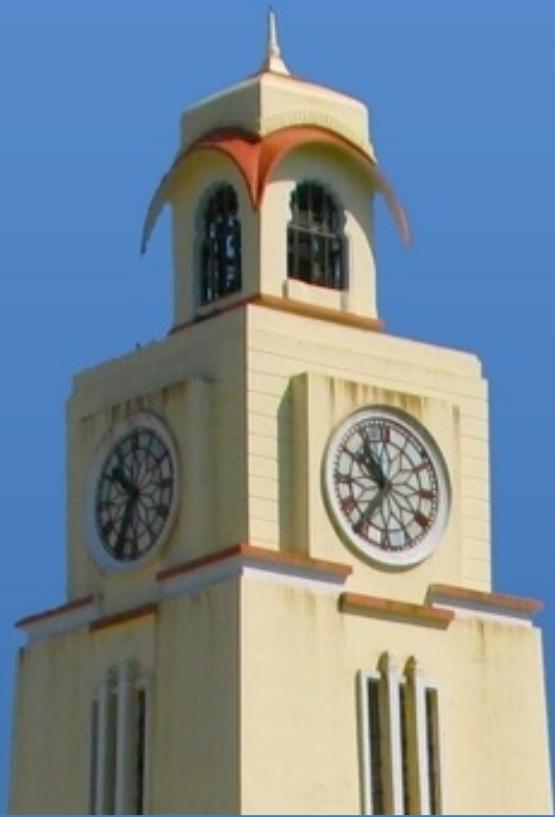
Prashant Joshi



Software Testing Methodologies

BITS Pilani

Prashant Joshi



SS 16.1 Compare & Contrast – Combinatorial Techniques

16.1 Self Study

To explore:

- Compare & contrast use of combinatorial techniques for test reduction

Study Work:

- Identify and study various combinatorial test techniques
- Review the test techniques which enable reduction of test cases
- Compare and contrast those techniques
- Comment on the risk of reduction of test cases



Software Testing Methodologies

BITS Pilani

Prashant Joshi