Emerging Trends in Software Test Plan Automation

Dr Ravi Gorthi and Dr Anjaneyulu Pasala
Test Automation Research Lab
Software Engineering and Technology Labs
Infosys Technologies Limited, Bangalore, India

Topics

Session # 1

- An Overview
  - Activities of Test Planning
  - What are Test Scenarios, Cases and Data
  - Relevance of Test Planning
  - Current Practices and their Limitations
  - Emerging Trends

- Model Based Approach to Test Plan Automation
  - Test Case Generation
    - Generation of Test Scenarios and Cases from UML UCAD
    - Generation of Test Scenarios and Cases from State Diagrams
    - Case Studies
  - Regression Test Case Selection
    - Selection of Regression Test Cases from consecutive versions of UCADs
    - Case Studies

(Break for 15 mins)

Topics

Session # 2

- (Source / Executable) Code Based Approach to Test Plan Automation
  - Selection of Regression Test Cases
    - Current Practices and their Limitations
    - Selection through Static Analysis of Source Code
    - Selection through Dynamic Analysis of Executable Code
    - Case Study

Wrap-up (15 minutes)

An Overview

- Activities of Test Planning
  - Analysis System Requirements Specifications (SRS)
  - Preparation of Test Scenarios, Cases, Data and Scripts
  - Selection of Regression Test Scenarios, Cases, Data and Scripts
  - Test Effort Estimation and Resource Allocations
  - Preparation of Test Execution Set-up

Test Case Generation

- What are Test Scenarios, Cases and Data
  - Test Scenarios
    - Testing needs of a logical unit of SRS (e.g. use-case / a logical path in a use-case)
  - Test Cases
    - Test requirement descriptions related to each unit of stimuli-response
    - Typically described in terms of <Inputs to software system, A set of conditions, Outputs from software system>
      - E.g. <Input: user-id and password; Condition: incorrect password; Output: error message-1>
  - Test Data (corresponding to each test case)
    - Specifications of test inputs, conditions and expected output
      - E.g. <Inputs: ravi_gorthi and abc2$password; Condition: abc2$password is an incorrect password; Output: “Invalid Password - enter password again”>

An Overview

- Relevance of Test Planning
  - NIST report 2002
    - Software Errors Cost U.S. Economy $59.5 Billion Annually
    - More than a third of these costs ($22.2 b) could be eliminated by an improved testing infrastructure
    - An undetected defect, post software deployment, costs $14K to fix
  - Medium to large s/w applications are observed to have anywhere between 5K to 100K test cases
  - Typically consumes 30 to 50% of *total testing efforts* (testing typically consumes 40 to 60% of total software TCO)
  - Every 10% improvement in productivity and quality of testing can lead to saving of millions of dollars

An Overview

- Current Practices and their Limitations

  (The following observations are typical across the IT world)

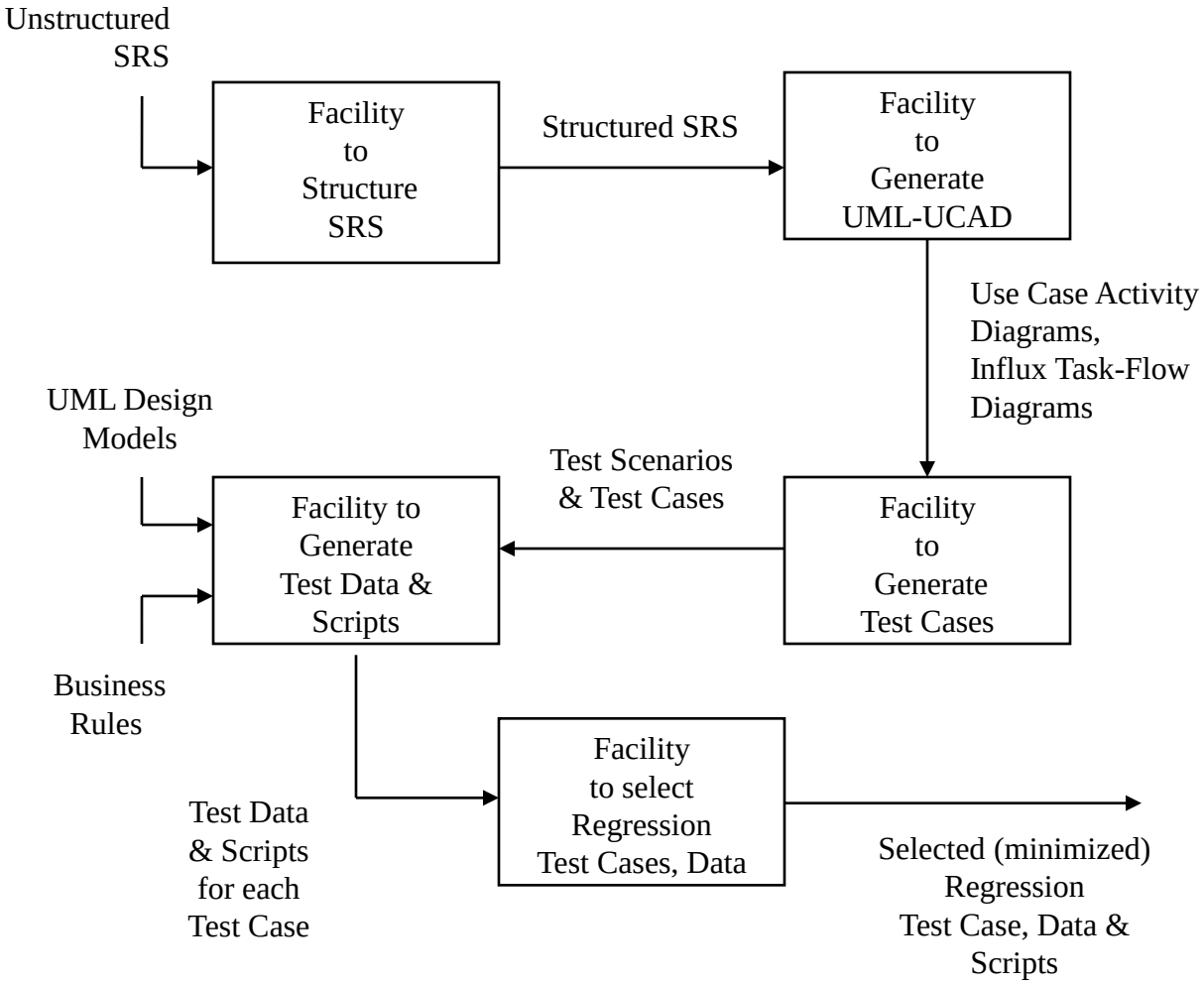
  – Manual Analysis of SRS

  – Manual Generation of Test Scenarios, Cases, Data and Scripts

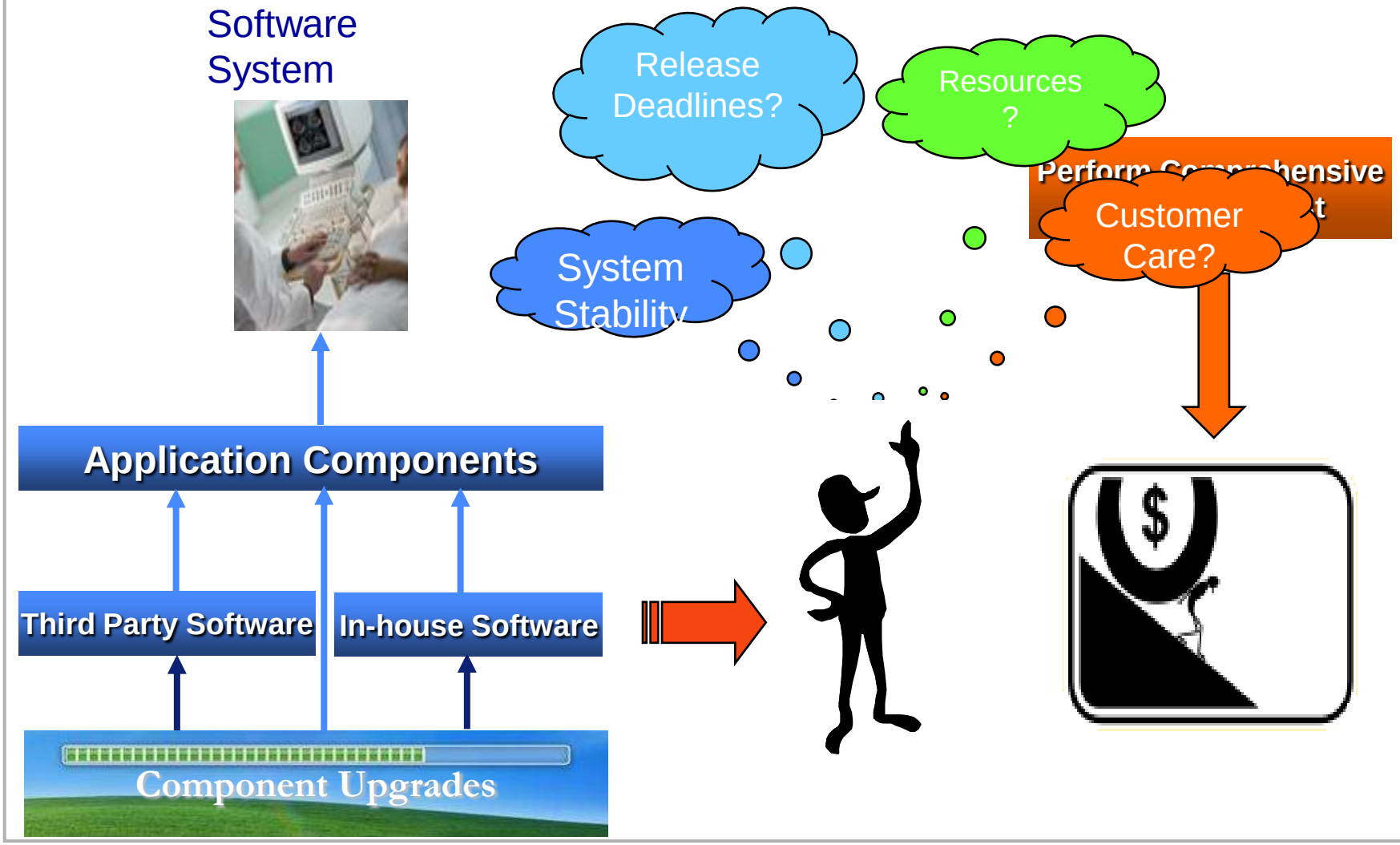
  – Manual Selection of Test Scenarios, Cases, Data and Scripts

  – Lack of fine grained analysis techniques for test effort estimation

Emerging Trends: Model Based Approach



Emerging Trends: Code Based Approach



Emerging Trends: Code Based Approach

Software System

Application Components

Third Party Software

In-house Software

Upgrades Analyzer

Patch Impact Analysis
Suggested Regression Test Cases

Savings

Topics

Session # 1

- ✓ An Overview
  - ✓ Activities of Test Planning
  - ✓ What are Test Scenarios, Cases and Data
  - ✓ Relevance of Test Planning
  - ✓ Current Practices and their Limitations
  - ✓ Emerging Trends

- Model Based Approach to Test Plan Automation
  - Test Case Generation
    - Generation of Test Scenarios and Cases from UML UCAD
    - Generation of Test Scenarios and Cases from State Diagrams
    - Case Studies
  - Regression Test Case Selection
    - Selection of Regression Test Cases from consecutive versions of UCADs
    - Case Studies

(Break for 15 mins)

Model Based Approach to Test Plan Automation

- Generation of
  - system test , regression test and integration test
    - scenarios, cases, data and scripts

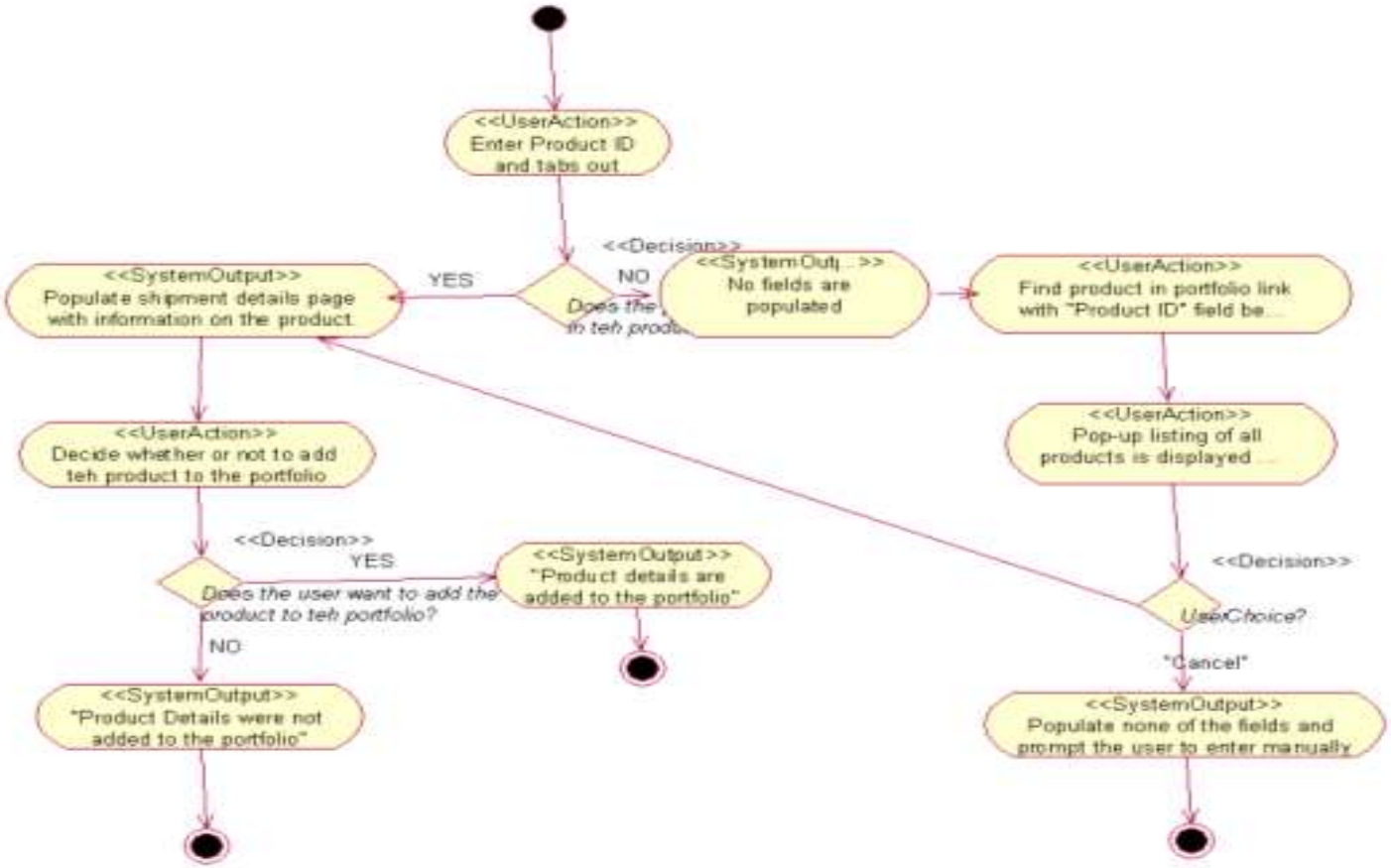
  from

  - Analysis models
    - UML Use-Case Activity Diagrams, State Diagrams, Communication, Collaboration Diagrams
  - Design Models
    - Class Diagrams, Sequence Diagrams

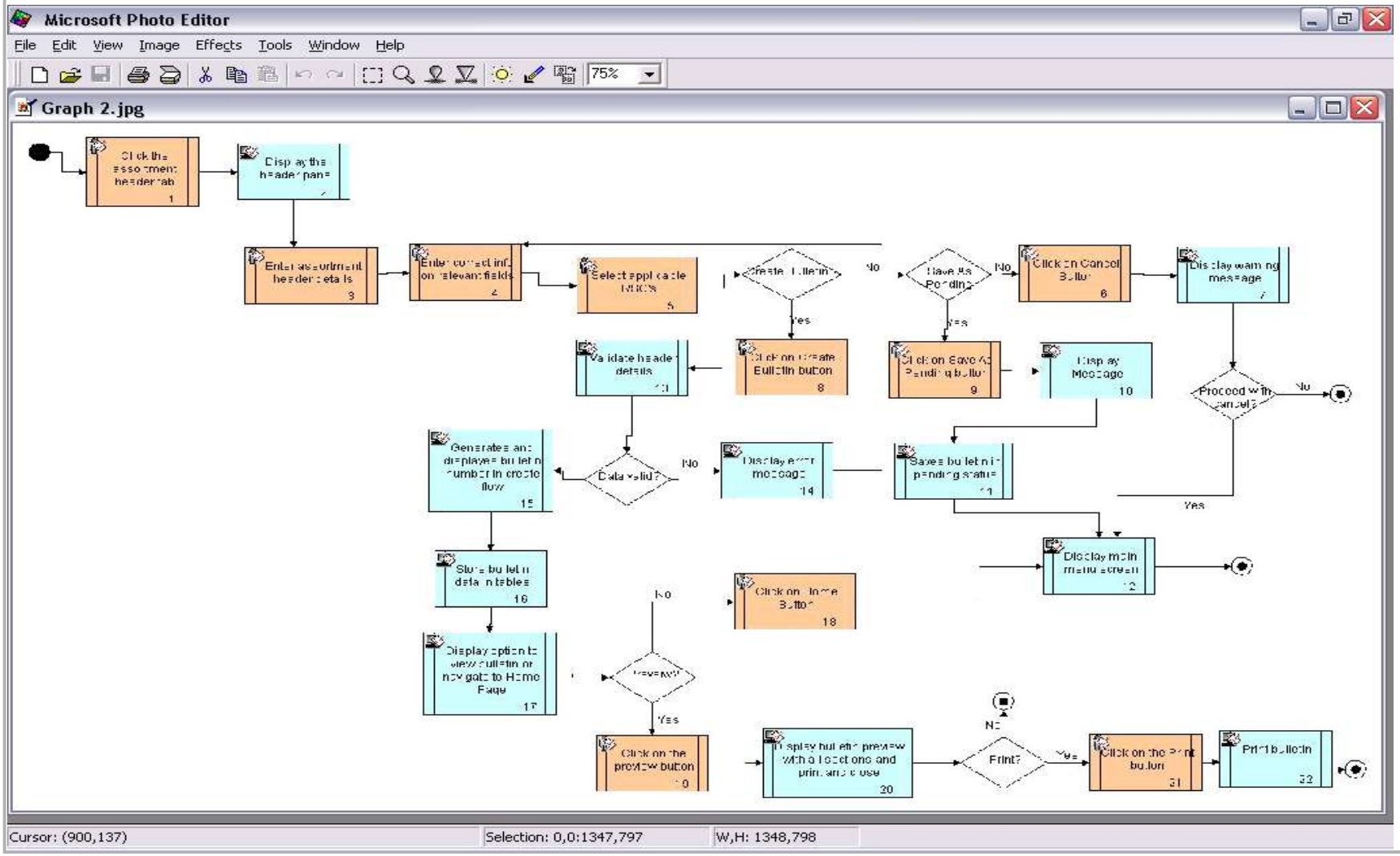
Test Case Generation

- From UML Use-Case Activity Diagrams (UCAD)
  - UCAD: Directed Cyclic Graph
  - Automatic Generation of Test Scenarios
    - One can automatically generate test scenarios through a DFS with restrictions on traversal of cyclic paths
  - Automatic Generation of Test Cases
    - One can automatically generate test cases by slicing each scenario into tuples of <Inputs, Processing, Conditions, Expected Output>

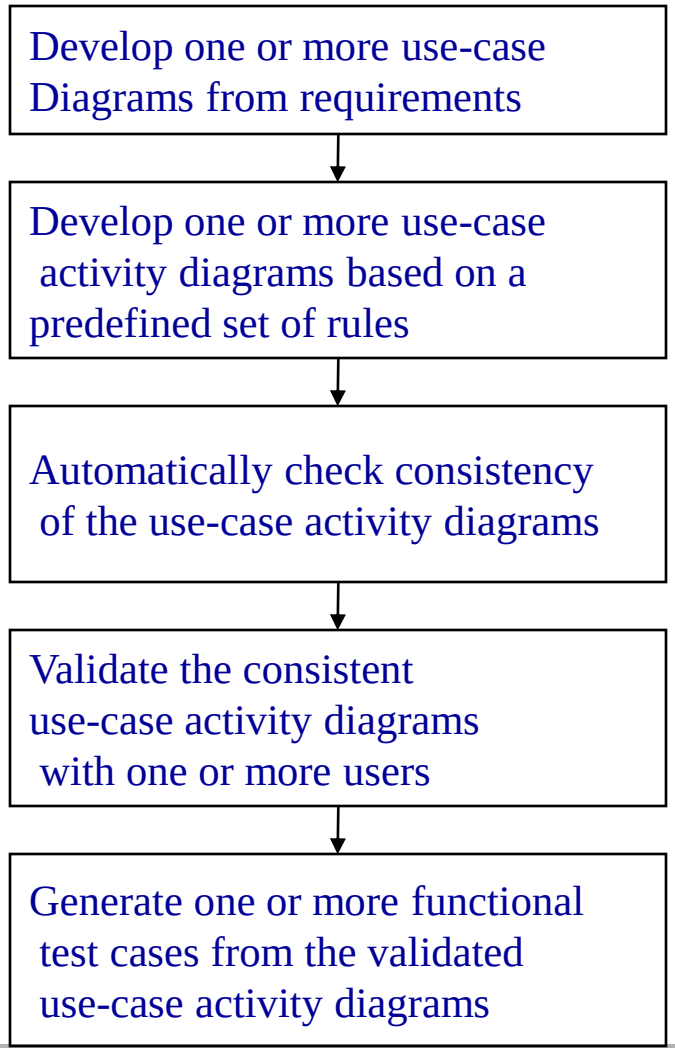
Sample UML Use Case Activity Diagram (UCAD)



Sample UML Use Case Activity Diagram (UCAD)



Test Case Generation from UCADs: Methodology



Test Case Generation from UCADs: Case Study

- Consider the following use case from ‘Automatic Teller Machine (ATM) System’: With-draw Cash using a debit-card from ATM
  - System displays the msg: “Enter the debit card number (Swipe the card)”
  - User inputs the debit-card number (swipes the card)
  - System validates the card (assume the card number is valid)
  - System displays the msg: “Enter the PIN”
  - User enters the PIN
  - System Validates the PIN (assume the PIN is valid)
  - System displays the menu of choices and the msg: “Choose from the Menu”
  - User selects the Menu option: ‘With-draw Cash from SB Account”
  - System displays the msg: “Enter the amount”
  - User enters the amount
  - System validates the amount (assume the amount is correct)
  - System displays the msg: “Take the amount; Thanks you!”

Test Case Generation from UCADs: Case Study

Space left for participant to draw the UML Use-Case Activity Diagram

Test Case Generation from UCADs: Case Study

- Add the following exceptions to the ATM Use Case
  - Debit-card swiped by the user is invalid
  - PIN entered by the user is invalid (the 1st time error)
  - PIN entered by the user is invalid (the 3rd consecutive time error)
  - Amount entered by the user is invalid (amount > current balance)
  - Amount entered by the user is invalid (amount > daily upper-limit)

Test Case Generation from UCADs: Case Study

Space left for participant to draw the UML Use-Case Activity Diagram to handle the exception cases

Test Case Generation from UCADs: Case Study

Space left for participant to extract each path in the UCAD and generate test cases

Test Case Generation from UCADs: Case Study

Space left for participant to extract each path in the UCAD and generate test cases

Test Case Generation from UCADs: Case Study

Scenario #1

Precondition(s):
none

#	User Input	Conditions	Expected Output
1	Enter Product ID and tabs out	Does the product ID exist in teh product portfolie? = "NO"	1. No fields are populated
2	Find product in portfolio link with \"Product ID\" field being empty		
3	Pop-up listing of all products is displayed and select desired product and click on a link.	UserChoice? = n/a	1. Populate shipment details page with information on the product.
4	Decide whether or not to add teh product to the portfolio	Does the user want to add the product to teh portfolio? = "YES"	1. \"Product details are added to the portfolio\"

Scenario #2

Precondition(s):
none

#	User Input	Conditions	Expected Output
1	Enter Product ID and tabs out	Does the product ID exist in teh product portfolie? = "NO"	1. No fields are populated
	Find product in portfolio link with		

Results of TCG in a few Real-world Projects

PRODUCTIVITY FIGURES FROM FIELD TRIALS	**Healthcare-Claims**	**Internet Banking**	**Retail – Ecommerce**
No of test cases generated	1082	1772	23366
No of task flows involved	32	48	586
Effort estimated for manual procedure (Person Days)	23	38	465.9
Effort spent using our tool (Person Days)	13.5	17.6	132
% Effort Saving	41%	54%	72%

Test Case Generation

- From UML Use-Case Activity Diagrams (UCAD)
  - Peter Zielczynski [8] and Jim Heumann [9] offer the basics on the analysis of use-cases to manually generate test cases; good to start with.
  - Lionel Briand and Yvan Labiche [12], Clementine et al [10], Chen et al [11], Linzhang et al [13] and Chen et al [14] offer the next level of detail on semi-auto / auto generation of test cases from use-cases
  - Ravi Gorthi et al [1] use a novel concept from Paul Gerrard [15] called ‘unit of behavior’ to automatically generate test scenarios and test cases from UCADs

Test Case Generation

- From System Requirements Specifications (SRS)
  - SRS is typically semi-structured
  - Structured SRS
    - First break a given SRS into a set of Use-Cases
    - Express each Use-Case as ordered sequence of tuples of <Inputs, Conditions, Expected Output>
    - One can automatically generate Use-Case Activity Diagrams (UCADs) from thus Structured SRS
  - Automatic Generation of Test Scenarios
    - One can automatically generate test scenarios through a DFS with restrictions on traversal of cyclic paths
  - Automatic Generation of Test Cases
    - One can automatically generate test cases by slicing each scenario into tuples of <Inputs, Conditions, Expected Output>

Structuring SRS: Use Case Template

Name	UC-01 …		
Description	The use case has …		
Pre-condition	…		
Interaction Steps	Input	1	The user …
	Process (optional)	2	The system …

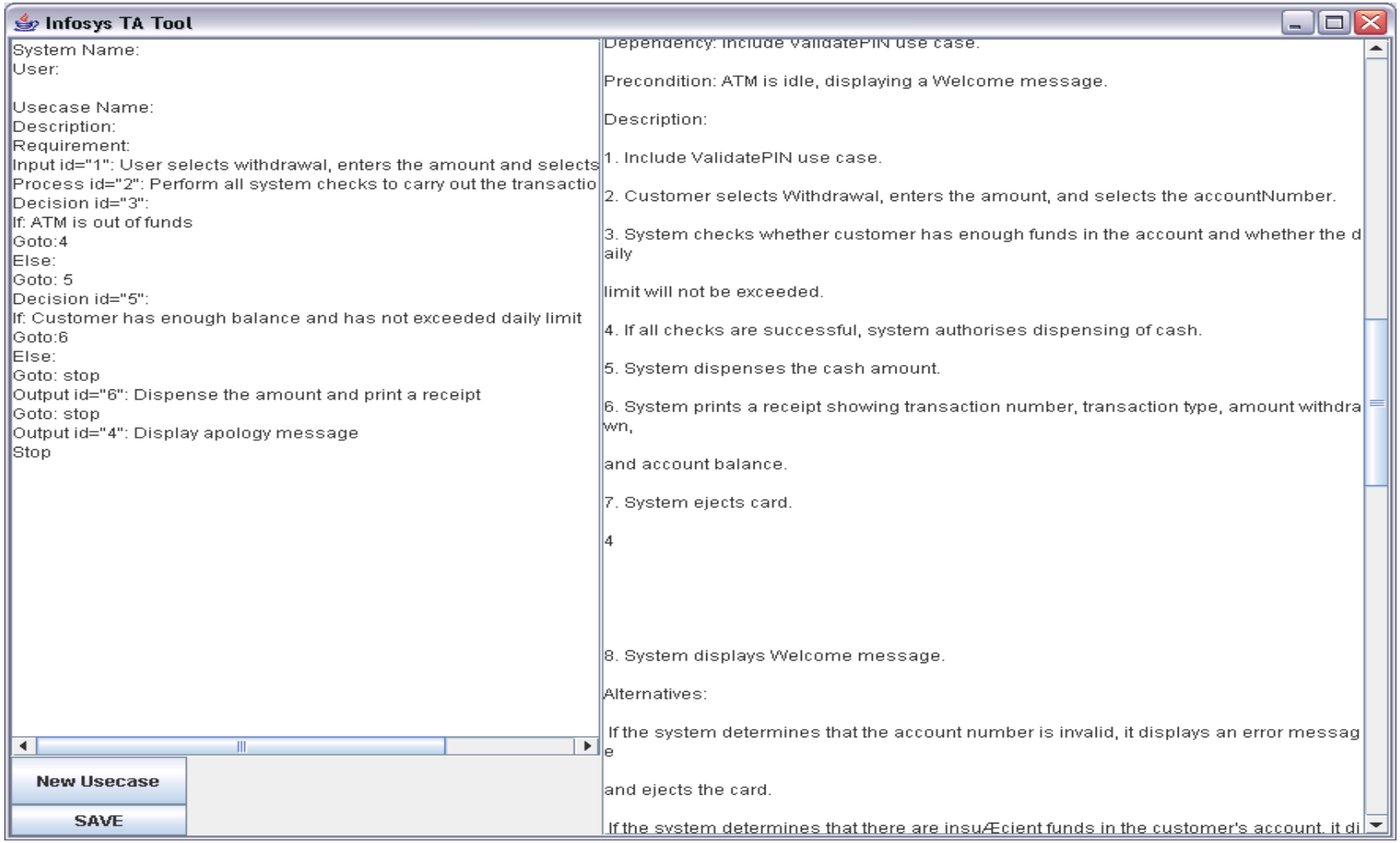
	Decision (optional)	3	If (Condition is True) Goto 4
			Else
			Goto 5

	Output	4	The system Displays …

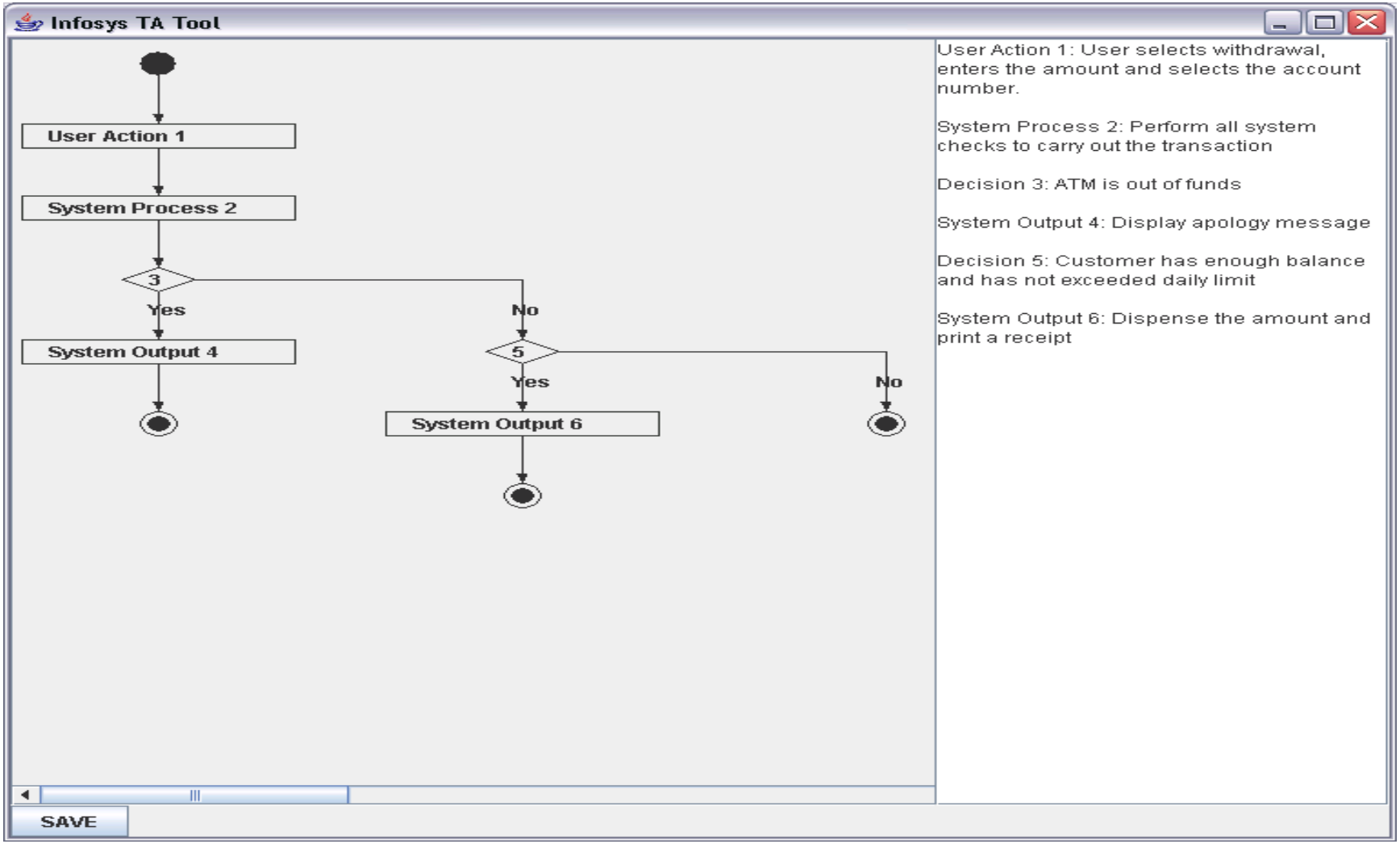
	Decision (optional)	5	If (…) Goto …
			Else
			Stop

	…	…	…
Post-Condition	…		
Non-Functional Requirements(Optional)	<Keywords such as frequency, performance, priority, fault tolerance>: Requirement Description		

ATM Example : Structuring SRS



Automatic Generation of Use Case Activity Diagrams (UCADs) from Structured SRS



Test Case Generation from Structured SRS

Scenario #1

Precondition(s):
none

#	User Input	Conditions	Expected Output
1	Enter Product ID and tabs out	Does the product ID exist in teh product portfolie? = "NO"	1. No fields are populated
2	Find product in portfolio link with \"Product ID\" field being empty		
3	Pop-up listing of all products is displayed and select desired product and click on a link.	UserChoice? = n/a	1. Populate shipment details page with information on the product.
4	Decide whether or not to add teh product to the portfolio	Does the user want to add the product to teh portfolio? = "YES"	1. \"Product details are added to the portfolio\"

Scenario #2

Precondition(s):
none

#	User Input	Conditions	Expected Output
1	Enter Product ID and tabs out	Does the product ID exist in teh product portfolie? = "NO"	1. No fields are populated
	Find product in portfolio link with		

Test Case Generation

- From UML State Diagrams / Charts
  - Philippe and Pascale [20], Supaporn and Wanchai [21] offer the basics on automatically generating of test cases from UML state diagrams
  - Stefania Gnesi et al [22] and Valdivino Santiago et all [23] offer the next level of details on the automated generation of test cases from UML state diagrams
  - Jeff Offutt et al [24] and Stefan Hildenbrand [25] offer great details on the analysis of state-based specifications and finite state automata to generate test cases.

Topics

Session # 1

- ✓ An Overview
  - ✓ Activities of Test Planning
  - ✓ What are Test Scenarios, Cases and Data
  - ✓ Relevance of Test Planning
  - ✓ Current Practices and their Limitations
  - ✓ Emerging Trends
- ✓ Model Based Approach to Test Plan Automation
  - ✓ Test Case Generation
    - ✓ Generation of Test Scenarios and Cases from UML UCAD
    - ✓ Generation of Test Scenarios and Cases from State Diagrams
    - ✓ Case Studies
  - – Regression Test Case Selection
    - Selection of Regression Test Cases from consecutive versions of UCADs
    - Case Studies

(Break for 15 mins)

Model Based Regression Test Case Selection

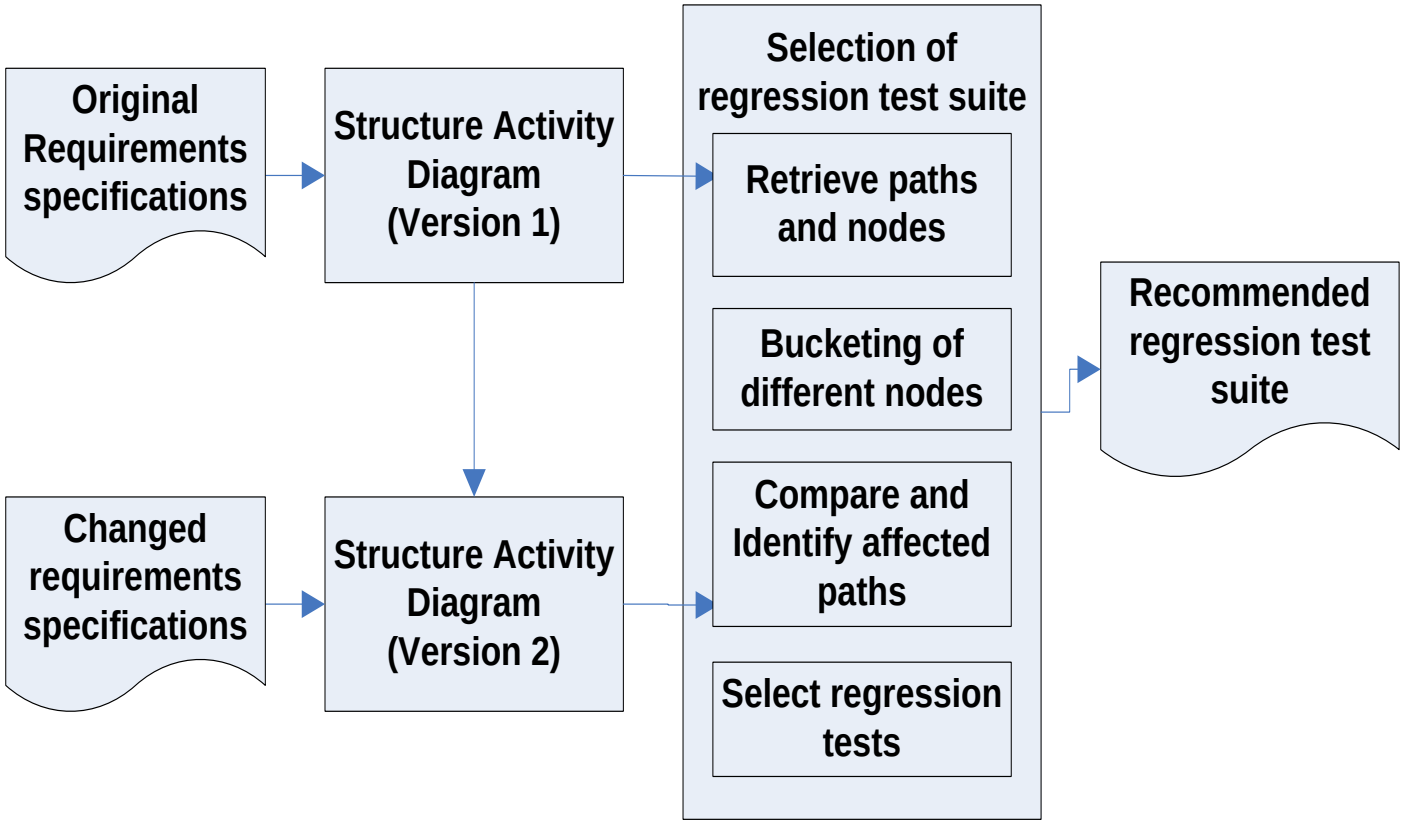
Some important issues related to regression testing:

- When a software application is modified due to ‘change requests’ received from business users, the application need to be tested to ensure that the modified system meets all the functional (and non-functional) requirements
- Typically, medium to large software applications are observed to involve,
  - a few tens of to a few hundreds of ‘o-o classes’
  - A few thousands of to a few tens of thousands of ‘methods’
- In general, each cycle of software maintenance involves changes to around 15 to 30% of methods
- There will be many classes and methods which remain unchanged across maintenance cycles
- Do we need to test a modified application, ***completely***? Answer is **NO!**

Model Based Regression Test Case Selection

- Input
  - Two ***consecutive versions*** of UML Use-Case Activity Diagrams (UCADs)
    - **UCAD-v1 and UCAD-v2**
- Processing
  - ***Identify*** changed nodes in UCAD-v2 with respect to UCAD-v1
    - **Modified, newly added, deleted or shifted nodes**
  - ***Select*** regression test suite
    - **Paths in the UCAD-v2 that are affected due to the above types of node changes**
  - ***Paths*** in UCAD-v2 that are the same as those in UCAD-v1 need ***NOT be tested***
- Output
  - ***Selected regression test suite*** to test UCAD-v2

Model Based Regression Test Case Selection

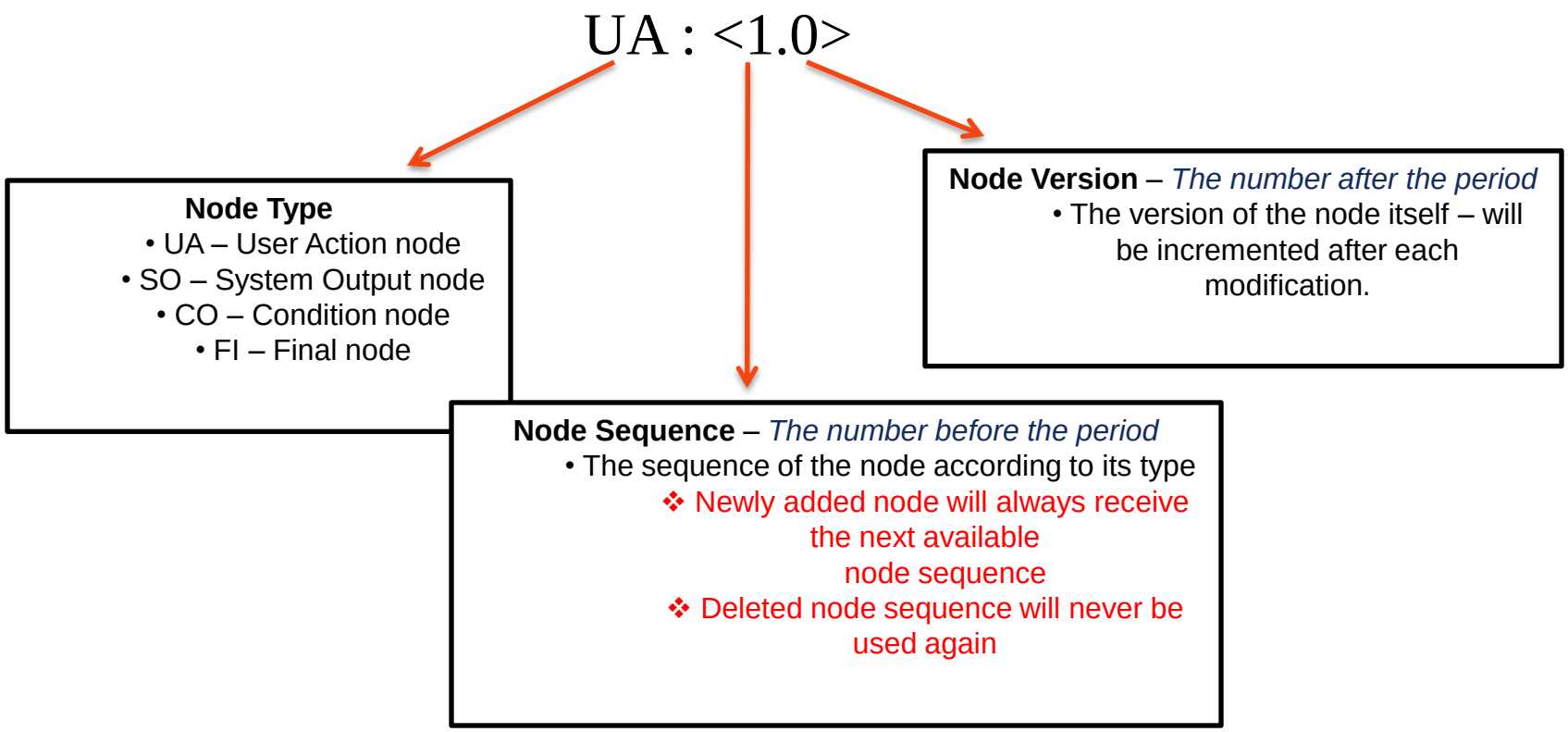


Model Based Regression Test Case Selection

- ***Identify*** changes to the nodes of UCAD-v2 with respect to UCAD-v1
  - Nodes in a given UCAD are structured as
    - **Node type <Sequence-number.Version-number>**
    - **E.g UA <1.0>; SP <2.3>; CO <3.1>; SO <1.4>**
  - All changes to the nodes of UCAD-v2 can be categorized as
    - **Modifications to an existing node**
    - **Deletion of an existing node**
    - **Shifting of an existing node**
    - **Addition of a new node**
- Generate ***Selected*** Regression Test Suite for UCAD-v2
  - Paths that contain one or more ***changed*** nodes

Model Based Regression Test Case Selection

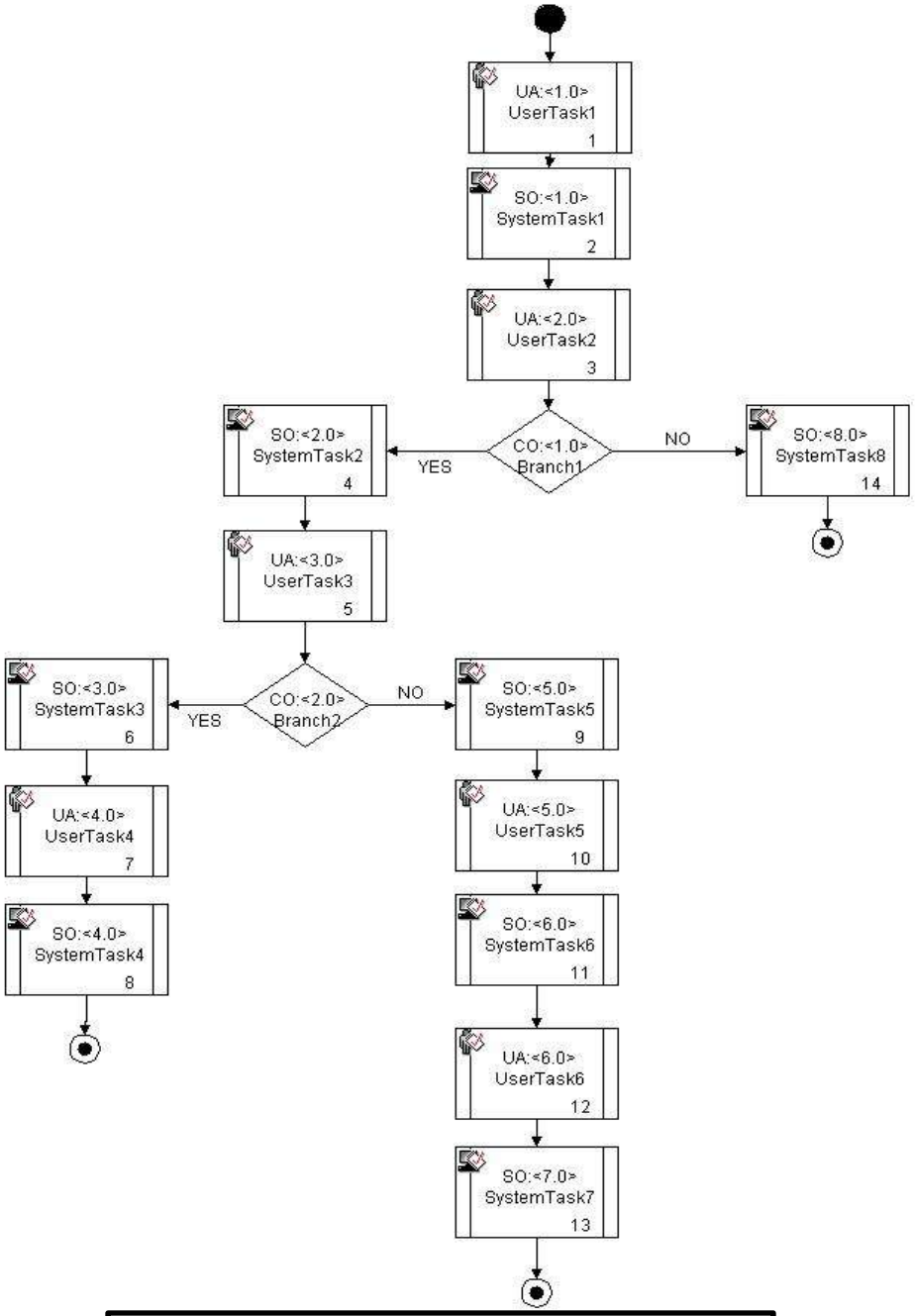
- Examples : UA: <1.0> , SO: <2.2>, CO: <3.0>, FI: <1.0>
- Structure of Node Version Number



Model Based Regression Test Case Selection

- Some nodes are more critical than the others.
- Example: Verification of PIN number and account balance VS. Verification of correct display of welcome message

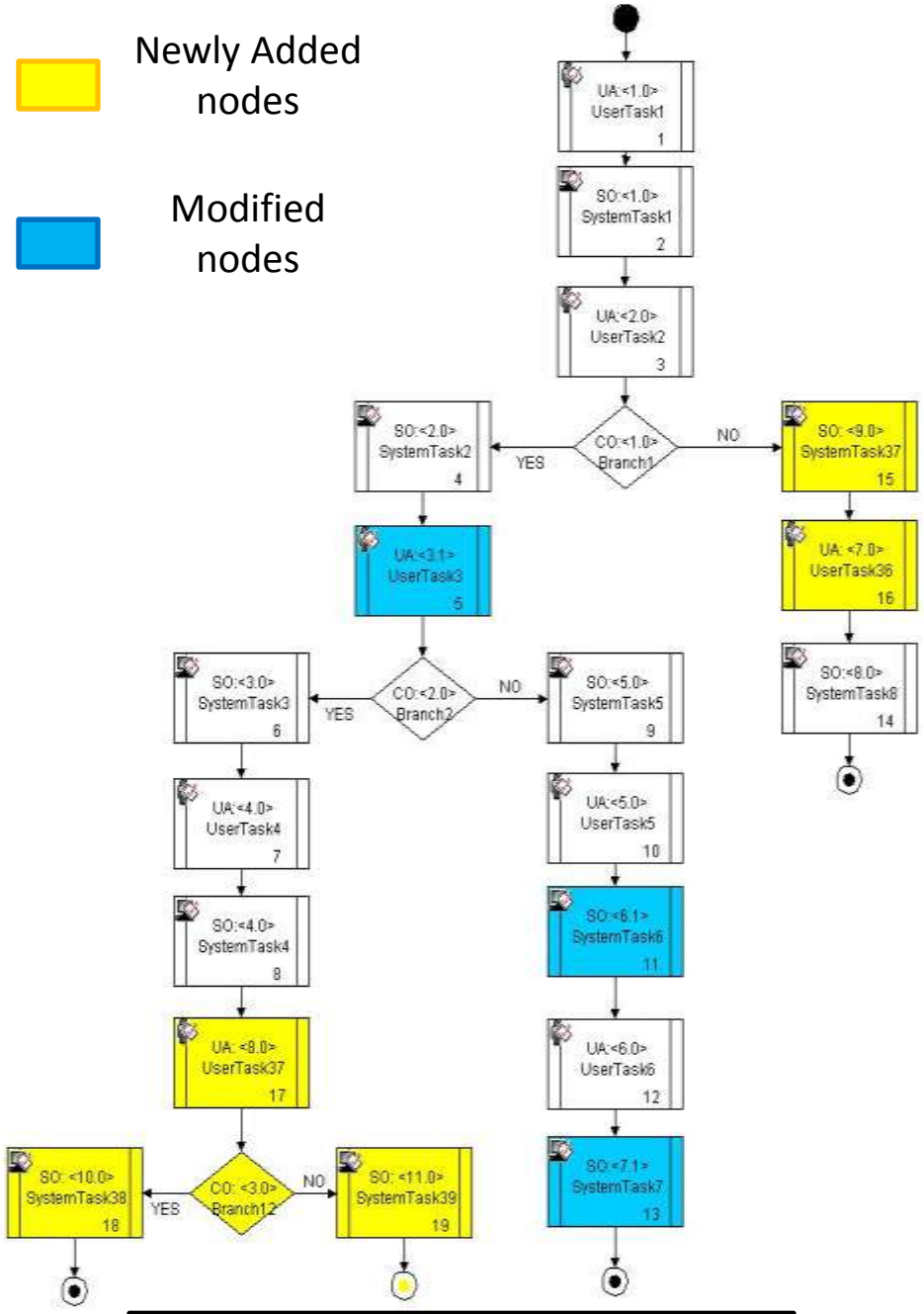
- Assign criticality to nodes
  - [H] – High criticality
  - [M] – Medium criticality
  - [L] – Low criticality (Default setting)



Original Influx Taskflow Diagram

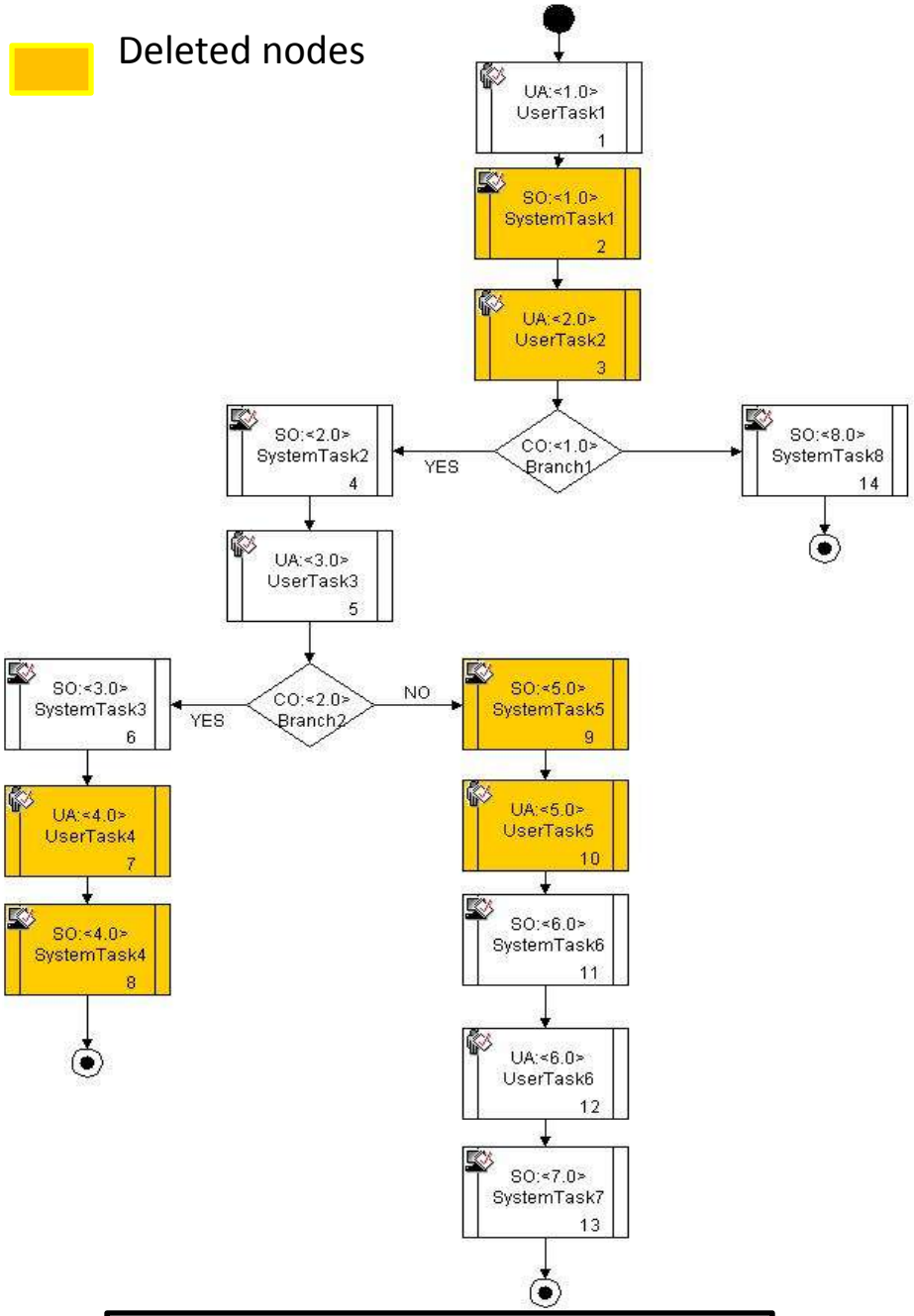
Newly Added nodes

Modified nodes



New Influx Taskflow Diagram

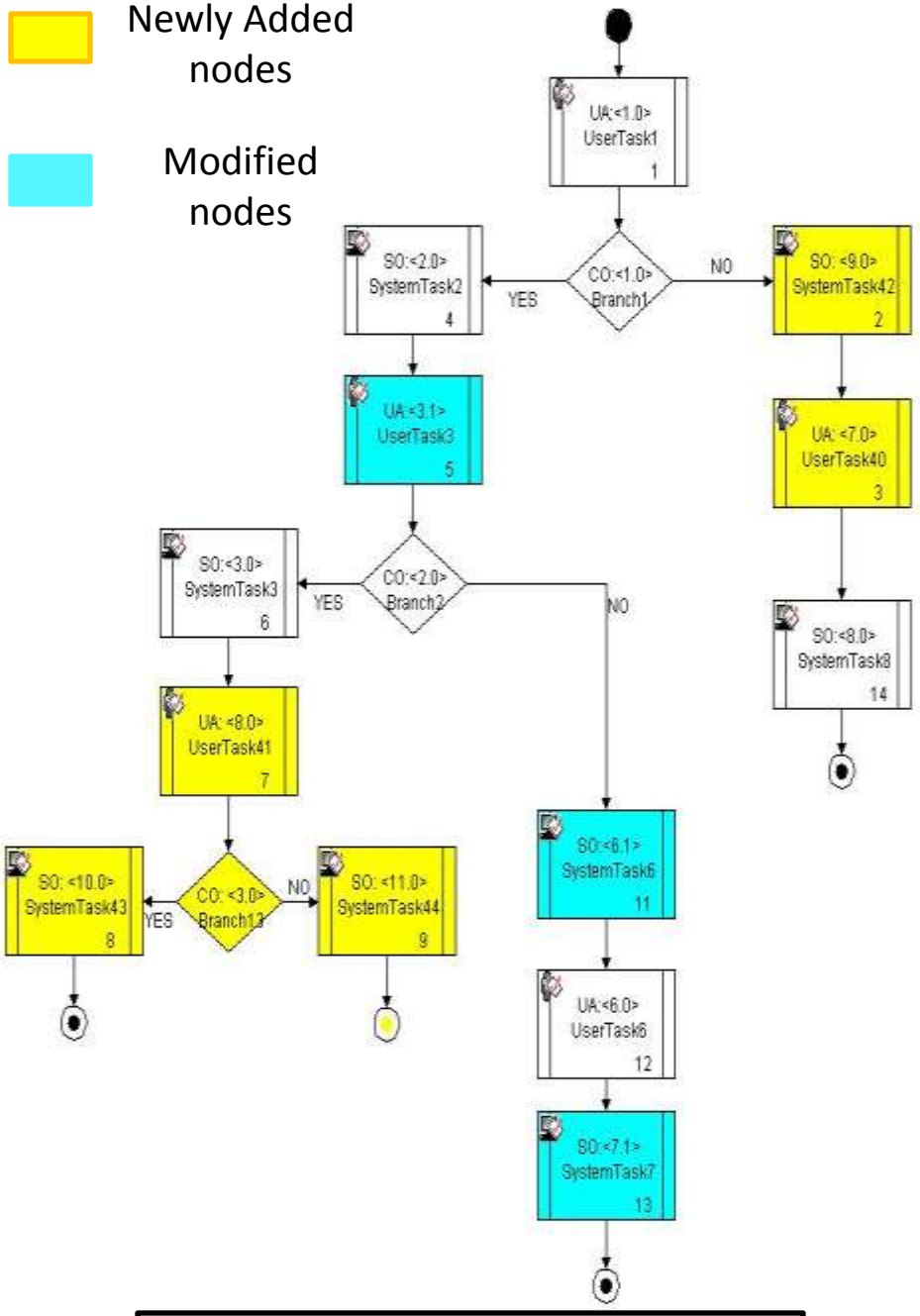
Deleted nodes



Original Influx Taskflow Diagram

Newly Added nodes

Modified nodes



New Influx Taskflow Diagram

Model Based Regression Test Case Selection

Description	New ID	Description	Status
Enters the operation to be performed	N/A	N/A	Deleted
Presses button to Withdraw cash	N/A	N/A	Deleted

Test Steps	Input	Conditions	Expected Output
1	Enter the card	Card inserted properly? = Yes	1. Display a message to enter the PIN
2	Enter the PIN	Cash withdrawl? = yes	1. Display a message to continue with the transa
3	Enter the amount	Cash available? = Yes	1. Display a message to collect the cash

Description	New ID	Description	Status
Enters the operation to be performed	N/A	N/A	Deleted
Presses button to Withdraw cash	N/A	N/A	Deleted

Experimental Results from a few Real-World Projects

	Healthcare-Claims	**Internet Banking**	**Retail Order Management**
No of UCAD paths involved	320	730	4300
No of regression test cases selected	1610	4780	20410
Effort estimated for manual procedure (Person Days)	23	78	466
Effort spent using our tool (Person Days)	13.5	37.6	186
Productivity gain	***41%***	***52%***	***60%***

Topics

- ✓ Session # 1
- ✓ An Overview
  - ✓ Activities of Test Planning
  - ✓ What are Test Scenarios, Cases and Data
  - ✓ Relevance of Test Planning
  - ✓ Current Practices and their Limitations
  - ✓ Emerging Trends
- ✓ Model Based Approach to Test Plan Automation
  - ✓ Test Case Generation
    - ✓ Generation of Test Scenarios and Cases from UML UCAD
    - ✓ Generation of Test Scenarios and Cases from State Diagrams
    - ✓ Case Studies
  - ✓ Regression Test Case Selection
    - ✓ Selection of Regression Test Cases from consecutive versions of UCADs
    - ✓ Case Studies

(Break for 15 mins)

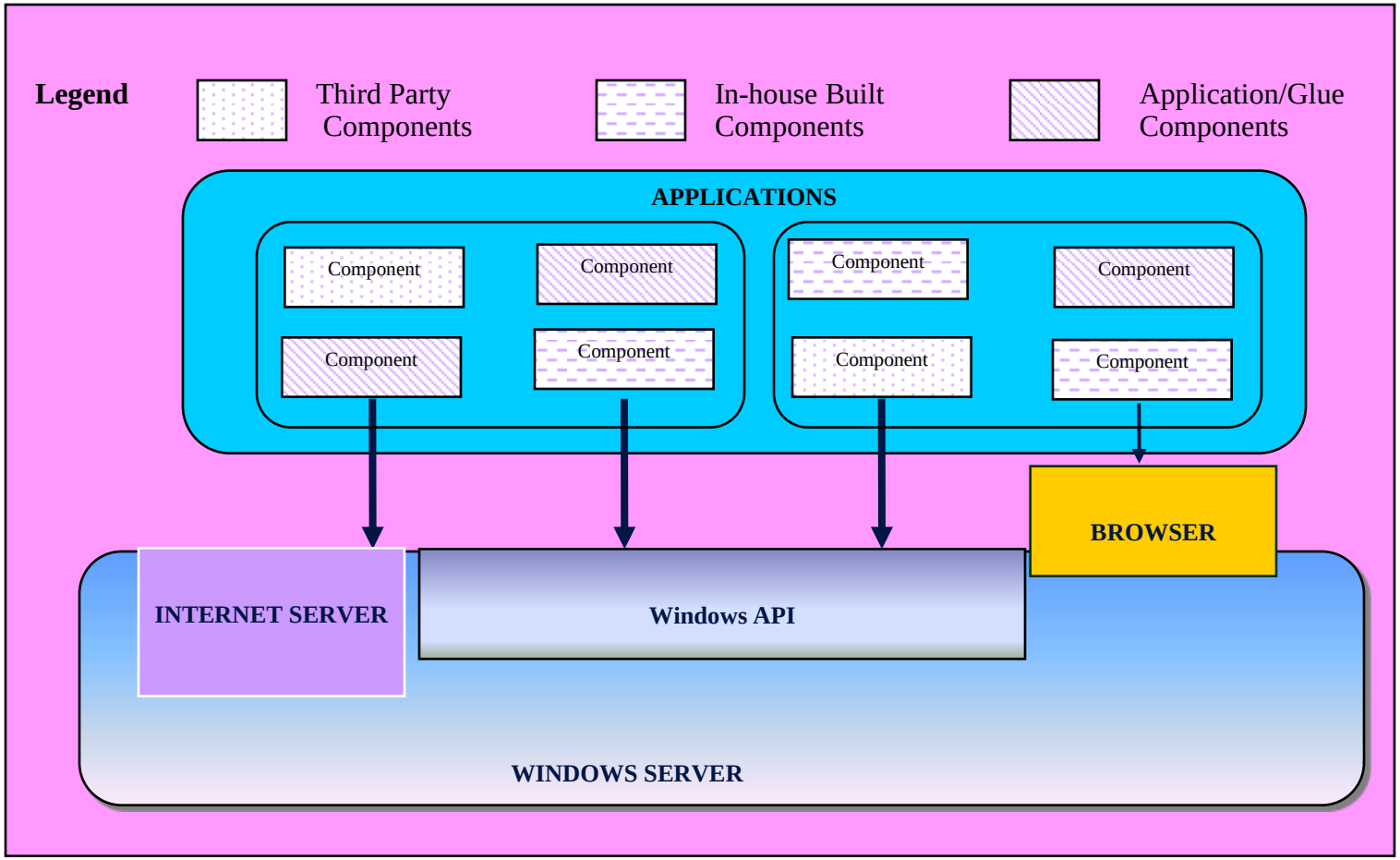
Topics

Session # 2

- (Source / Executable) Code Based Approach to Test Plan Automation
  - Selection of Regression Test Cases
    - Current Practices and their Limitations
    - Selection through Static Analysis of Source Code
    - Selection through Dynamic Analysis of Executable Code
    - Case Study

Wrap-up (15 minutes)

Component Based Software System



Regression Testing Fundamentals

- What is regression testing?
  - Regression testing involves selective re-testing of an application or component to verify that changes have not caused any unintended affects and that the application or component still complies with its specified requirements
  - Provide sufficient confidence that the changes were correct

- When does Regression testing come into use?
  - Product upgrades / Application enhancements / COTS upgrades
  - Releasing service patches of products
  - Fixing bugs in existing system / Maintenance / Production support

Regression Testing Strategies

- Retest_all strategy
  - Rerun all tests of system test suite
  - Expensive both in time and resources
  - inefficient
- Selective re-test strategy
  - Execute sub-set of test cases that creates sufficient confidence in modified software behavior
  - Cost effective
- How is selective regression testing done?
  - **Assess** the impact of change to the software
  - **Gather** test cases/test suites
  - **Run** the identified tests to validate that the changes do not impact existing capabilities of the software

Process of Regression Testing

- P – program, P^1 – Modified program, T – Original Test Suite
- **Selecting** T^1 – a set of test cases to re-execute on P^1
- **Test** P^1 with T^1
- If necessary **create** T^{11} – a set of additional test cases for P^1
- **Test** P^1 with T^{11}
- **Create** T^{111} a new test suite for P^1 from T, T^1 and T^{11}

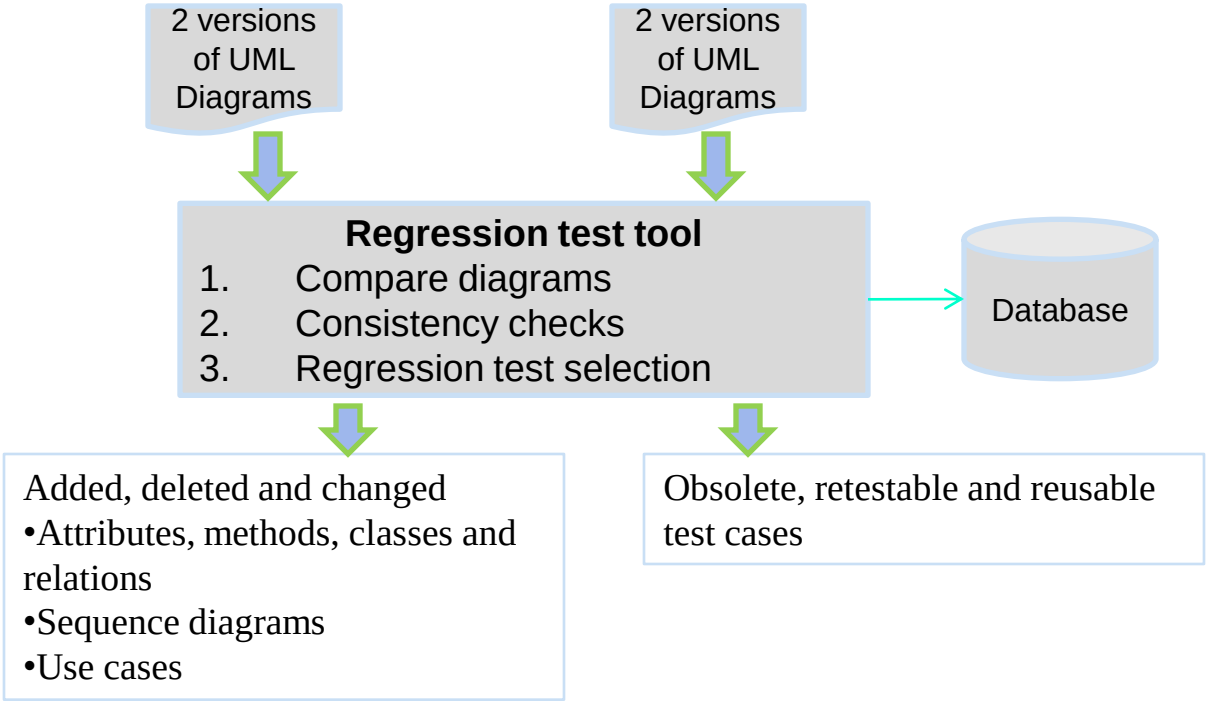
- Regression test suite = $T^1 + T^{11}$
- Updated system test suite $T^{111} = T\text{-}T^{(obsolete)} + T^{11}$
- T^1 = Retestable test cases
- $T – T^1 – T^{(Obsolete)}$ = Reusable test cases

Regression Tests Selection Techniques

- Software specification based techniques
  - changes made to requirements
- Model based techniques
  - Specification and design models
  - Availability of models with complete details
- Code based techniques
  - source code and binaries are available
  - Test cases are very precise
  - Programming languages specific tools/techniques
  - Static techniques
    - Analyze the information made available on changes
  - Dynamic techniques
    - application is executed according to its intended use, more comprehensive in selecting tests

Model based Techniques

- Changes between two versions of the UML designs
  - Class, sequence and use case diagrams
  - Detect the sets of added, changed and deleted
    - Attributes, methods, relationships and classes



Code based Techniques

- Coverage techniques – locate affected or modified components and select tests that exercise those components
- Minimization techniques – select minimal sets of tests through affected or modified components
- Safe techniques – Includes every test in system test suite that can expose faults in modified software

Control Flow Graph based Regression Test Selection Techniques

- Source code analysis – using control flow graph
- Runtime behavioral analysis
  - Useful when source code is not available
- Interactions are at three levels
  - Component level interactions
  - Method level interactions
  - Statement level interactions
    - Too costly

Control flow graphs are constructed at statement or component or method levels

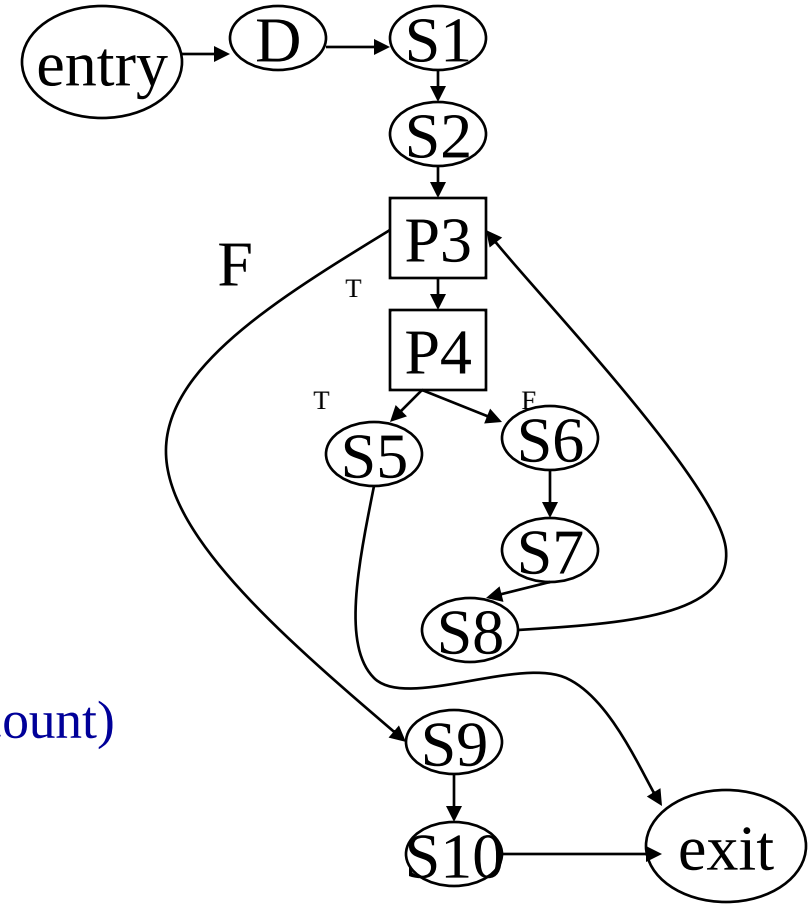
Source Code Analysis

- Control flow graph (CFG) is constructed
  - For both versions of a procedure or program
  - Flow graph is constructed for executed statements
- A unique mapping between execution statements and nodes of CFG
- CFG nodes are labeled by the text of the statements to which they correspond
- A pair-wise comparison of the nodes is performed
- When labels are not lexicographically equivalent the corresponding test is recommended

- Algorithm is safe – select every test that expose faults in the modified programs

Source Code based Algorithm – An example (1/3)

```
S1:   count = 0
S2:   fread (fileptrn)
P3:   while (not EOF) do
P4:   if (n<0)
S5:   Return(error)
      Else
S6:   Numarray[count] = n
S7:   Count++
      Endif
S8:   Fread(fileptr,n)
      Endwhile
S9:   Avg=calcavg(numarray,count)
S10:  Return(avg)
```



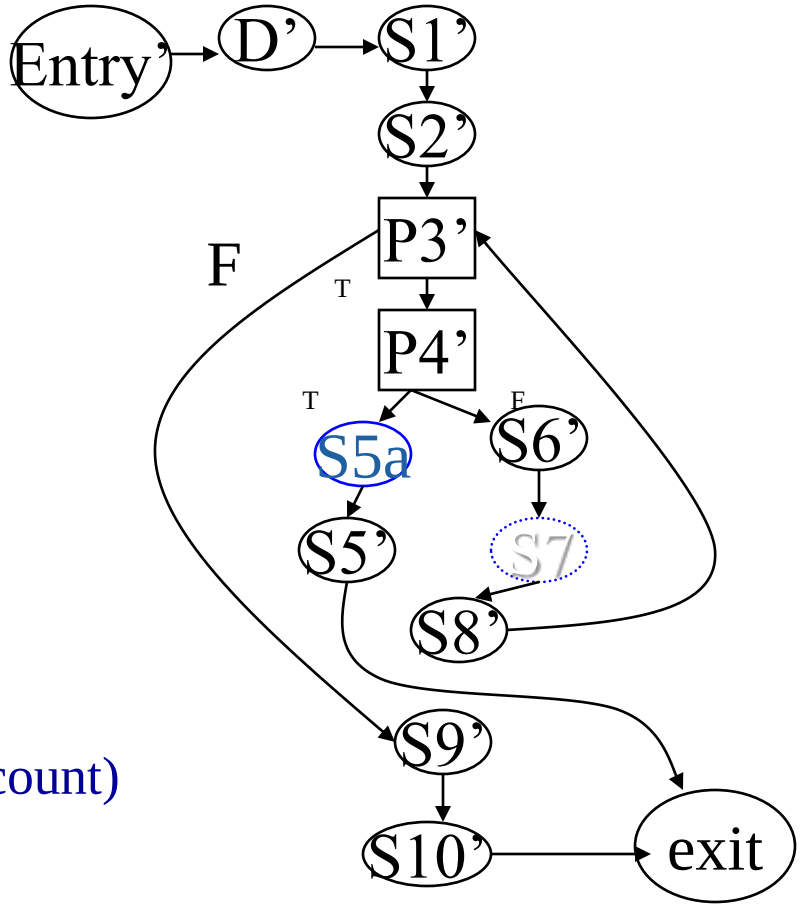
Source Code based Algorithm – An example (2/3)

- Test information

Test	Type	Output	Edges traversed
t1	Empty file	0	(entry,D), (D,S1), (S1,S2), (S2,P3), (P3,S9), (S9,S10), (S10, exit)
t2	-1	Error	(entry,D), (D,S1), (S1,S2), (S2,P3), (P3,P4), (P4,S5), (S5,exit)
t3	1 2 3	2	(entry,D), (D,S1), (S1,S2), (S2,P3), (P3,P4), (P4,S6), (S6,S7), (S7,S8), (S8,P3), (P3,S9), (S9,S10), (S10,exit)

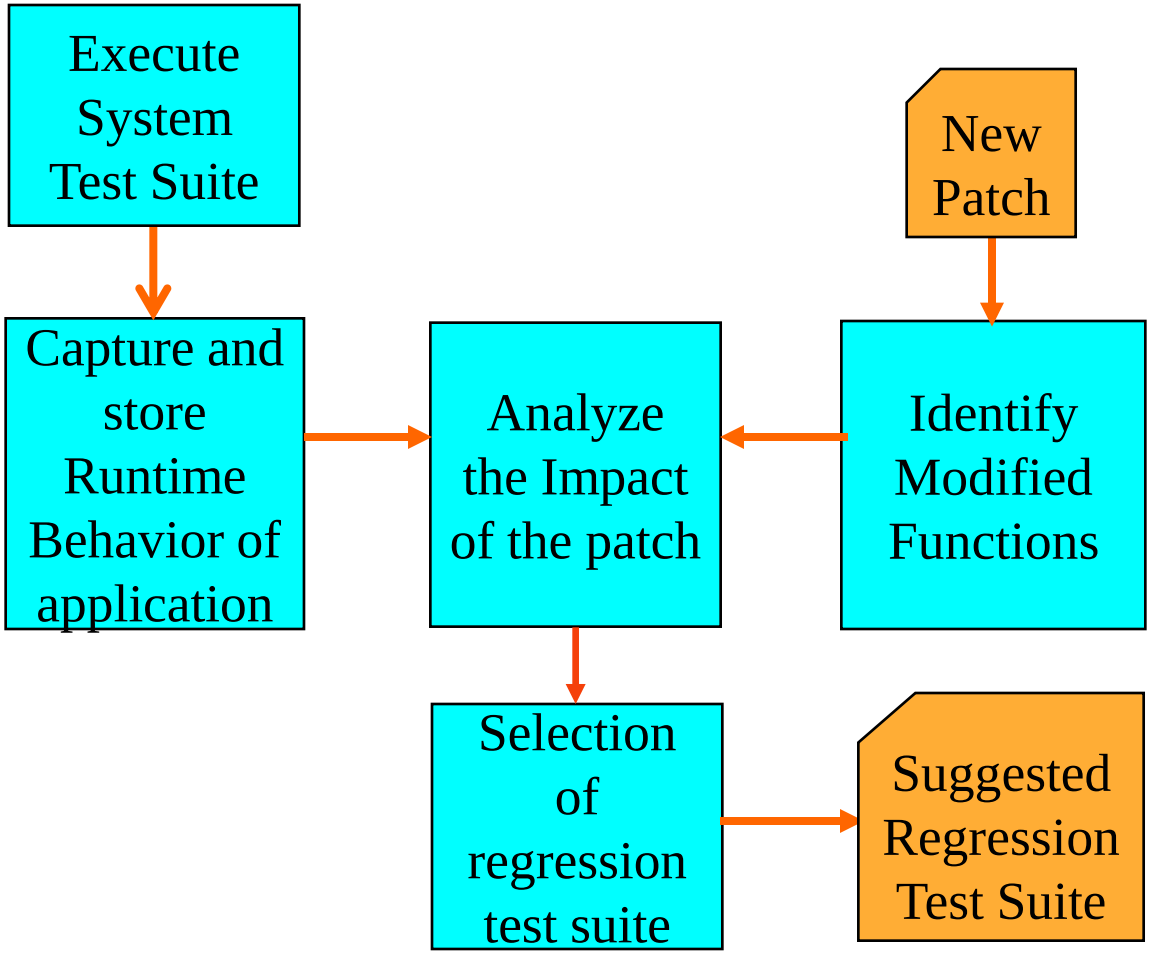
Source Code based Algorithm – An example (3/3)

```
S1':    count = 0
S2':    fread (fileptrn)
P3':    while (not EOF) do
P4':    if (n<0)
S5a     Print("Bad input")
S5':    Return(error)
        Else
S6':    Numarray[count] = n
S7
        Endif
S8':    Fread(fileptr,n)
        Endwhile
S9':    Avg=calcavg(numarray,count)
S10':   Return(avg)
```

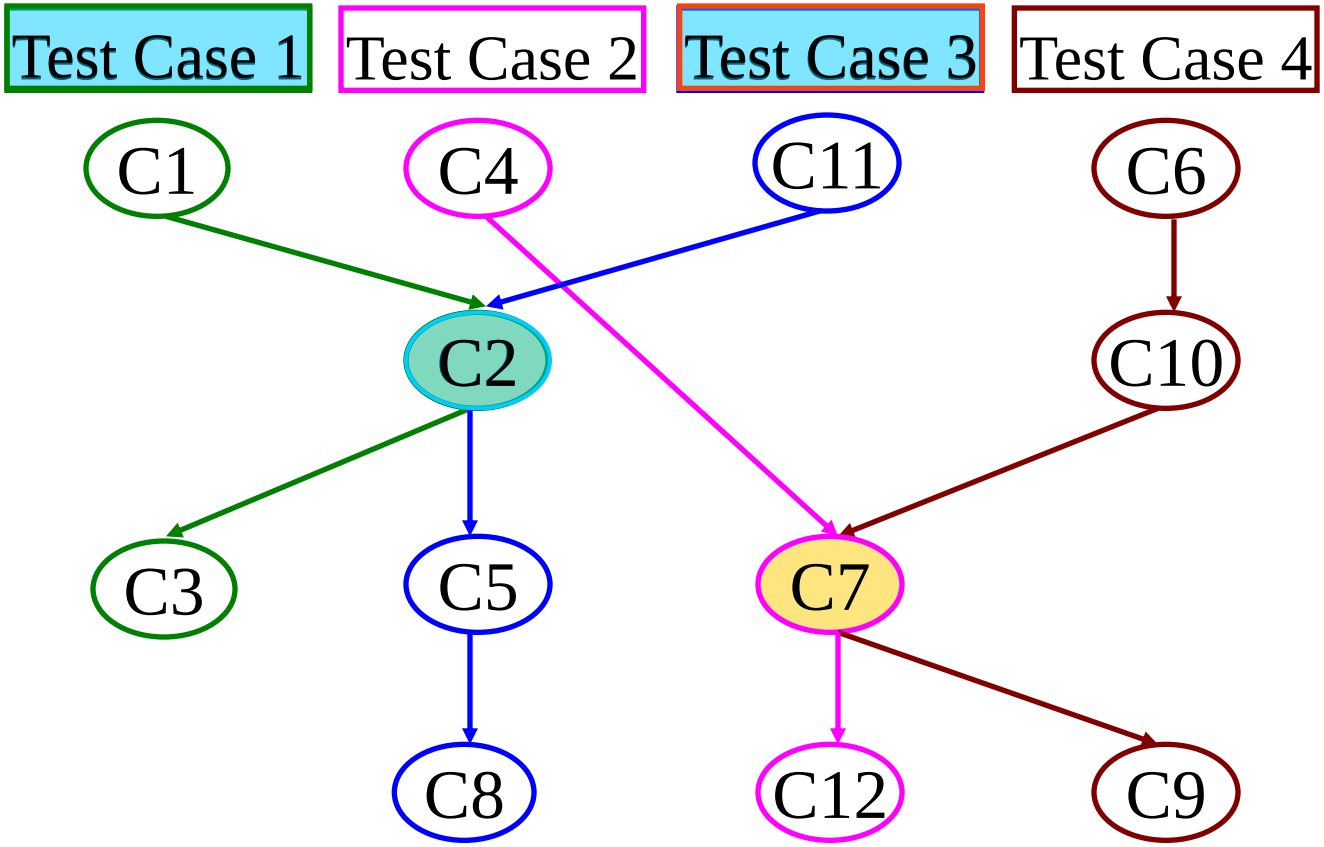


Runtime behavior analysis based approach: The Key-Idea

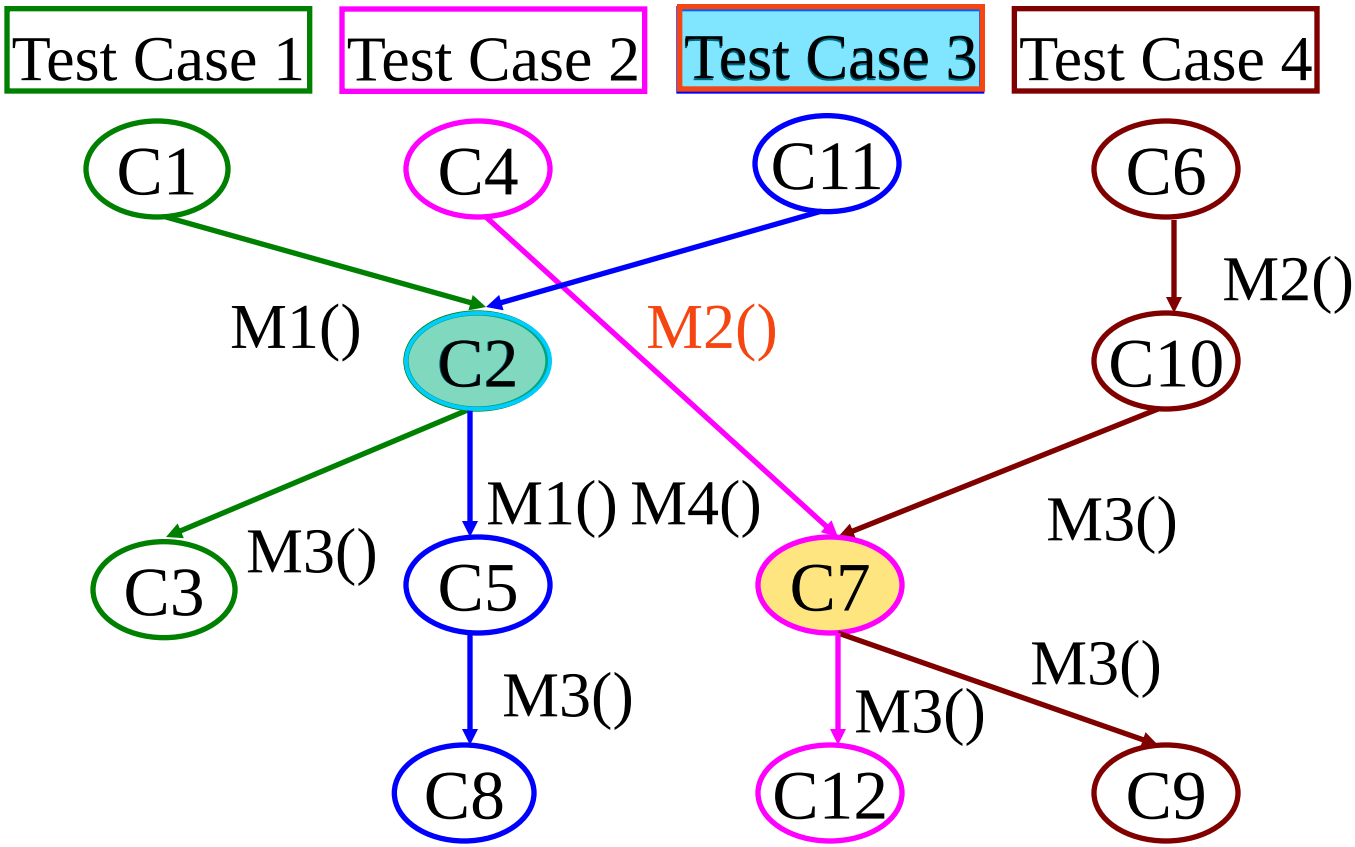
Capture and Analyze run-time behavior of the application



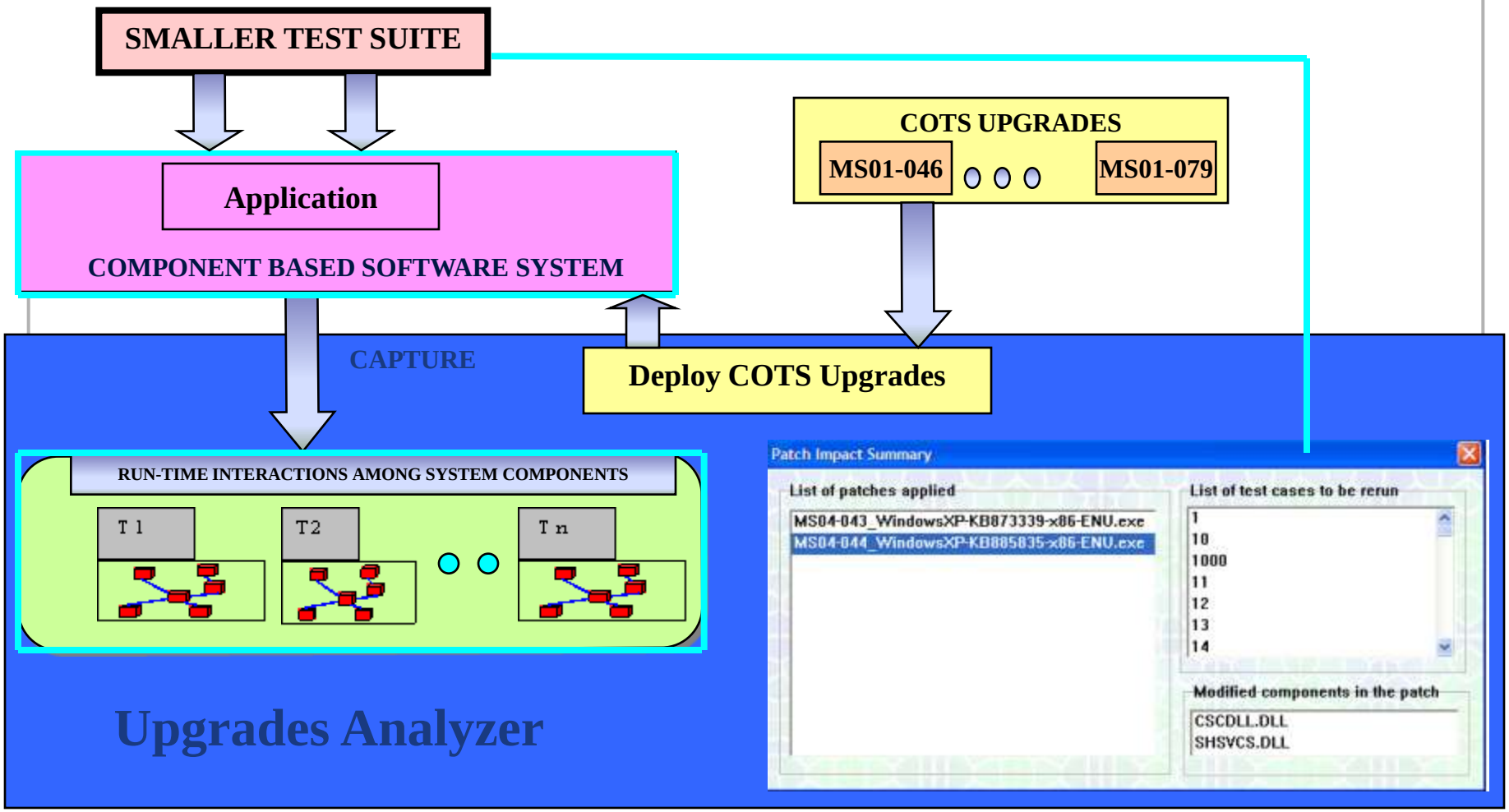
Test Case Selection based on Interaction among Components



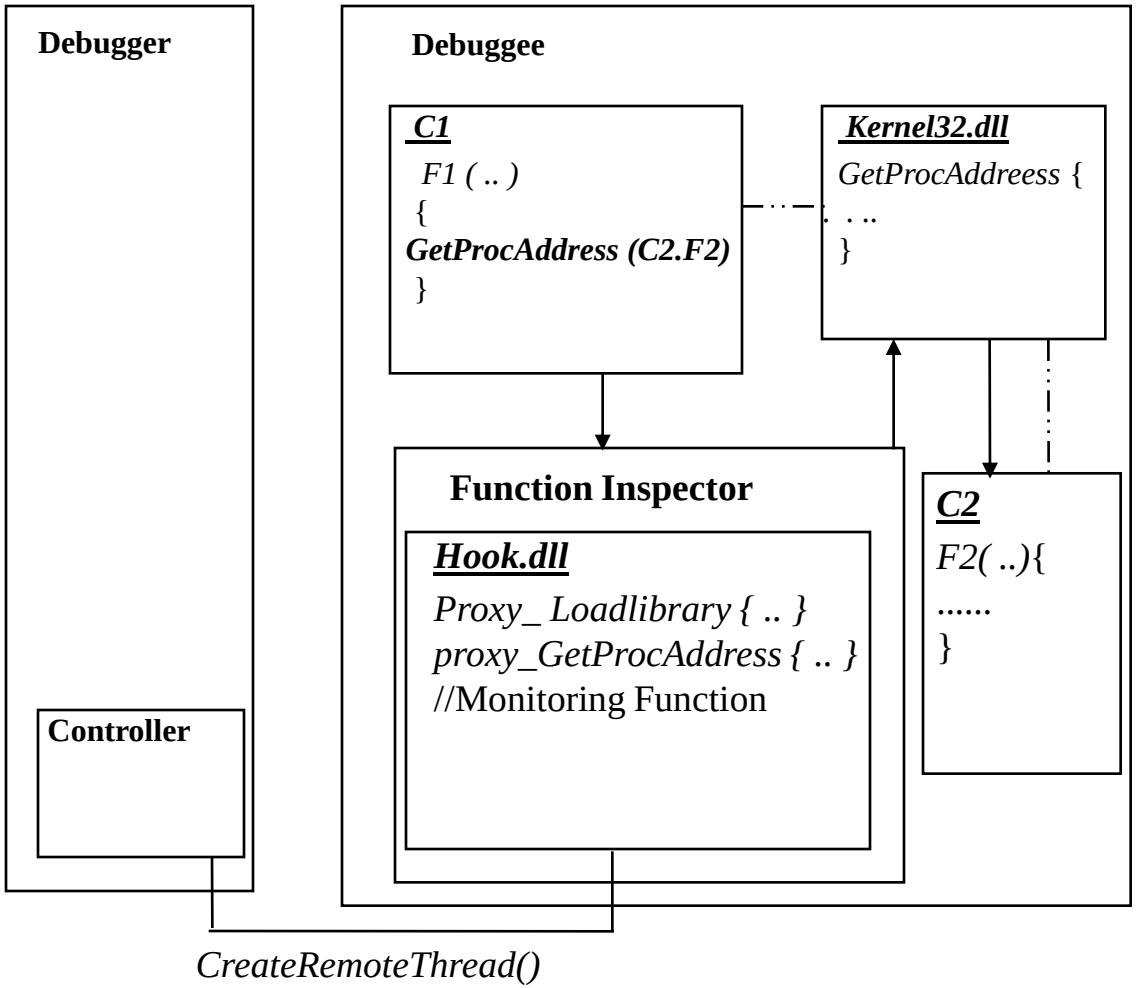
Test Case Selection based on Affected Methods



Automated Test Framework for COTS Components



A Procedure to Capture Runtime behavior of a Process

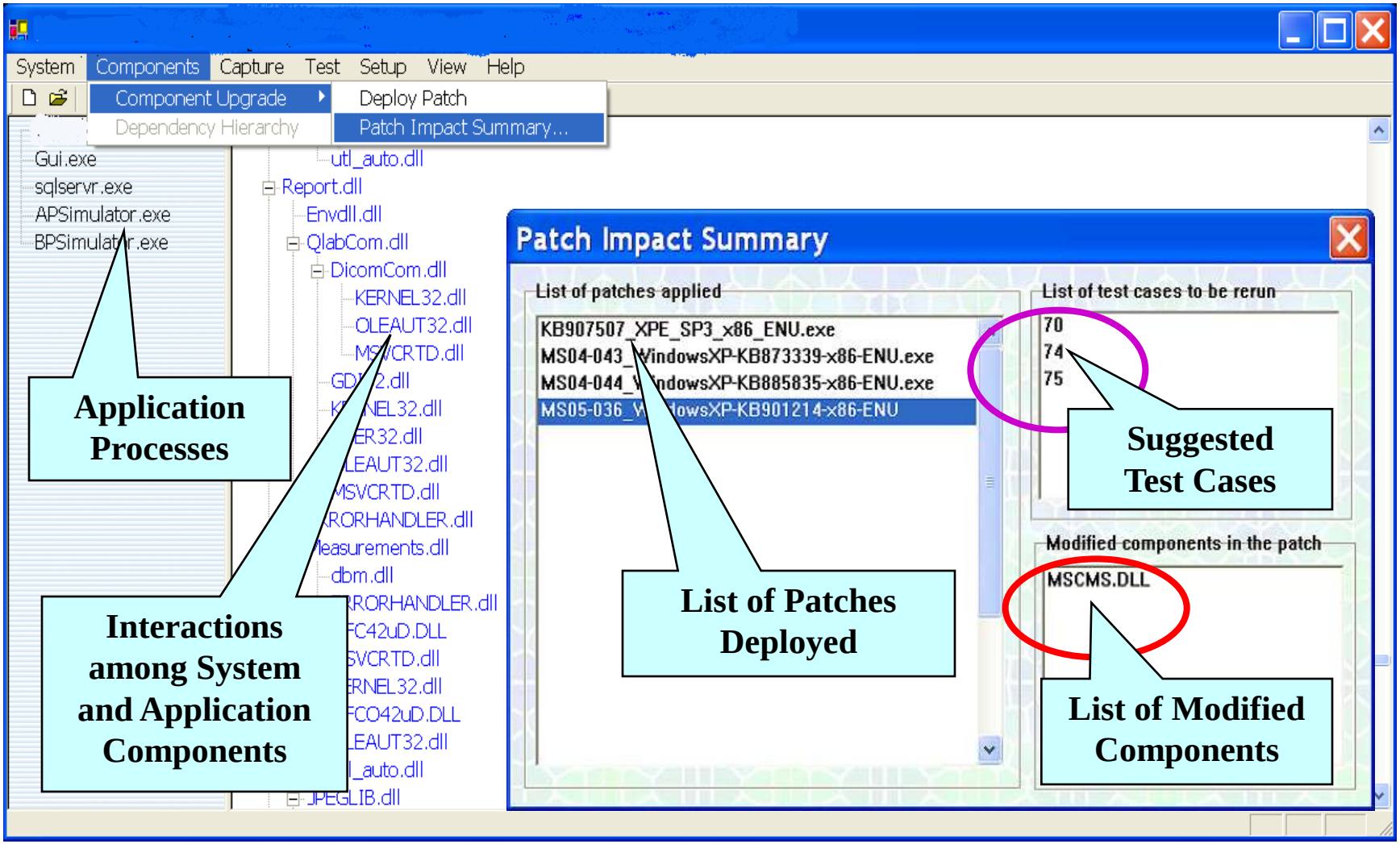


Identifying Altered Functions

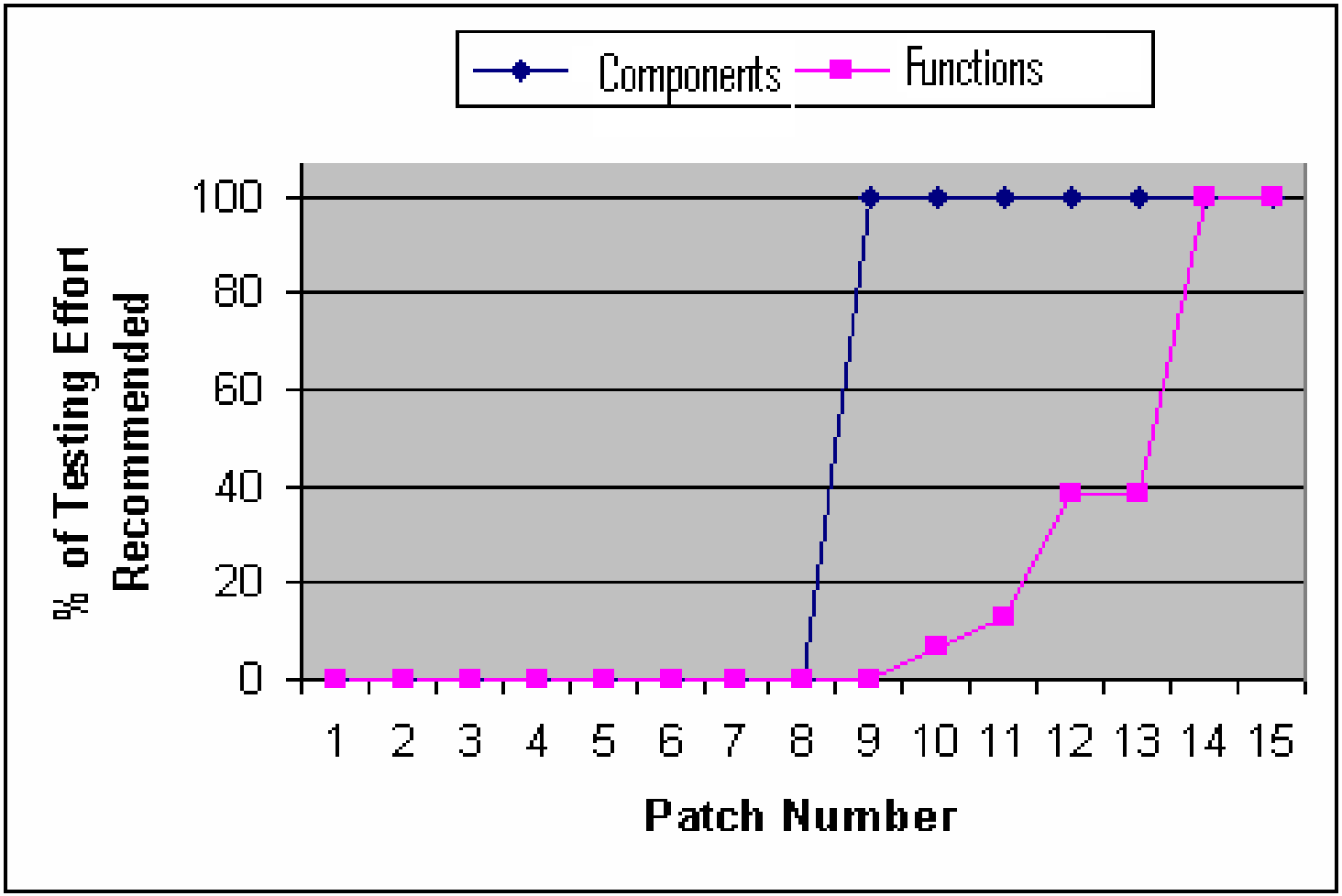
- DLL and EXE are reverse engineered into assembly language using PE Explorer

- Resolved the following changes
  - Change in names
  - Branch inversion
  - Reordering of instructions
  - Code movement due to optimizations
  - Changes to the callee functions code

Components Upgrades Analyzer - A Snapshot



Comparison of Results



Case Study - Results (1/2)

Tested on 47 patches of Windows XP and IE

- 21 out of these 47 do not require regression test
- Critical – 10 out of 27 do not require regression test
- Important – 8 out of 16 do not require regression test
- Moderate – 3 out of 4 doesn't require regression test

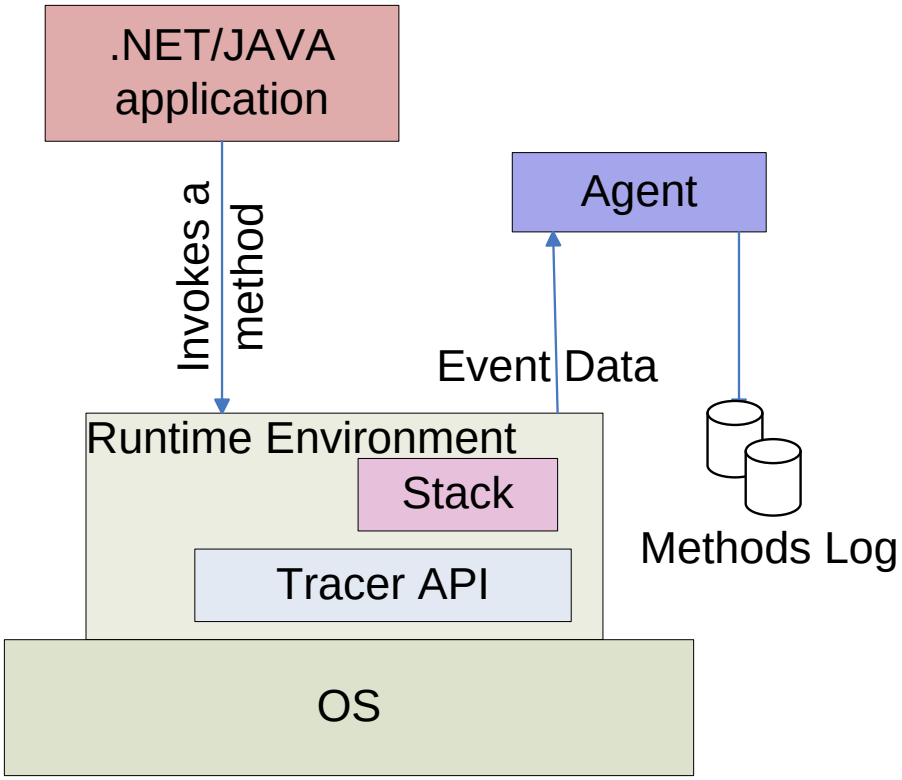
- No new or deprecated components are found except for the major releases of SP1 of IE (3) and XP (25)

Case Study - Results (2/2)

Modified Components in CIG	No. of Patches	Time taken in general /patch	Time taken with our approach /patch
None	21	6 man days	~one hour
Partial	1	6 man days	~half day
Critical component	25	6 man days	API testing is proposed

Process of Capturing Methods calls at Runtime

1. **Setup**
2. **Register for events**
3. **Capture events**



Change Impact Analysis

- Identify affected methods in all components
- Reverse engineer binaries into IL or Assembly objects
- Identify syntactic changes carried to method and class IL objects
  - Compares two versions of a program for changed methods
- Identify all affected methods by propagating syntactic changes
  - Semantic analysis
    - Dynamic binding, exceptions etc
- Change impact analysis is carried out for each release/upgrade

Semantic Analysis

- Changes due to dynamic binding
- Changes to an inheritance tree
- Changes to exception handling
- Changes to implementation of exceptions

Changes due to Dynamic Binding

Old version

```
Class A{
    int sum;
    virtual int calc() {
      return sum;
    }}
Class B:A{
     int calc() {
       return A::sum/3;
    }}
Class D { …
void int func1(B obj)
{
   return obj.calc();
}}
```

New version

```
Class A{
    int sum;
    virtual int calc() {
      return sum*sum;
    }}
 Class B:A{
     int calc() {
       return A::sum/3;
    }}
Class D{ …
void int func1(B obj)
{
   return obj.calc();
}}
```

Changes to Inheritance Tree

Old version

```
Class A: System.Array
{      static int i = 0;
     Void int objVal()
      {
          return i;
      }}
Class B:A
{…
}
```

New version

```
Class A
{      static int i =1;
     Void int objVal()
      {
          return i;
      }}
Class B:A
{…
}
```

Changes to Exceptions Handling

Old version

```
void func1()
{
  try{…}
  catch(e1){…}
  catch(e2){…}
}
void func2()
{
  try{…}
  catch(e1){return;}
  catch(e2){…}
}
```

New version

```
void func1()
{
  try{…}
  catch(e1){…}
}
void func2()
{
  try{…}
  catch(e1){func2();}
  catch(e2){…}
}
```

Changes to Implementation of Exceptions

Old version

```
Class E: Exception
{
 string msg = “Error”;
}
void doSomething()
{   ….
   if(null == something)
      throw E;
}
```

New version

```
Class E: Exception
{
 string msg = “Alarm!”
}
void doSomething()
{   ….
   if(null == something)
      throw E;
}
```

InARTS – Automatic Testing of Applications

- Captures runtime interactions among components and methods
- Builds the base MIG for all system test cases
- Identify all affected methods
- Automatic selection of test cases to rerun

Case study - NunitForms

Alpha 2 is the base version and 50 test cases were created

Upgrade No.	# Test cases recommended	%of test effort saved
Alpha 3	11	78%
Alpha 4	25	50%
Alpha 5	11	78%

Selective References

1. Ravi Gorthi et al., “Model-Based Automated Test Case Generation”, SETLabs Briefings, Vol 6, No 1, 2008, pp 39 – 46. http://www.infosys.com/research/publications/toc-software-validation.asp
2. Ravi Gorthi, Anjaneyulu Pasala, et al., Specification-based Approach to Select Regression Test Suite to Validate Changed Software, accepted for publication at the 15th Asia-Pacific Software Engineering Conference (APSEC - 2008), Dec. 3-5, 2008.
3. Anjaneyulu Pasala, Ravi Gorthi, et al., “Selection of regression test suite to validate software applications upon deployment of upgrades”, 19th Australian Software Engineering conference (ASWEC – 2008), 25-28 March 2008, pp 130-138.
4. Anjaneyulu P, et al. “An Approach for Test Suite Selection to Validate Applications on Deployment of COTS Upgrades”, *12th IEEE Asia Pacific Software Engineering Conference*, Dec. 2005, pp 401–407.
5. Anjaneyulu P, et al “An Approach Based on Modeling Dynamic Behavior of the System to Assess the Impact of COTS Upgrades”, 13th IEEE Asia-Pacific Software Engineering Conference (APSEC), Dec. 2006, pp 19-26.
6. Anjaneyulu Pasala, Ravi Gorthi et al., “How to Select Regression Tests to Validate Applications upon Deployment of Upgrades”, SETLabs Briefings, Vol 6, No1, 2008, pages 55 – 63.
7. Anjaneyulu Pasala et al., “On the validation of API execution-sequence to assess the correctness of application upon COTS upgrades deployment”, 6th IEEE Internation conference on COTS based software systems, (ICCBSS - 2007), February-March 2007, at Banff, Canada.
8. Peter Zielczynski, “Traceability from Use Cases to Test Cases”, http://www.ibm.com/developerworks/rational/library/04/r-3217/

Selective References

9. Jim Heumann, “Generating Test Cases from Use Cases”, the Rational Edge, June 2001
10. Clementine Nebut, et al, “Automatic Test Generation: A Use Case Driven Approach”, IEEE Transactions on Software Engineering, vol 32, n 3, March 2006
11. Chen T Y, et al, “A Choice Relation Framework for Supporting Category-Partition Test Case Generation”, IEEE Transactions on Software Engineering, vol 29, no 7, July 2003
12. Lionel Briand and Yvan Labiche, “A UML-Based Approach to System Testing”, Carleton University TR SCE-01-01-Version 4, June 2002
13. Wang Linzhang, et al, “Generating Test Cases from UML Activity Diagram based on Gray-Box Method”, Proceedings of APSEC 2004
14. Chen Mingsong, et al, “Automatic Test Case Generation for UML Activity Diagrams”, AST 06, Shanghai, China, May 2006
15. Paul Gerrard, “Testing Requirements”, Systeme Evolutif, http://www.evolutif.co.uk/testReqs/TESTREQS.html
16. Thomas Ostrand and Marc Balcer, “The Category Partition Method for Specifying and Generating Functional Tests”, Communications of the ACM, vol 31, no 6, June 1988
17. Aynur Abdurazik and Jeff Offutt, “Generating Test Cases from UML Specifications”, ISE-TR-99-09, Information and Software Engineering, George Mason University, May, 1999
18. James Clarke, “Automated Test Generation from a Behavioral Model”, Software Quality Week Conference, May, 1998

Selective References

19. James Clarke, “Automated Test Generation from a Behavioral Model”, Software Quality Week Conference, May, 1998
20. Philippe Chevalley and Pascale Thevenod-Fosse, “Automated Generation of Statistical Test Cases from UML State Diagrams”, Proceedings of COMPSAC 2001
21. Supaporn Kansomkeat and Wanchai Rivepiboon, “Automated-Generating Test Case Using UML Statechart Diagrams”, Proceedings of SAICSIT, 2003
22. Stefania Gnesi et al, “Formal Test-Case Generation for UML Statecharts”, Proceedings of 9th IEEE Int Conf on Engineering Complex Computer Systems, 2004
23. A-Valdivino Santiago et al, “A Practical Approach for Automated Test Case Generation using Statecharts”, Proceedings of COMPSAC 2006
24. Jeff Offutt et al, “Generating test data from state-based specifications”, in Software Testing, Verification and Reliability, John Wiley & Sons Ltd, 2003
25. Stefan Hildenbrand, “Generation of Test Cases”, Semester Thesis, Summer of 2005, ETH Zurich
26. Basanieri F and Bertolino A, “A Practical Approach to UML-based derivation of integration tests”, http://www1.isti.cnr.it/~antonia/publications2002.html
27. Richard DeMillo and Jefferson Offutt, “Constraint-Based Automatic Test Data Generation”, IEEE Transactions on Software Engineering, vol 17, no 9, Sep 1991
28. Bogdan Korel, “Automated Software Test Data Generation”, IEEE Transactions on Software Engineering, vol 16, no 8, August 1990

Selective References

29. Gregg Rothermel et al., “a safe, efficient regression test selection technique”, ACM transactions on software engineering and methodology, Vol. 6, No. 2, April 1997, pp 173 – 210.
30. Greg Rothermel, et al., “Analyzing Regression Test Selection Techniques”, IEEE Transactions on Software Engineering, Vol. 22, No. 8, August 1996, pp 529-551.
31. Zheng J, et al. “Applying Regression Test Selection for COTS-based Applications”, International Conference on Software Engineering (ICSE), May 20-28, 2006, pp 512-521.
32. Briand, L.C.; Labiche, Y.; Soccar, G., “Automating impact analysis and regression test selection based on UML designs”, International Conference on Software Maintenance, 3-6 October 2002, pp 252 – 261.