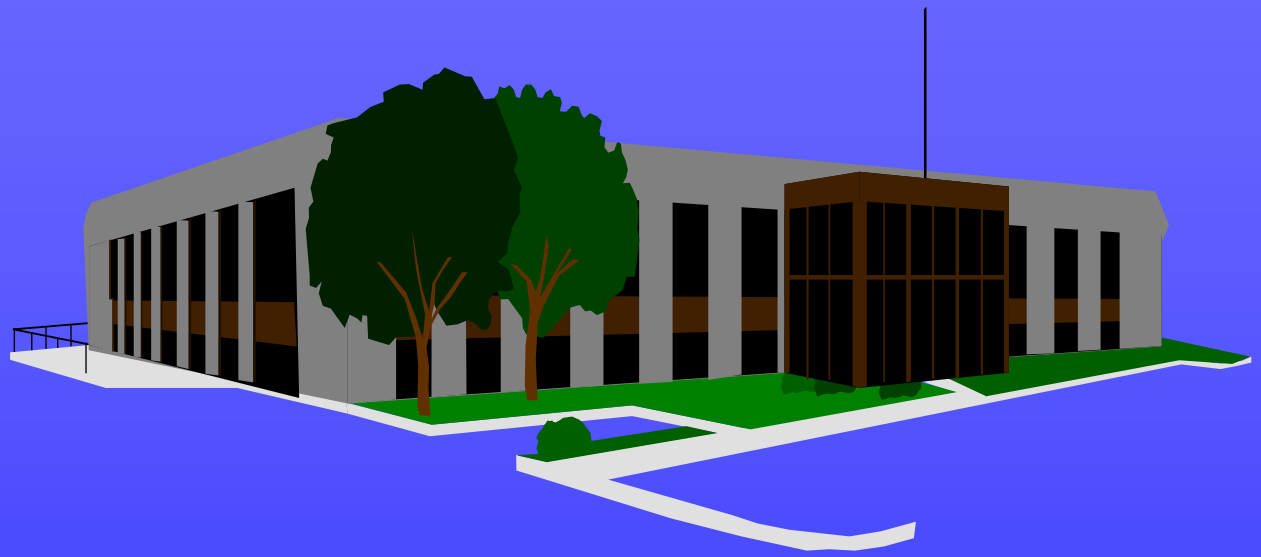
A Test Scenario Approach to Support Change Impact Analysis



Dr. Suhaimi Ibrahim
Centre For Adv. Soft. Engineering
UTM, City Campus,
Kuala Lumpur

Outline

- What is Regression Testing?
- Selective Test Cases
- Needs of Concept Location
- Requirements Traceability
- What is Reconnaissance Approach?
- Instrumentation Process
- Compilation Process
- Test Scenarios
- Analyzing
- Case Study – OBA Project
- Results and Discussion
- CATIA Application
- Conclusion and Future Work

What is Regression Testing?

- Regression testing is a process of testing changes to software system to make sure that the existing software functionalities still work with the new changes.
- Regression is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.
- Common methods of regression testing include rerunning previously run tests and checking whether previously fixed faults have reemerged.

What is Regression Testing?

- Theoretically, after each fix, one must run the entire batch of test cases previously run against the system, to ensure that it has not been damaged in an obscure way. In practice, such regression testing must indeed approximate this theoretical idea, but it is very costly.

- Regression testing is based on the idea of reusing a test and acceptance standard, rather than forgetting about them once the test is successful. On each iteration of true regression testing, all existing, validated tests are run, and the new results are compared to the already achieved standards.

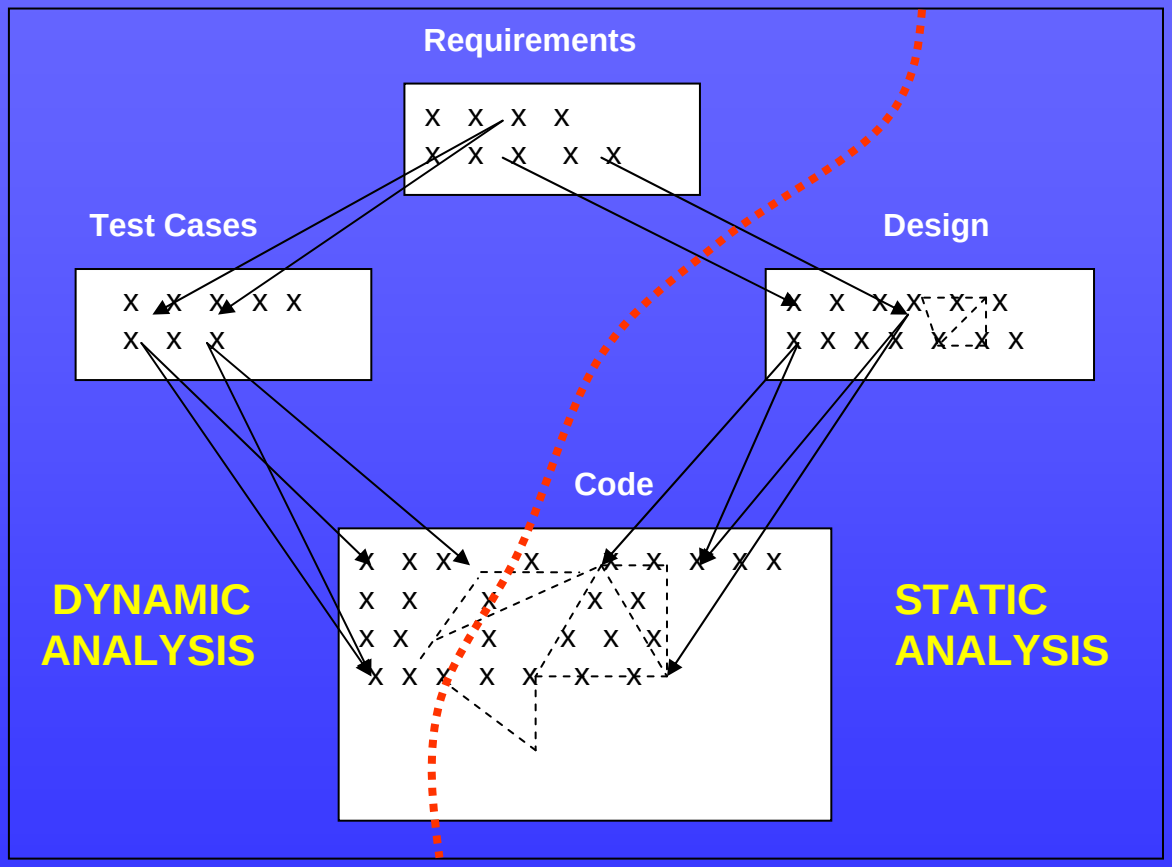
Selective Test Cases

- Rerunning the entire test cases is tedious and costly.
- Only affected test cases are useful to rerun the existing functionalities of the program.
- Selected test cases are crucial
- Selected test cases can reduce test cycles
- How to determine the selected test cases….

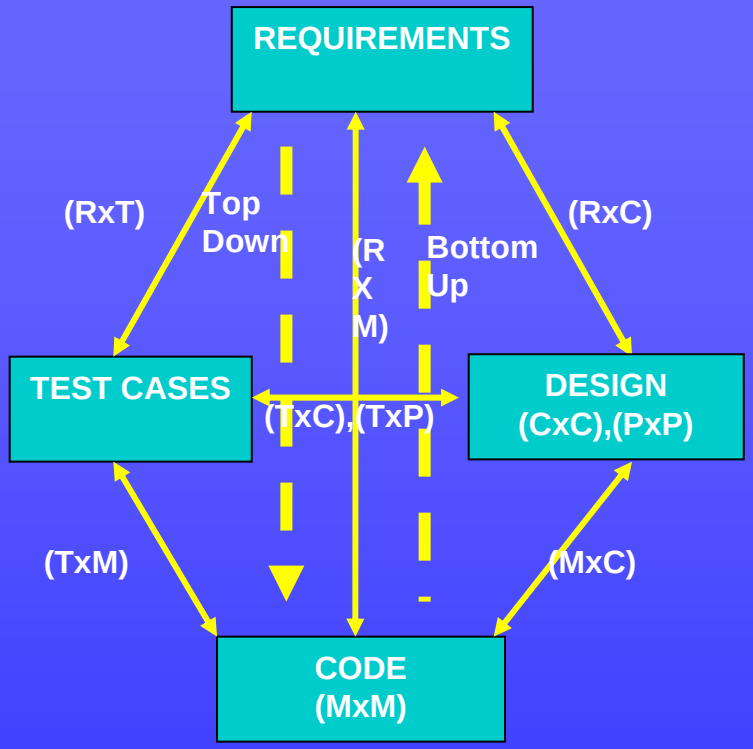
Need of Concept Location

- Concept location is a process of locating a feature in the system e.g. age, total income, order purchasing, credit card system, change request, requirements, test cases.
- For example, on each change request there is a need to identify the potential change impact prior to the actual change.
- CCB needs to consult developers for some change requests prior to decision making.
- Some kind of tool automation is required
- Existing tools/CASE tools more focus on software development, its integration and coordination.
- Apply reconnaissance techniques ……

Requirements Traceability



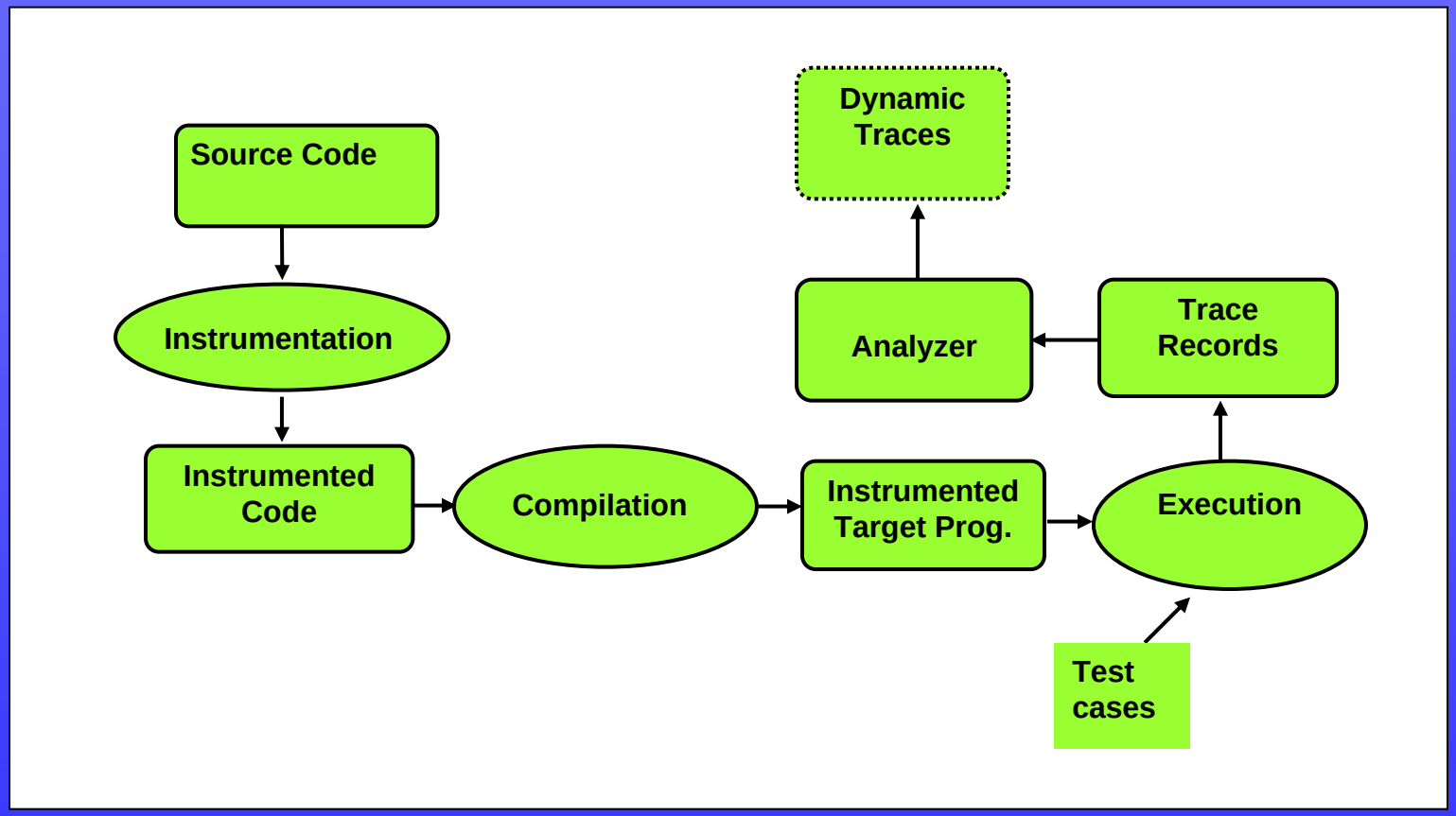
Requirements Traceability



What is Reconnaissance Approach?

- A dynamic analysis approach derived from test cases.
- Capture the impacted code
- Analyze the impacted code
- Create traceability between test cases and impacted modules
- Any change to a module can be made traceable to its corresponding test cases.

What is Reconnaissance Approach?



Instrumentation Process

- Instrumentation via our tool, *CodeMentor* to embed some markers into the original code.
  - Start of function/class - e switch will insert instrumentation at the entry point of a function or class, including main() function.
  - Return - r switch will insert instrumentation whenever it encounters an explicit or implicit RETURN.
  - Exit - x switch will insert instrumentation whenever it encounters an EXIT statement.
  - Condition - d switch will insert instrumentation whenever it encounters a decision statement.

Instrumented Code

```
/* FUNCTION PURPOSE : CCruiseInfo constructor                          */
/*                                                                     */
/* PARAMETER                : None                                     */
/*                                                                     */
/* RETURN VALUE             : None                                     */
/*********************************************************************/

CCruiseInfo::CCruiseInfo()
{
fputs("\n#TRACE# I am Inside Function *CCruiseInfo.CCruiseInfo()* Line=55",dst_file1);
fprintf(dst_file1,"^ %d ^",++count_exe);
            bActive = false;
            bAcceleration = false;
            bAccelerationPedal = false;
          if(countIFTrue1 > 0)
           {
fputs("\n#TRACE# If statement Line : 59",dst_file1);
              ++countIFTrue1;
           }
            bResume = false;
            bSuspend = false;
            dPrevCruiseSpeed = 0;
            throttle->controlThrottle(throttleCommand, throttlePosition, 0);
            displayMsg->displayCommonMsg("--------------------"); //display
            led->lightLed(ledShm,1,false);
            led->lightLed(ledShm,2,false);
            led->lightLed(ledShm,3,false);

fputs("\n#TRACE# END OF Function CCruiseInfo.CCruiseInfo() Line=67  ",dst_file1);
fprintf(dst_file1,"^ %d ^",++count_exe);
}

/*******************  END OF FUNCTION  ************************/
```

Entry Instrumentation

Decision (IF) Instrumentation

Return Instrumentation

Compilation Process

- To compile the instrumented code
- To generate the instrumented target program
- To make it ready for test scenarios

Test Scenarios

- Test scenarios involve program execution over test cases.
- Each requirement is characterized by one or more test cases.
- Each test case is run to generate the impacted code components.
- The impacted code components are saved for further analysis.

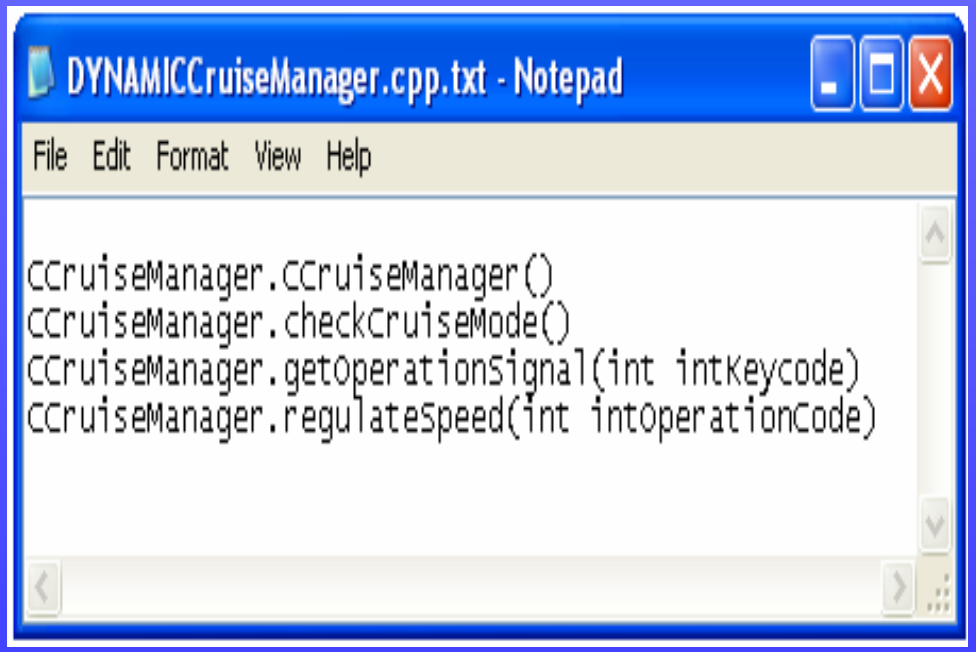
Snapshot of Trace Records

```
#TRACE# I am Inside Function *CCruiseInfo.CCruiseInfo()* Line=55^ 1 ^
#TRACE# END OF Function CCruiseInfo.CCruiseInfo() Line=67  ^ 2 ^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 3 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^ 4
^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 5 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^ 6
^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 7 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^ 8
^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 9 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^
10 ^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 11 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^
12 ^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 13 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^
14 ^
#TRACE# I am Inside Function *CCruiseInfo.setCruiseStatus(char operationState, bool
bStatus)* Line=133^ 15 ^
#TRACE# If statement Line : 135
#TRACE# If statement Line : 137
#TRACE# END OF Function CCruiseInfo.setCruiseStatus(char operationState, bool
bStatus) Line=182  ^ 16 ^
#TRACE# I am Inside Function *CCruiseInfo.isActive()* Line=99^ 17 ^
#TRACE# exit from Function CCruiseInfo.isActive() by return( ) statement Line=100 ^
18 ^
#TRACE# I am Inside Function *CCruiseInfo.setCruiseStatus(char operationState, bool
bStatus)* Line=133^ 19 ^
#TRACE# If statement Line : 135
#TRACE# END OF Function CCruiseInfo.setCruiseStatus(char operationState, bool
bStatus) Line=182  ^ 20 ^
```

Analyzer

- Trace the impacted components via a *TestAnalyzer*
- Remove some duplicate components.
- Produce the impacted components in terms of classes and methods

A Class And its Impacted Methods

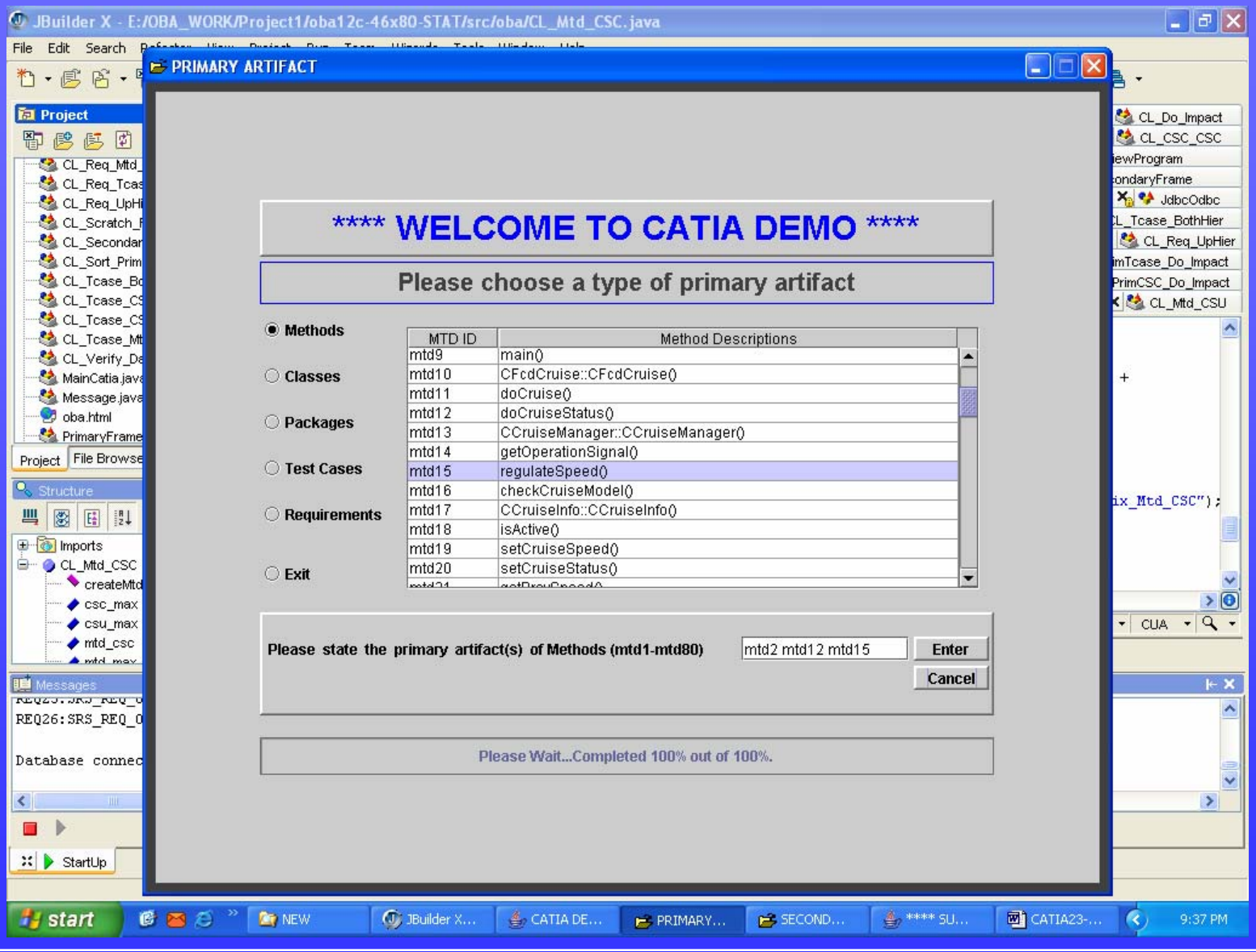


Case Study
- OBA Project

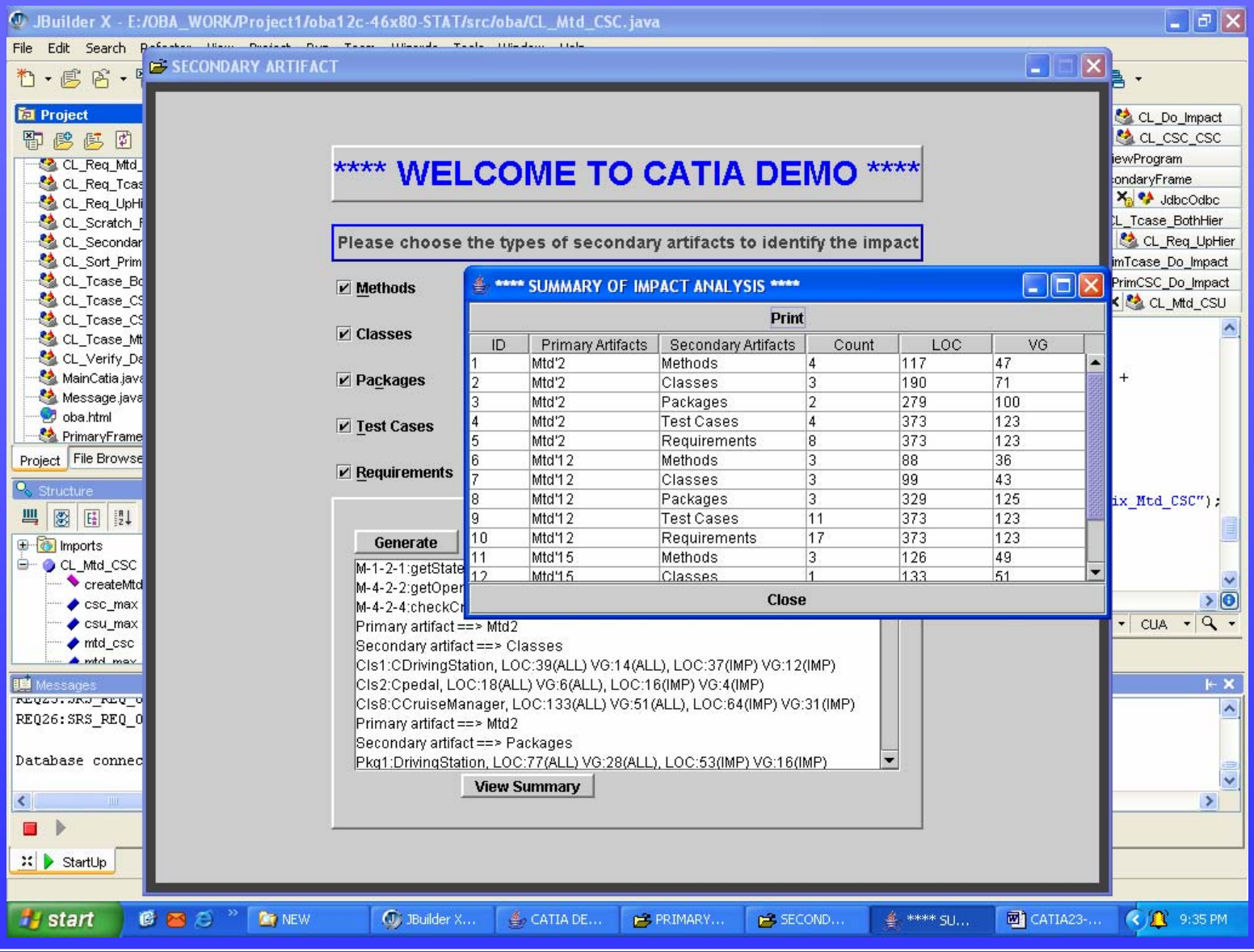
- The OBA project was built with a complete project management and documentation adhering to DOD standards, MIL-STD-498.
- Based on the UML specification and design standards
- Consists of 480 pages of system documentation
- Code size of 4k LOC
- We captured from the OBA project
  - 12 packages
  - 23 classes
  - 80 methods
  - 34 test cases
  - 46 requirements

	T1	T2	T3	T4	T5	T6	Tn
C1	1	1	1	1	1	1	..
C2	0	1	0	0	0	0	..
C3	1	1	1	1	1	1	..
C4	0	1	0	0	0	0	..
C5	1	1	1	1	1	1	..
C6	1	1	1	1	1	1	..
C7	1	1	1	0	0	0	..
C8	1	1	1	0	0	0	..
C9	1	1	1	0	0	0	..
C10	0	1	0	0	0	0	..
C11	1	1	1	1	1	1	..
C12	1	1	1	1	1	1	..
C13	1	1	0	0	0	0	..
C14	1	0	0	1	1	1	..
C15	1	0	0	0	1	1	..
C16	1	0	0	1	0	0	..
C17	1	0	0	0	1	1	..
C18	1	0	0	1	1	0	..
C19	1	0	0	0	0	1	..
C20	1	0	0	0	0	1	..
C21	1	0	0	0	0	1	..
C22	0	1	1	1	1	1	..
C23	1	1	1	1	1	1	..

CATIA Tool


JBuilder X - E:/OBA_WORK/Project1/oba12c-46x80-STAT/src/oba/CL_Mtd_CSC.java
PRIMARY ARTIFACT
**** WELCOME TO CATIA DEMO ****
Please choose a type of primary artifact
Methods
Classes
Packages
Test Cases
Requirements
Exit
MTD ID
Method Descriptions
mtd9 main()
mtd10 CFcdCruise::CFcdCruise()
mtd11 doCruise()
mtd12 doCruiseStatus()
mtd13 CCruiseManager::CCruiseManager()
mtd14 getOperationSignal()
mtd15 regulateSpeed()
mtd16 checkCruiseModel()
mtd17 CCruiseInfo::CCruiseInfo()
mtd18 isActive()
mtd19 setCruiseSpeed()
mtd20 setCruiseStatus()
Please state the primary artifact(s) of Methods (mtd1-mtd80)
mtd2 mtd12 mtd15
Enter
Cancel
Please Wait...Completed 100% out of 100%.

CATIA Tool


JBuilder X - E:/OBA_WORK/Project1/oba12c-46x80-STAT/src/oba/CL_Mtd_CSC.java
SECONDARY ARTIFACT
**** WELCOME TO CATIA DEMO ****
Please choose the types of secondary artifacts to identify the impact
Methods
Classes
Packages
Test Cases
Requirements
Generate
Primary artifact ==> Mtd2
Secondary artifact ==> Classes
Cls1:CDrivingStation, LOC:39(ALL) VG:14(ALL), LOC:37(IMP) VG:12(IMP)
Cls2:Cpedal, LOC:18(ALL) VG:6(ALL), LOC:16(IMP) VG:4(IMP)
Cls8:CCruiseManager, LOC:133(ALL) VG:51(ALL), LOC:64(IMP) VG:31(IMP)
Primary artifact ==> Mtd2
Secondary artifact ==> Packages
Pkg1:DrivingStation, LOC:77(ALL) VG:28(ALL), LOC:53(IMP) VG:16(IMP)
View Summary
**** SUMMARY OF IMPACT ANALYSIS ****
Print
ID | Primary Artifacts | Secondary Artifacts | Count | LOC | VG
1 | Mtd'2 | Methods | 4 | 117 | 47
2 | Mtd'2 | Classes | 3 | 190 | 71
3 | Mtd'2 | Packages | 2 | 279 | 100
4 | Mtd'2 | Test Cases | 4 | 373 | 123
5 | Mtd'2 | Requirements | 8 | 373 | 123
6 | Mtd'12 | Methods | 3 | 88 | 36
7 | Mtd'12 | Classes | 3 | 99 | 43
8 | Mtd'12 | Packages | 3 | 329 | 125
9 | Mtd'12 | Test Cases | 11 | 373 | 123
10 | Mtd'12 | Requirements | 17 | 373 | 123
11 | Mtd'15 | Methods | 3 | 126 | 49
12 | Mtd'15 | Classes | 1 | 133 | 51
Close
start
9:35 PM

Conclusion and Future Work

- Ability to dynamically trace components: test cases, impacted code/functionalities, requirements and design.
- Ability to support regression testing.
- Support requirement traceability for change impact analysis.
- Provide initial data to support other needs e.g. visualization, cost estimation, change schedule and program understanding.

End of presentation

Email: suhaimi@citycampus.utm.my