

Integration Testing

Wei-Tek Tsai
Department of Computer Science and
Engineering
Arizona State University
Tempe, AZ 85287

What is Integration Testing?

- A kind of testing that carrying out integration tests, which idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. Beyond that, if the program is composed of more than one process, they should be tested in pairs rather than all at once.
- Integration testing is critical to ensure the functional correctness of the integrated system. The objective of it is to discover interface errors among the elements being integrated
- Integration testing can be divided into two categories:
 - Incremental integration testing: incremental testing expands the set of integrated modules progressively.
 - Non-incremental integration testing: by no incremental testing, software modules are combined and tested randomly.

Integration Testing is Different Than System Testing

- System testing specifically goes after behaviors and bugs that are properties of the entire system as distinct from properties attributable to components (unless, of course, the component in question is the entire system). Examples of system testing issues: resource loss bugs, throughput bugs, performance, security, recovery, transaction synchronization bugs (often misnamed "timing bugs").

When To Carry Out The Integration Testing?

- Integration testing is carried out when integrating
 - Units or modules to form a component
 - Components to form a product
 - Products to form a system
- Integration testing is sometimes defined as *the level of testing between unit and system*

Integration Testing Strategies

- Instantaneous vs. incremental integration testing
 - Instantaneous integration testing is sometimes referred to as the big bang approach. Locating subtle errors can be very difficult after the bang
 - Incremental integration testing results in some additional overhead, but can significantly reduce error localization and correction time. The optimum incremental approach is inherently dependent on the individual project and the pros and cons of the various alternatives

Integration Testing Strategies (cont.)

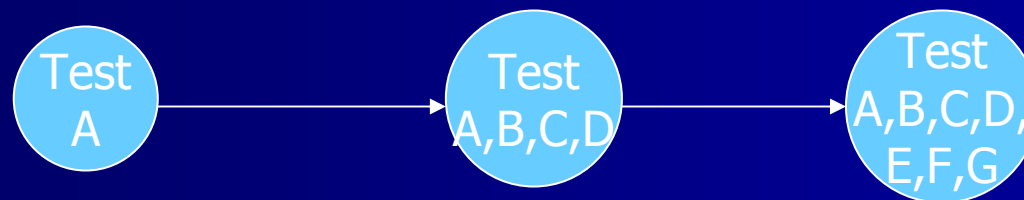
- When we are integrating a group of modules to form a component, the control structure of which will form a “calling hierarchy”
- In what order should the modules be integrated?
 - From the top module toward the bottom?
 - From the bottom module toward the top?
 - By function?
 - Critical or high-risk modules first?
 - By availability?

Top Down Integration Testing

- In top down integration testing
 - The high level control routines are tested first, possibly with the middle level control structures present only as stubs. Subprogram *stubs* are incomplete subprograms which are only present to allow the higher level control routines to be tested. Thus a menu driven program may have the major menu options initially only present as stubs, which merely announce that they have been successfully called, in order to allow the high level menu driver to be tested.

Top Down Integration Testing (cont.)

- The top level, usually one controlling component, is tested by itself. Then, all components called by the tested component(s) are combined and tested as a larger unit. This approach is reapplied until all components are incorporated.



Top Down Integration Testing (cont.)

- Top down testing can be proceed into
 - depth-first
 - Each module is tested in increasing detail, replacing more and more levels of detail with actual code rather than stubs
 - breadth-first
 - Refining all the modules at the same level of control throughout the application

Top Down Integration Testing (cont.)

- In practice a combination of the two techniques would be used.
 - At the initial stages all the modules might be only partly functional, possibly being implemented only to deal with non-erroneous data. These would be tested in breadth-first manner
 - Over a period of time each would be replaced with successive refinements which were closer to the full functionality. This allows depth-first testing of a module to be performed simultaneously with breadth-first testing of all the modules.

Top Down Integration Testing (cont.)

– Advantage

- Allows early verification of high-level behavior
- One driver (at most) is required
- Modules can be added one at a time with each step if desired
- Supports both “breadth first” and “depth first” approaches

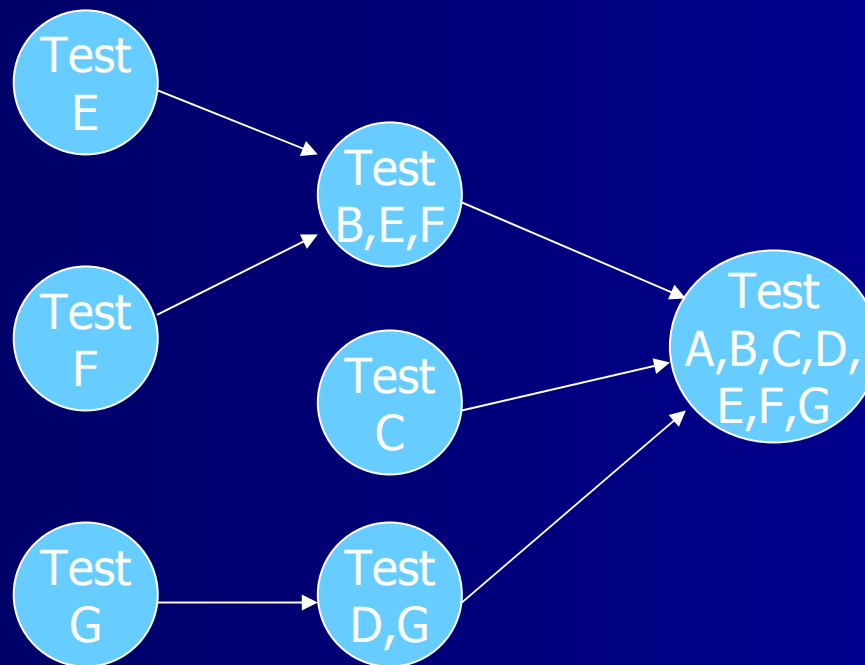
– Disadvantage

- Delays verification of low-level behavior
- Stubs are required for missing elements
- Test cases inputs and outputs may be difficult to formulate or interpret

Bottom Up Integration Testing

- The other major category of integration testing is bottom up integration testing where an individual module is tested from a test harness. Once a set of individual modules have been tested they are then combined into a collection of modules, known as builds, which are then tested by a second test harness. This process can continue until the build consists of the entire application.

Bottom Up Integration Testing (cont.)



Bottom Up Integration Testing (cont.)

■ Advantages

- Allows early verification of low-level behavior
- No stubs are required
- Easier to formulate input data for some subtrees; easier to interpret output data for others

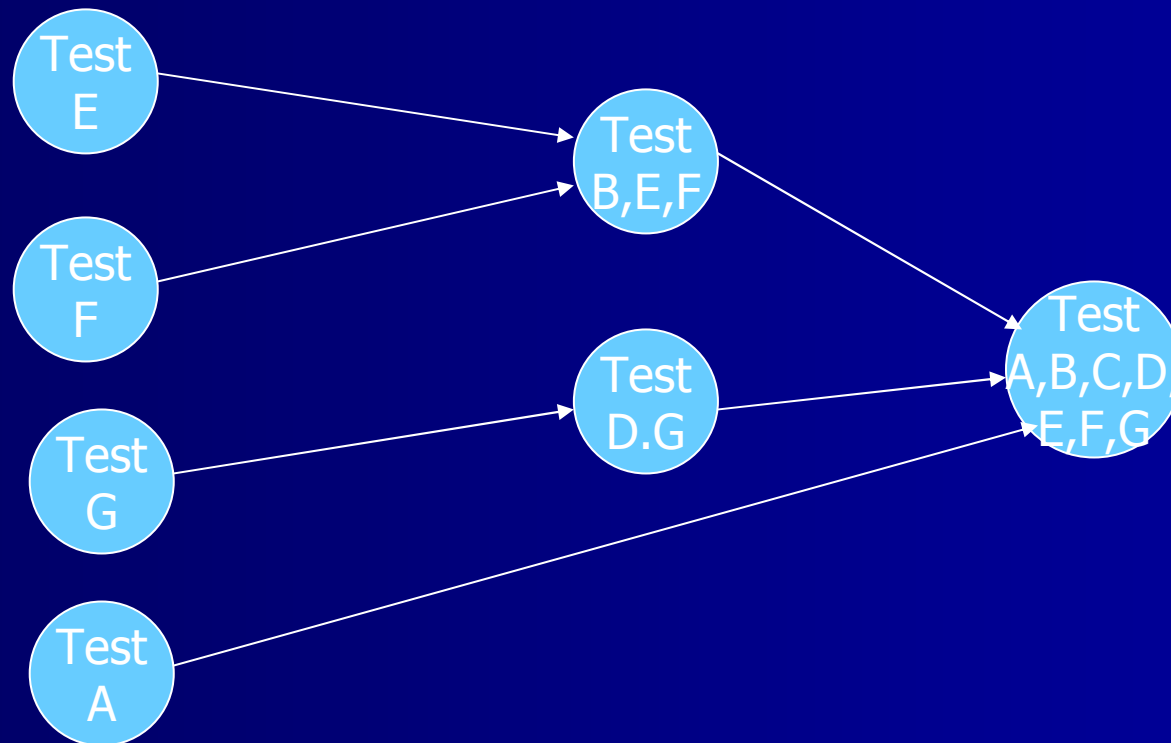
■ Disadvantages

- Delays verification of high-level behavior
- Drivers are required for missing elements
- As subtrees are combined, a large number of elements may be integrated at one time

Sandwich Integration Testing

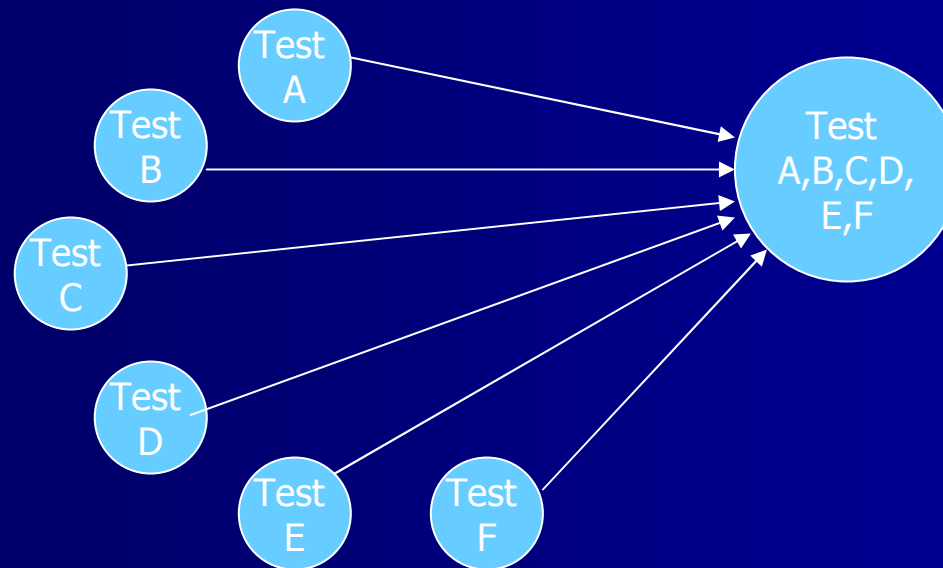
- In practice a combination of top-down and bottom-up testing would be used, which is defined as **sandwich testing** by Myers (1979). In a large software project being developed by a number of sub-teams, or a smaller project where different modules were being built by individuals. The sub-teams or individuals would conduct bottom-up testing of the modules which they were constructing before releasing them to an integration team which would assemble them together for top-down testing.

Sandwich Integration Testing (cont.)



Big-bang Integration Testing

- Definition by Myers (1979):
When all components are tested in isolation, it is tempting to mix them together as the final system and see if it works the first time.



Big-bang Integration Testing (cont.)

- Many programmers use the big-bang approach for small systems, but it is not practical for large ones.
- In fact, since big-bang testing has several disadvantages, it is not recommended for any system
 - It requires both stubs and drivers to test the independent components.
 - It is difficult to find the cause of any failure because all components are merged at once
 - Interface faults cannot be distinguished easily from other types of faults.

Comparison of Integration Strategies (Myers 1979)

	Top-down	Bottom-up	Sandwich
integration	Early	Early	Early
Time to basic working program	Early	Late	Early
Component drivers needed	No	Yes	Yes
Stubs needed	Yes	No	Yes
Work parallelism at beginning	Low	Medium	Medium
Ability to test particular paths	Hard	Easy	Medium
Ability to plan and control sequence	Hard	Easy	Hard

Hybrid Incremental Integration Approaches

- Risk Driven
 - Start by integrating the most critical or complex modules together with modules they call or are called by
- Schedule Driven
 - To the extent possible, integrate modules as they become available
- Function or Thread Driven
 - Integrate the modules associated with a key function (thread); continue the process by selecting another function, etc.

How About Object-Oriented Systems?

- Suppose a collection of cooperating objects are to be integrated to form a system or sub-system. Will the control structure necessarily be hierarchical?
- Which of the incremental integration strategies identified above appear to be applicable? Which do not?

Problem With Integration Testing

- Time consuming and expensive. Software development projects with 50% to 70% of effort on testing, and 50% to 70% of the testing effort on integration testing
- Most integration testing techniques are principles, such as incremental integration, top-down, and bottom-up integration
- One common Technique - explore the programming or design structure of the program. It is useful, but it is applicable to software written using the related techniques only. For example, it is possible that testing technique may not be applicable to a C++ program because Java has no pointers but C++ does.

Problem With Integration Testing (cont.)

- The SUTs are always very large, and each system may have many subsystems interconnected to each other. Verifying and validating these large interconnected systems has been a challenge.
- Most testing techniques focus on white-box testing and few take the functional approach to testing, while in practice, functional testing has been found to be effective in detecting faults.
- In fact, integration testing has been often the bottleneck of system and software development for both commercial and government application development

What is Required?

- An integration testing technique need to be applicable to a wide variety of applications using various programming languages such as COBOL, FORTRAN, ADA, Java, C and C++ and design techniques such as object-oriented design patterns and architecture
- It must be applicable to both legacy application as well as modern applications.
- It must be mostly functional or (black-box), however, it should be able to extend to include certain white-box information such as network characteristics and overall system architecture

What is Required? (cont.)

- Due to market competition, it is important to develop systems rapidly but without sacrificing any system qualities.
- Changes will be frequent and cannot be avoided
 - Extreme Programming (XP) and agile processes now welcome changes
 - System development and testing carried out in an incremental manner
 - Rapid and adaptive system development throughout the system life cycle.

End-to-End (E2E) Integration Testing

- This test process verifies that a defined set of interconnected systems, now subsystems of the integrated system, will perform correctly
 - Individual subsystems and applications may have been tested and approved, but unobserved faults may still exist
- It focuses exclusively from the end user's point of view
 - Different from traditional integration testing, which can focus on any subset of systems
- It assumes that both module testing and integration testing have been performed and approved, which may include multiple levels of integration testing

E2E Integration Testing

- It is independent of any development processes
- It can be developed early in the life cycle ,so test requirements can be concurrently carried out in the development process
- It assists in test case generation, improves productivity of software projects.
- It supports risk analysis by which risky areas can be identified so that they can be thoroughly tested
- It supports change management so that regression testing and ripple effect analysis can be carried out properly and efficiently.
- It supports statistical quality evaluation so that decision makers may evaluate the testing effort objectively and quantitatively
- It supports remote project management and distributed collaboration so that engineers and project managers can work together via the internet

History

- In 1999, we started a project on end-to-end integration testing for military applications.
- We began by specifying thin threads or test scenarios. Each thin thread is an execution path from the beginning to the end.
- Document and organize thin threads, and allow thin threads to be analyzed
 - Input/output, actions, conditions, events
 - Dependency analysis
 - Regression testing and ripple effect analysis
 - This is motivated by a trip to a DoD testing site during Y2K testing

History (cont.)

- In 2000, we moved from thin threads to scenarios, and in 2002 we implemented the scenario tool that convert scenarios into thin threads automatically.
- In 2002, Zhu reported verification patterns that also used scenarios.
 - 90+% of system scenarios can be classified into just few patterns.

History (cont.)

- In the meantime, test frameworks such as JUnit, HttpUnit, and Cactus are emerging at the same time. Their aim is to store test scripts and perform test execution as an integrated part of development process such as Extreme Programming. They used many design patterns.
- We developed similar ideas back in 1994-1996 at Guidant but the goal is mainly at the organizing and generating test scripts instead of test execution.
- We have implemented several OO test frameworks in Java for testing a variety of applications including embedded systems.

History (cont.)

- By specifying system scenarios only, it is possible to know the system behaviors, performance, and constraint verification by simulation
 - Separate environment simulation from system simulation
 - Generating a state model from simulation
 - Generating unidentified scenarios
 - Constraints checking
 - Partial simulation + partial execution to get realistic behavior and performance data. This is the goal of the current DoD project.

Related work – DOORS

ATM Withdraw Cash Scenario (Scenario Steps captured in DOORS Objects)

Step No	Name	Element Type	From	To	Performed By
1.2.1.1.2 Scenario Steps					
1	The Customer enters their card into the ATM	Card	Message	Customer	ATM
2	The ATM determines that the Card is valid for use in this ATM	Card Valid	Action		ATM
3	The ATM determines which Bank this ATM will serve as a client for	Determine Bank	Action		ATM
4	The ATM requests entry of Customer PIN	PIN?	Message	ATM	Customer
5	The Customer enters the correct PIN into the ATM	PIN	Message	Customer	ATM
6	The ATM requests validation of the PIN by the Bank	Validate PIN	Message	ATM	Bank
7	The Bank informs the ATM that the entered PIN is valid	PIN Valid	Message	Bank	ATM
8	The ATM requests the customer to select a transaction	Transaction?	Message	ATM	Customer
9	The Customer selects the withdraw transaction	Withdraw	Message	Customer	ATM
10	The ATM requests the customer to enter the withdrawal amount	Amount?	Message	ATM	Customer
11	The Customer enters the withdrawal amount into the ATM	Amount	Message	Customer	ATM
12	The ATM determines that the withdrawal amount specified by the customer can be supported by the denomination of notes held	Amount Valid	Action		ATM
13	The ATM requests authorisation of the withdrawal from the Bank	Withdrawal Request	Message	ATM	Bank
14	The Bank sends authorisation of the withdrawal to the ATM	Withdrawal Authorised	Message	Bank	ATM
15	The ATM asks the Customer whether a receipt is required for the withdrawal	Receipt?	Message	ATM	Customer
16	The Customer specifies that a receipt is required for the withdrawal	Receipt Required	Message	Customer	ATM
17	The ATM ejects the card	Card	Message	ATM	Customer
18	The ATM dispenses the cash	Cash	Message	ATM	Customer
19	The ATM dispenses the receipt for the transaction	Receipt	Message	ATM	Customer

Related work – DOORS (Cont.)

- *DOORS Scenario Modeling*, Telelogic, UK
 - Structuring scenarios to maximize reuse
 - Similar to our scenario templates concept
 - Linking scenarios to other parts of system specification
 - Similar to our dependency analysis
 - Scenarios = Actors + Flow of information between actors/Action performed by the actors
 - Similar to our scenario description
 - But we could not find any direct tool support for scenario analysis

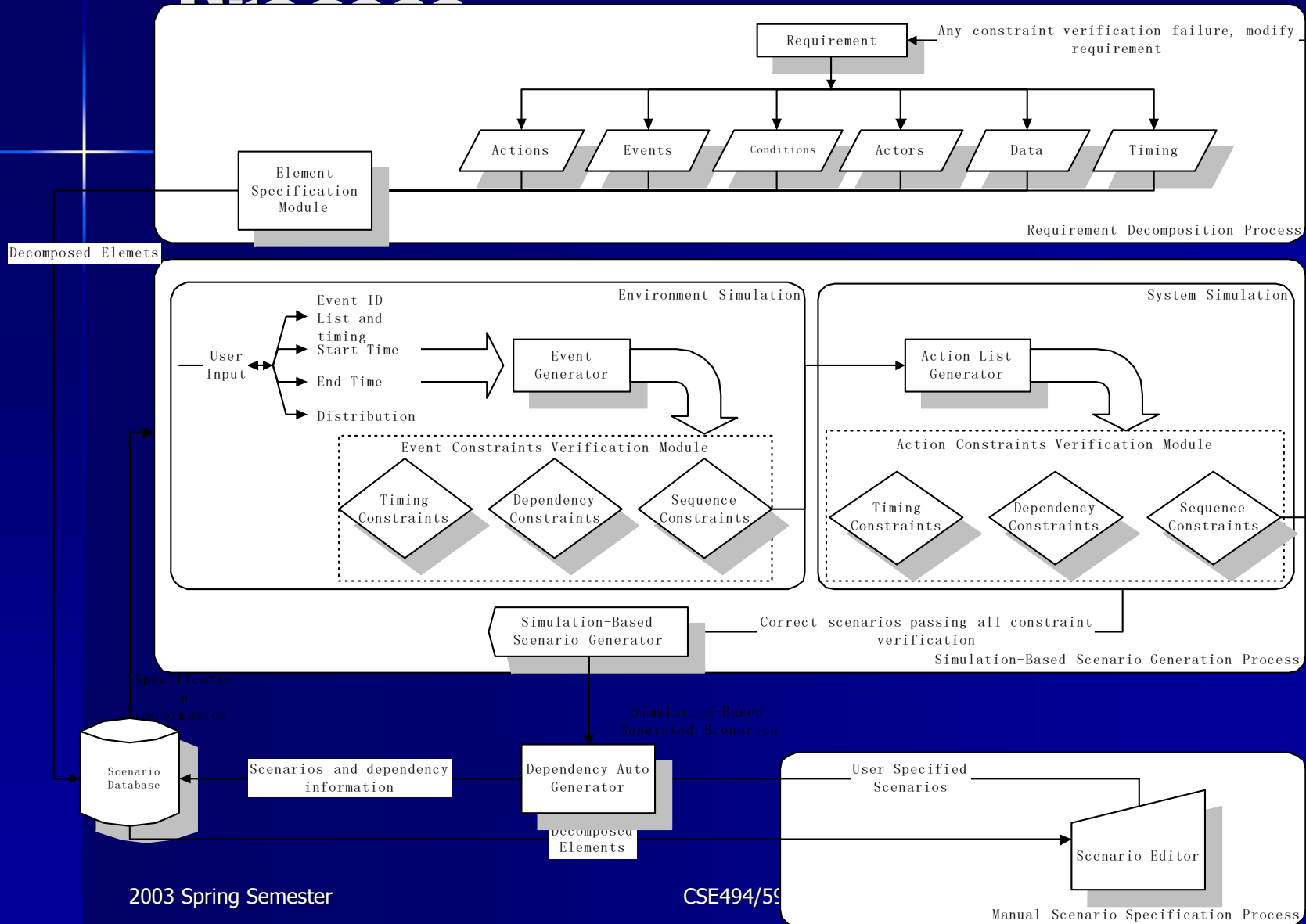
Related work – NASA SWAP

- *SWAP Scenario Specification*, NASA, Goddard Space Flight Center
 - Use scenario as an informal test procedure and demonstration script for the project.
 - Can not provide abilities to do various analysis on the natural language description of the scenarios
 - Defines *Pre-conditions*, *Post-conditions* and *Flow of events* for scenarios.
 - Similar to our scenarios but does not contain detailed information (ACDATE) for test generation

Comparison with Related Scenario-based Work

	SCENT	Pei Hsia	NASA	DOORS	E2E
<i>Perform dependency analysis</i>	Yes	Yes, but manually	No	Yes, but manually	Yes, automatically
<i>Test Configuration</i>	Yes	No	No	Yes	Yes
<i>Configuration Dependency</i>	No	No	No	No	Yes
<i>Test Condition</i>	Yes	No	Yes	No	Yes
<i>Condition Dependency</i>	No	No	No	No	Yes
<i>Group scenarios</i>	No	No	No	No	Yes
<i>Generate test scenarios</i>	Yes	Yes	Yes	Yes	Yes
<i>Generate complex scenarios</i>	No	No	No	No	Yes
<i>Rank each scenario</i>	No	Yes	No	No	Yes
<i>Impact analysis</i>	No	No	No	No	Yes
<i>Statistical model</i>	No	No	No	No	Yes
<i>Regression testing</i>	No	Yes	No	No	Yes
<i>Tool support</i>	No	No	No	Yes	Yes ⁵

E2E Test and Evaluation (T&E)



Scenario Specification

- Starting with a set of system scenarios
 - Scenario group (related scenarios)
 - Sub-scenarios (for sub-actions or SoS)
 - Atomic and complex scenarios
- Each scenario has the following information (ACDATE)
 - Actor: Components of a system
 - Condition: System decision information
 - Data: Input and output used
 - Action: System method
 - Timing: Timing information
 - Event: External stimuli as well as action result
- It is NOT necessary to be complete and consistent to begin with

ACDATE Definition

- Actors, Conditions, Data, Actions, Timing, Events (ACDATE)
 - They are the base-elements to construct the system requirements and scenarios

ACDATE Definition (cont.)

- Actor:

- Definition: A role played by a user or an external system interacting with the system to be specified
- An actor can be an internal subsystem, user, or device
- Actors are used to describe the source and destination of the actions and events

ACDATE Definition (cont.)

■ Condition

- Definition: One of a set of specified values that a data item can assume.
- in system specification, condition presents a specific system part-status, not global status
- All of conditions compose the system global status
- Condition is useful and necessary element in the *SELECTION* and *REPITITION* control structures
- The environment status could be a condition and cab be part of the system global status

Conditions (cont.)

- Some possible conditions that may affect thin thread
 - Communication conditions
 - Sequencing and timing conditions
 - Data conditions
 - Environmental conditions
- Conditions can also be grouped and organized into a tree structure to facilitate reuse and management. The root is the system under test. The branch node is a collection of conditions related to each other. A leaf node represents a concrete condition

Conditions (cont.)

- Relationships among conditions
 - Independent: two conditions are independent if one can happen regardless of the other
 - Trigger/trigger-by: one condition may trigger the other condition to activate
 - Mutually exclusive: two conditions are mutually exclusive if they cannot exist at the same time
 - Related: two conditions are related to each other if they are used in the same thread, or they are mutually exclusive. If condition A and B are used in the same thin thread, they are related to each other due to either system functionality or system structure

Conditions (cont.)

- By analyzing the relationships among conditions, a tester can build certain test criteria and select the most typical conditions to test effectively
 - We need to test the combination of related conditions because those not related may not affect the execution of related thin thread
- Conditions can also assist in risk analysis. A thin thread is risky if
 - It has a condition that has a high probability to failure
 - It has a condition whose failure can cause serious consequences
- Conditions can also be used in carrying out regression testing
 - A change in a condition potentially affects all the thin thread that share the same condition, and they are candidates for regression testing.

ACDATE Definition (cont.)

■ Data:

- Definition: Any representations such as characters or analog quantities to which meaning is, or might be, assigned.
- Data will be used to present the semantic of condition, event and action
- All of the data value/status can present the system global status
- Environment value will be the external data of the system. Internal data can be the system or subsystem's status, variables or the constant number in the requirement

ACDATE Definition (cont.)

■ Action

- Definition: The changes that occur in the system under study as a result of its functioning.
- Action must change the system global status, whatever the internal system status or external system status.
- Most of the system specifications are used to describe the system actions in some specific conditions or events
- Action has the properties or formal description to specify the detail semantic, such as the trigger, the affected actor, or the data change

ACDATE Definition (cont.)

■ Timing

- Definition: This specifies information related to timing such as delay and deadline for processes or events.
- Timing is one of the most important characteristics of real-time and embedded systems [Douglass 98].
- Timing is important because many systems have strict deadlines or delays. For example, after detecting an abnormal heart condition, a defibrillator must wait for a certain time it can apply therapy (i.e., a *delay* constraint) and the therapy must be applied before another time (i.e., a *deadline* constraint).
- Actions are specified with delay, execution, and deadline times and possibly also the start time or events.
- Events can have time-of-generation, and time-of-arrival.

ACDATE Definition (cont.)

- Event:

- Definition: the reception of a request sent by another actor to invoke an operation list according to the system status.
- Event will have a set of actions list according to different global status, at least one action list.
- Events can have many different types, from hardware, software or human:
 - Signal event
 - Timer event
 - Call event

Elevator Example -- ACDATE

- Actor: Elevator cart, Elevator door
- Data: elevator cart up, elevator cart down, elevator door open, elevator door close
- Action: TurnOnElevator, TurnOffElevator, OpenDoorInOneSecond, SendElevatorIsOnMessage, etc.
- Timing: elevator should be closed automatically after 10 seconds.
- Event: Press the open button, press the close button, press the up button, press the down button, etc.

Elevator Example - Condition

<ul style="list-style-type: none">■ <i>ElevatorIsOn</i>■ <i>ElevatorIsOff</i>	<ul style="list-style-type: none">■ <i>PressCloseButton</i>■ <i>PressOpenButton</i>
<ul style="list-style-type: none">■ <i>PressOnButton</i>■ <i>PressOffButton</i>■ <i>PressWButton</i>	<ul style="list-style-type: none">■ <i>PressUpButton</i>■ <i>PressDownButton</i>■ <i>PressUpOrDownButton</i>
<ul style="list-style-type: none">■ <i>DoorIsOpen</i>■ <i>DoorIsClosed</i>■ <i>DoorIsJustOpened</i>	<ul style="list-style-type: none">■ <i>ElevatorIsOnCertainFloor</i>■ <i>ElevatorIsnotOnCertainFloor</i>
<ul style="list-style-type: none">■ <i>ElevatorIsMove</i>■ <i>ElevatorIsStill</i>	<ul style="list-style-type: none">■ <i>ElevatorIsMovingUp</i>■ <i>ElevatorIsMovingDown</i>
<ul style="list-style-type: none">■ <i>DownDestinationIsHigherThanUpDestination</i>	<ul style="list-style-type: none">■ <i>DownDeatinationIsLowerThanUpDestination</i>

Elevator Example - ACDATE

<i>Scenario Description</i>	<i>Press the "Open" button. If the elevator is moving or off or not on a certain floor, a message will be given.</i>
<i>Event</i>	<i>User press the "Open" button</i>
<i>Pre-condition</i>	<i>Elevator is moving or off or not on a certain floor</i>
<i>Actor</i>	<i>Elevator</i>
<i>Action</i>	<i>Send out a message</i>
<i>Input</i>	<i>"Open" "Elevator is moving"/ "Elevator is off"/ "Elevator is not on a certain floor"</i>
<i>Output</i>	<i>"The operation is illegal"</i>
<i>Post-condition</i>	<i>The elevator keeps previous status.</i>

Reusability – Scenario Templates

- Scenario templates support verification pattern approach for system verification.
- Scenario templates construct the abstract scenario architectures as the invariant logical parts for the scenarios from the requirement patterns.
 - Similar business applications share the business logic, which can be represented by scenario templates.
 - When template changed, all of the inherent scenarios will be changed accordingly.
 - When template is changed, all the inherent scenarios will be changed accordingly.

Thin Thread

- A thin thread is defined as:

A complete trace (E2E) of data/messages using a minimally representative sample of external input data transformed through an interconnected set of systems (architecture) to produce a minimally representative sample of external output data. The execution of a thin thread demonstrates a method to perform a specified function

Thin Thread

- It is a minimum usage scenario of the integrated system
- It is a complete scenario from the end user's point
 - The system takes input data, performs some computation, and produces resulting output
- It describes the whole scenario and it describes *just* one function

Thin Thread

- Relationships among thin threads
 - Group/subgroup
 - Thin threads that share certain commonalities can be grouped together into a thin-thread group
 - Such grouping is hierarchical
 - From such grouping, all thin threads and thin thread groups can be arranged into a thin-thread tree. Its root is the overall integrated system under test. Its branch node represents a collection of related thin threads. A leaf represents a concrete thin thread
 - Thin threads in the same thin thread group are related by their common functionality

Thin Thread

- Contained in: the execution path of a thin-thread A can be a part of the path of another thin-thread B. For this relationship, A is a sub-thread of B, and B depends on A
 - Identical: A thin-thread A has the same path as another thin-thread B. In this case, A and B may share a certain set of attributes, such as conditions
 - Independent: Thin-threads A and B have completely different paths.
- These relationships among thin threads can be useful in scheduling test case execution
- If thin thread is on a critical path of the system, it should be tested thoroughly and probably as early as possible.
 - If a set of thin threads will be selected for testing, it may be appropriate to select thin threads with independent execution paths to ensure some kinds of coverage

E2E Integration Testing Features – Thin Thread

- Car Alarm system example

- 1. Control system

- 1.1 Car key

- 1.1.1 If either of the doors is opened, turning on the car-key makes the alarm horn beep three times (at this time, the operations by the remote controller/car-key are invalid)

- 1.1.2 The alarm system does not sound if one uses the car key to unlocked and open the door

- 1.1.3 The alarm system does not sound if one uses the car key to unlock and open the trunk

- 1.1.4 The engine cannot be started until the alarm system is turned off (i.e. disarmed) and a key is used to start the engine

- ...

- 1.2 Remote controller

- ...

- 1.3 By hand (break in)

- ...

- 2. Hood

- ...

- ...

Scenario vs. Thin Thread

- One exact and complete execution path in a scenario is a thin thread.
- Scenario combines related thin threads together.
- A scenario is a group of thin threads.
- Scenario uses pseudo codes (IF-THEN-ELSE, WHILE) to describe operations, thin thread describes testing in a sequential manner.
- The number of scenarios is less than that of thin threads.
- It is easier to specify to a scenario, but it is easier to generate test cases from a thin thread.
- The E2E tool can derive the corresponding thin threads from a scenario.

Requirements → Scenarios → Thin-Threads → Test cases

Scenario Analysis

- Various analysis techniques can be applied to ensure that system scenarios specified are correct with respect to the system requirements
- Static analysis
 - Completeness and consistency analysis
 - Timing analysis
 - Dependency analysis
 - Risk analysis

Scenario Analysis (continued)

■ Simulation

- Tracing events and actions specified in scenarios. It is possible to determine the scalability, performance (delays and throughput of the system), and behaviors of the system. No need to write simulation code.

■ Various analysis can be obtained by simulation

- Completeness analysis is performed based on
 - Dynamic analysis by simulation – as it may identify execution paths not specified in system scenarios;
 - Forms a feedback loop to complete scenarios
 - Potentially the entire system can be constructed this way
- Timing conflicts detection
- Concurrency analysis
- System performance analysis

Dependence Analysis

- Dependence analysis provides the foundation for regression testing and ripple effect analysis. The REA uses the dependency information to determine where additional changes may be needed to keep all the components consistent
 - Functional Dependence
 - Input Dependence
 - Output Dependence
 - Input/Output Dependence
 - Persistent Data Dependence
 - Execution Dependence
 - Condition Dependence

Timing Analysis

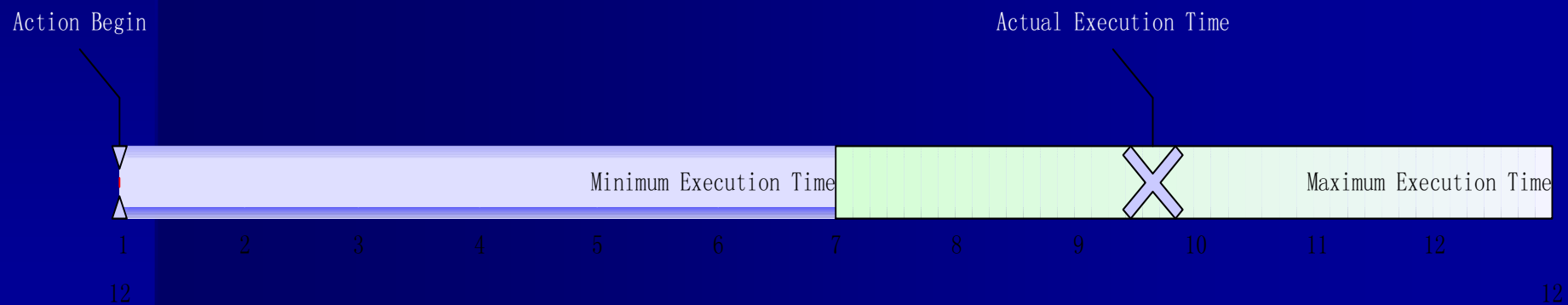
- Software systems, especially the real-time systems (usually a embedded system is real-time system), need to produce a correct output according to a pre-defined time schedule.
- Timing analysis is important to the verification of a system's behavior.

E2E Timing Information

- Two types of timing information for E2E testing
 - Execution Time
 - Minimum Execution Time
 - Maximum Execution Time
 - Timing Constraint
 - At – Define a time spot
 - Before – Define the deadline
 - After – Define the earliest start time
 - Between – Define the duration

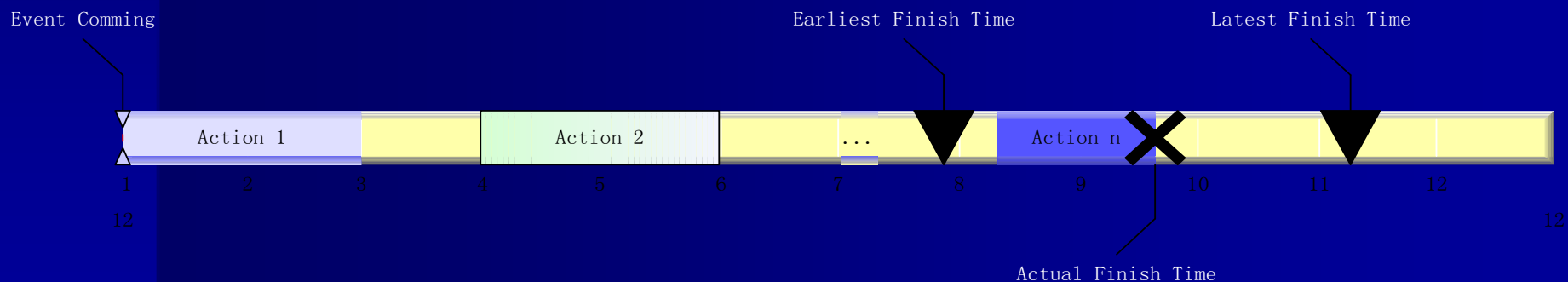
Execution Time

- Execution time is specified with a minimum value and a maximum value



Timing Constraints

- Timing Constraints specify the earliest finish time (T_E) and latest finish time (T_L) of the triggered actions
 - At: $T_E = T_L$
 - Before: $T_E = 0 < T_L$
 - After: $T_E < T_L = \infty$
 - Between: $T_E < T_L$



Basic E2E Timing Analysis

- When we are defining scenarios, we specify timing constraints for the conditions and actions
- Each action has its own property of execution time
 - This execution time can follow some distribution model such as Poisson distribution.

Basic E2E Timing Analysis (Cont.)

- With timing information specified for scenarios, we can do timing analysis for the scenarios:
 - Execution Time Analysis
 - Minimum, maximum, estimated, average
 - Deadline Analysis
 - Earliest start, latest start, earliest finish, latest finish

Timing Failures

- A timing-critical system has many timing constraints
- The implementation of a timing-critical system may contains timing failures
- To detect timing failures, we need to
 - Identify what kinds of timing failures may exist
 - Conditions for failures
 - Frequency of their occurrence
 - Criticality of failures

Types of Timing Failures

- Late Events/Missed Deadlines
- Premature/Early Events
- Range failures
- Interval Synchronization Problems
- Rate failures
 - Based on time intervals
 - Based on event counts
- Periodic failures

Simulation for Timing Analysis

- Simulation is a good way to find the potential timing failures:
 - Load/Stress Testing Simulation
 - Configuration Testing Simulation
 - Phase Testing Simulation
 - Concurrency Testing Simulation
 - Interval Synchronization Testing Simulation

E2E Integration Testing Features – Risk Analysis

- Risk analysis is an important activity in system and software development.
- According to it, the critical aspects of the system can be tested thoroughly
- When the resource is limited, the test effort can focus on those important threads first

E2E Integration Testing Features – Risk Analysis

- One way of ranking thin threads is to assign a risk for each thin thread based on at least the following two factors:
 - The probability that the system will fail a given thin thread
 - The following components often have a high-failure probability
 - Those components that have shown to be unreliable during prior module or integration testing
 - Those components that have complex implementation or incorporate complex functionality
 - Those components connected to many other components
 - Those components that have been recently changed due to faults or changes in functionality

E2E Integration Testing Features – Risk Analysis

- The consequence of failures – a thin thread should be tested more if its failure may jeopardize the system's mission or cause great harm to the system's environment

E2E Integration Testing Features – Risk Analysis

- The risk of a thin-thread is a function of its failure probability and the consequence of its failure:

$$\text{Risk}_{\text{thin-thread}} = F(\text{probability}_{\text{thin-thread}}, \text{Consequence}_{\text{thin-thread}})$$

- The risk of a condition can be estimated as:

$$\text{Risk}_{\text{condition}} = F(\text{probability}_{\text{condition}}, \text{Consequence}_{\text{condition}})$$

E2E Integration Testing Features – Risk Analysis

- The E2E test case is generated based on a thin thread and its condition. The test case risk can be estimated based on the risk of the thin thread and the risk of its conditions:

$$\text{Risk}_{\text{test-case}} = F(\text{probability}_{\text{test-case}}, \text{Consequence}_{\text{test-case}})$$

E2E Integration Testing Features – Risk Analysis

- The risk assignment of a thin thread is dynamic
 - Its value changes as the project progresses.
 - At the beginning of a testing project, where the probability of failure is high, almost all the thin threads are of high risk. As the system gets tested and corrected, its probability of failure may go down, and eventually, some of the thin threads will have minimum risks.
 - For high-consequence thin threads where the consequence of failures remains high; thus, even though their failure probabilities may go down, their risks remain high
 - Faults are often introduced during software modifications; thus, the risk assignment for those thin threads that executed on those changed components should go up after modifications

E2E Test Scenario

- Each thin thread is a basic test scenario.
- Combination of thin threads forms complex scenario to accomplish use's complex functions.
- Combination of thin thread can use the following three control operators:
 - Sequence
 - Looping
 - Conditioned execution

How To Select Test Inputs?

- The test inputs are infinite. So, we need to select test inputs
 - To select those points that are near the boundary of conditions (boundary testing)
 - To select test inputs randomly (random testing)
 - To classify the inputs into equivalent classes and select typical data in each category (partition testing)
 - To generate test cases based on usage (usage-based testing)

E2E Testing Process

- *Test planning* : to specify key tasks, as well as the schedules and resources associated with them
- *Test design* : to develop test specifications, test scenarios, test cases, and test schedule
- *Test execution* : to execute test cases and document results
- *Test results analysis* : to analyze test coverage, evaluate testing, and identify defects
- *Re-testing and regression testing* : to perform additional testing on the modified system

E2E Testing Plan

- The major concerns of integration test planning are to identify the scope of the integrated system including its system structure and functionality, to identify the processes techniques, and tools to conduct the integration test, and to define the result evaluation criteria before the actual testing.

E2E Testing Plan

- Important elements that should be considered in an E2E test plan are:
 - Major goal
 - Test scope
 - System functional and non-functional requirements,
 - Test environment
 - Test automation
 - Test results analysis plan
 - Test reuse plan
 - System back up plan
 - Activity schedules
 - Exit Criteria

E2E Process -- RDP

- Requirement Decomposition Process:
 - Identify the conditions
 - Identify which conditions will be used for a specific system. Say, the system will act according to which conditions are satisfied.
 - Identify other related items
 - Identify *Actors, Events, Actions*, and *Data (input and output)* that will be used by the system.
 - Re-write the requirements:
 - Condition + Actor + Event + Action + Input/Output

E2E Testing Plan

- Test Working Group

The best way to identify the information needed for an E2E test is to establish effective communication by forming a test-working group.

E2E Testing Design

- E2E test design includes tasks to define the integrated system under test and define the test procedures
 - The integrated system under test can be defined by following two perspectives:
 - The E2E functionality view
 - The structure view, which consists of both physical structure and logical structure.

E2E Testing Design

- The system under test can be specified by a thin-thread tree with conditions attached. The thin-thread tree construction is an iterative process consisting of following activities.
 - Identify and specify thin threads
 - Identify and specify conditions associated with thin threads
 - Organize thin threads and conditions into trees.
 - Perform completeness and consistency checking.

E2E Testing Design

- Generate test cases based on the thin-thread tree and condition tree specification.
- Analyze the risk for each thin thread
- Analyze the usage for each thin thread
- Schedule test cases for execution

E2E Testing Design

- E2E test specification
 - The core of E2E testing
 - Representations of system requirements
 - Usage scenarios from the end user's point of view
 - Detailed descriptions of system behaviors from test perspective: normal inputs, abnormal inputs, regular cases, and exception handling
 - Semi-formal, hierarchically structured
 - Attached with data for test generation
 - Traced to other software artifacts
 - Two parts: thin threads and conditions

E2E Testing Design

- Thin-thread template definition
 - ID, name, description, inputs/outputs, pre/post conditions, components covered, status, agents, and risks
- Thin-thread group
 - A collection of thin threads with certain commonalities
 - Recursive grouping, forming a tree structure
- Thin-thread tree
 - Top down construction, functional decomposition
 - Bottom-up construction, abstraction and composition
- Thin-thread relationships
 - Thin threads with independent execution paths
 - Thin threads with covered execution paths
 - Thin threads with identical execution paths

E2E Testing Design

- Predicates that must be true to activate the function
- Examples include: data conditions, communication conditions, environment conditions, and system status
- Condition analysis is part of completeness and consistency analysis
 - Uncovers new thin threads
 - Uncovers incomplete/inconsistent conditions
 - Uncovers thin threads that are attached to contradictory conditions

E2E Testing Design

- Conditions definition template
 - ID, name, description, affected thin threads, etc.
- Condition group
 - A collection of conditions with certain commonalities
 - Recursive grouping, forming a tree structure
- Condition tree
 - Top down decomposition
 - Bottom-up abstraction and composition
- Condition relationships
 - Independent conditions
 - Mutually exclusive conditions
 - Trigger/Trigger-by conditions
 - Related conditions

E2E Testing Design

- Generate Test Cases from Thin Thread or Complex Scenario
 - Identify the subsystems involved include both the software and hardware systems
 - Identify the input data for the thread
 - Use the input data satisfying the conditions associated with the thin thread based on different testing techniques
 - Determine the expected results from the thin thread description

E2E Testing Design

- Risk Analysis of Thin Threads and Conditions
 - It's important to thoroughly test the most essential functionality of the integrated system, which will jeopardize the mission or cause significant harm to the environment should they fail. Ranking mechanisms can be used to analyze the relative importance of each thin thread, so that when the resource is stringent, the test effort can focus on those important threads first.

E2E Testing Design

- One way of ranking thin threads is to assign a risk for each thin thread based on at least the following two factors:
- The probability that the system will fail a given thin thread. The following components often have a high-failure probability.
 - Those components that have shown to be unreliable during prior module or integration testing
 - Those components that have complex implementation or incorporate complex functionality
 - Those components connected to many other components
 - Those components that have been recently changed due to faults or changes in functionality.
- The consequence of failures.

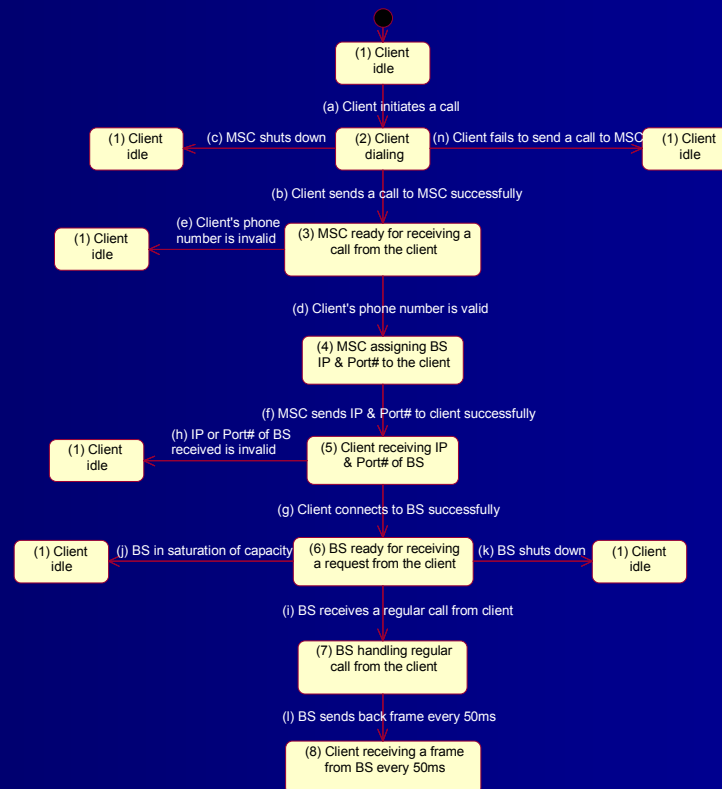
E2E Testing Design

- One way to estimate the failure probability is to use Assurance-Based Testing, where the failure probability and its confidence value can be determined using mathematical models based on the number of test cases and passed.
- The risk of a thin thread is dynamic, I.e., its value changes as the project progresses.
- The risk assignment also depends on the goal of testing.

Scenario-based State/Event Tree

- State models based on a scenario specification.
- For each scenario, one identify a sequence of states (each of which may be a composite state) and the associated events.
- Merging all the identified sequences of states results in a tree structure, called state/event tree .

Scenario-based State/Event Tree (continued)



Scenario-based State/Event Tree (continued)

- The proposed scenario-based state/event tree approach is different from the UML approach where states are used to model the behavior of an *object* only, while the state/event tree model the behavior of the *system* using scenarios and states.
- The state/event tree can be subject to various formal analysis such as completeness and consistency checking, dependency analysis, usage analysis, risk analysis, pattern analysis, redundancy analysis, REA, coverage analysis.
- Furthermore, an XML-based tool has been developed to support most of these analyses. This approach and tool have been used to test several state-based embedded systems successfully.

E2E Test Execution

- E2E Test Execution Preparation
 - Prior to the test, the test engineer needs to identify the following components.
 - The subsystem under test
 - The supporting systems, including hardware, firmware, database, and third party components, to ensure the subsystems can be executed properly.
 - The backup system and procedure, so that in case the test damage the system, it can be recovered.
 - The test data, including test input data, database, and files required to execute the test cases.
 - Test tools, including automatic input data generation tools, testing drivers and test results recording tools.
 - The test group.

E2E Test Execution

- E2E Testing in the Simulated Environment
 - Testing in the simulated environment includes these steps
 - Develop or acquire the simulation programs
 - Set the parameters of the simulation system as required by the system under test
 - Execute the application system
 - Select test cases, generate input data to the system external interfaces, record the execution results
 - Repeat the selection and execution of test cases, recovering the system and simulator states as necessary, until all the scheduled test cases have been exercised

E2E Test Execution

- E2E Testing in the Operational Environment
 - The process to perform testing in the operational environment includes these steps:
 - Set up the Environment
 - Invoke the application system
 - Select test cases, generate the input data to the system external interfaces, record the execution results
 - Repeat the selection and execution of test cases, recovering the system states as necessary, until all the scheduled test cases have been exercised.

E2E Test Execution

- Meeting the Exit Criteria
 - All of the scheduled test cases have been exercised
 - Testing coverage requirements have been achieved
 - Certain assurance has been gained.

E2E Test Execution

- Test Results Documentation
 - The results that should be documented during the E2E test execution include:
 - The test case selected and the requirement item that the test case is going to test against
 - The input data for each test case
 - The system output for each test case
 - The interface status of each subsystem, including the data exchanged on the interfaces between subsystems and between a subsystem and its supporting systems
 - The status of each subsystem under test, including hardware, software and supporting systems

E2E Testing Result Analysis

- Defect Identification and Correction
 - A defect is a mistake in the code that when executed may produce incorrect results.
 - To Identify the failures, the outputs of E2E test are compared against the expected outputs in the test specification.
 - Defects identified during the test should be prioritized and corrected.

E2E Testing Result Analysis

- Evaluate Ripple Effect
 - The phenomenon of the change to one part of a software artifact affecting other related parts is called **ripple effect**, and the iterative process of analyzing and eliminating side effects due to changes is called ripple effect analysis (REA)
 - The REA process follows these steps:
 - Propose a software modification;
 - Identify those parts of the software that depend on the changed segment
 - Determine if the dependent parts need to be changed to ensure consistency
 - If yes, continue with the first step starting with the parts that need to be changed. If no, stop and ready for software modification

E2E Testing Result Analysis

- The REA process is like a spanning tree, where the terminal nodes are the parts of software that depend on the previous parts but do not need to be changed.
- In particular, the REA process is not specific to any particular programming language or design paradigm. Specifically, the REA can be used to maintain consistency of the thin-thread tree and condition tree.

E2E Testing Result Analysis

- Assess Test Coverage
 - Additional testing may be necessary if any of the following situations exist.
 - If a specific feature or segment of a system has many failures or faults, it often implies that the feature or segment is error prone and requires additional testing
 - When defects are corrected, the modified system requires additional testing to ensure that the faults have been removed and no new faults are introduced during software modification.
 - When a functionality feature is changed, it is necessary to test the new feature to ensure that no new faults have been introduced during software modification.
 - When the software fails the test exit criteria, additional testing is required.

E2E Testing Result Analysis

■ Regression Testing

- Regression testing, or rerunning the existing test cases on the modified software, is commonly used whenever software program is modified. One key point is that regression testing only ensures that those parts that are supposed to remain unchanged.
- Regression testing should be carried out at multiple levels, first at the module level, then at the integration level, and finally at the E2E level.

E2E Testing Result Analysis

- At each level of regression test, the test engineer performs these tasks
 - Test case identification
 - Test case re-validation
 - Test case execution
 - Failure identification by examining test results
 - Fault identification and correction

E2E Testing Result Analysis

- Develop New Test Cases
 - Reevaluate the thin-thread tree and condition tree with respect to the modified system or requirements; reorganize, expand, or cut the thin-thread tree as necessary
 - Reevaluate the test cases with respect to the modified thin thread and condition tree. New test cases may need to be developed to:
 - Test the faulty areas or features
 - Test the modified or new features if their requirements are changed

E2E Testing Result Analysis

- Test Effectiveness Assessment

- The effectiveness of E2E testing should be assessed. The goal of testing is to find failures and faults; thus, the higher the number of failures/defects found, the more effective the test process. Given the same test effort, the metrics to assess the test effectiveness are:

- Number of defects detected/number of test cases used;
 - Number of thin threads passed/number of thin threads exercised
 - Number of failures detected/number of test cases used

Scenario Change Management

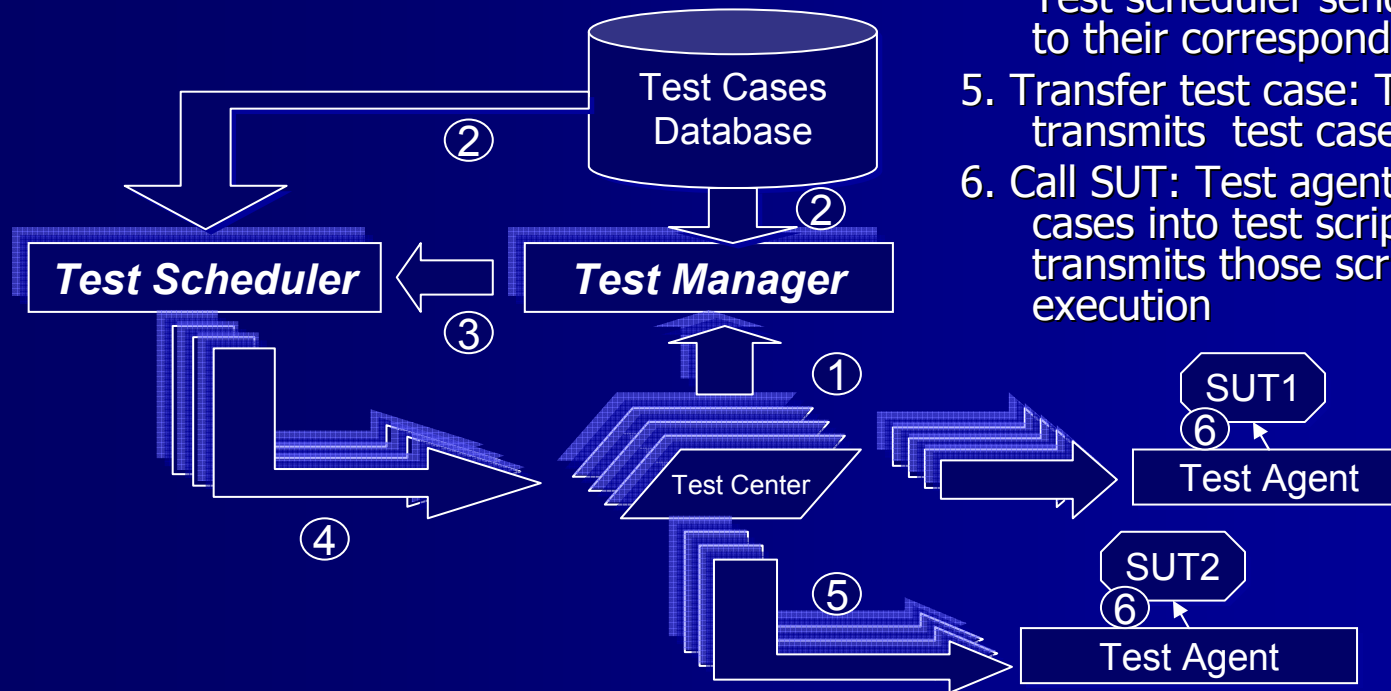
- E2E testing facilitate system changes by providing
 - Dependency analysis during system analysis, design, and testing.
 - Pattern analysis so that reusable components in analysis, design, and testing can be reused to support system modifications.

Remote Testing

- Testing distributed systems and devices is often difficult due to the asynchrony of distributed communication – also, the systems involved often reside at different locations.
- Test master and test agents communicate via network protocols
- Performs stress testing, scalability testing and performance testing
- Apply to web service testing
 - Many interesting issues involved here including UDDI check-in and checkout, domain test script interoperability, automatic generation of test scripts based on domain scenarios, SoS issues, and version management.

Distributed Test Execution

1. Registration: Test center registration process
2. Retrieve test cases: Test manager starts a test process/test plan that includes a set of test cases, which are retrieved from the database



3. Testing Schedule: Test manager sends current test plan's test cases and available test center information to the test scheduler
4. Transfer test cases to test center: Test scheduler sends testing cases to their corresponding test centers
5. Transfer test case: Test center transmits test cases to test agents
6. Call SUT: Test agent translates test cases into test scripts and transmits those scripts to SUT for execution

Tool Support

- An automated tool is now available and it supports the following E2E T&E activities:
 - Specification of scenarios
 - Specification of thin threads and thin thread trees
 - Specification of conditions and condition trees
 - Configuration management
 - Automated test case generation
 - Distributed test execution
 - Simulation
 - Query support and test report generation
 - Risk analysis
 - Ripple effect analysis and Regress testing
 - It has a GUI front-end and a database backend

E2E Database Overview

The screenshot displays the SCNRTools application interface for a project named [proj_LOE_V2.0.e2e]. The main window is divided into several panes:

- Left Pane:** A tree view showing the project structure under 'LOE_V2.0', including 'Data', 'Action', and three 'Phase' groups (Phase1, Phase2, Phase3), each containing multiple 'SubSCNR' configuration groups.
- Scenario Specification Table:** A table listing scenario elements.

RE A	Name	Creator	Created	Modifier	Last modified
✓	SurfaceTargetSubSCNR1	admin	2003-02-22 05:18:47	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR2	admin	2003-02-22 05:19:03	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR3	admin	2003-02-22 05:19:13	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR4	admin	2003-02-22 05:19:23	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR5	admin	2003-02-22 05:19:37	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR6	admin	2003-02-22 05:19:46	admin	2003-02-22 05:1
✓	SurfaceTargetSubSCNR7	admin	2003-02-22 05:51:56	admin	2003-02-22 05:5
✓	SurfaceTargetSubSCNR8	admin	2003-02-22 05:52:07	admin	2003-02-22 05:5
- Condition Specification Table:** A table listing condition elements.

RE A	Name	Creator	Created	Modifier	Last modified
✓	ImageNeededToBeSentBackT...	admin			
✓	ImageNeededToBePostedToWeb	admin			
- Configuration Specification Table:** A table listing configuration elements.

RE A	Name	Creator	Created	Modifier	Last modified
✓	CGGCCSSendTrackToSCCActor	admin	2003-02-22 05:46		
✓	SCCActorSendHOSTILEToGCC...	admin	2003-02-22 05:47		
✓	GCCSCopSendMsgToShooterM...	admin	2003-02-22 05:48		
✓	ShooterManuSendMsgToACQT...	admin	2003-02-22 05:49		
✓	ACQTGTSendMessageToGUN	admin	2003-02-22 05:50		
✓	BulletsReloadedToGUN	admin	2003-02-22 05:50		
✓	BulletsShotAtTarget	admin	2003-02-22 05:51		
- Scenario Dependency Table:** A table at the bottom showing dependencies between scenarios.

Name	Creator	Created	Modifier	Last modified
CriticalStrikeSub...	admin	2003-02-22 06:...	admin	2003-02-22 06:...
CriticalStrikeSub...	admin	2003-02-22 06:...	admin	2003-02-22 06:...
CriticalStrikeSub...	admin	2003-02-22 06:...	admin	2003-02-22 06:...
Scenario Dependency				

DOD E2E Scenario Example

The screenshot shows an XML Editor window titled 'XML Editor' with a menu bar containing 'C-Scenario', 'Tool', and 'Edit'. The main area displays a tree view of a scenario structure. The root node is 'ATOMIC SCENARIO'. It contains an 'IF CLAUSE' which is expanded to show its internal structure. The 'IF CLAUSE' contains an 'IF' block, which is further expanded to show a 'CONDITION' (COND ITEM) with the value 'CommandToLocateTargetBeDelivered (=63)'. This 'IF' block contains a 'THEN' block with an 'ACTION' 'EOIRLocateTarget (=64)'. This 'THEN' block contains another 'IF CLAUSE' which is expanded to show an 'IF' block with a 'CONDITION' (COND ITEM) 'ReconInfoNeededToBeSentToCrewActorDirectly (=68)'. This 'IF' block contains a 'THEN' block with an 'ACTION' 'EOIRSendReconInfoToCrewActor (=65)'. This 'THEN' block contains an 'ELSE' block with an 'IF CLAUSE' which is expanded to show an 'IF' block with a 'CONDITION' (COND ITEM) 'ReconInfoNeededToBeSentToWebCopForceview (=69)'. This 'IF' block contains a 'THEN' block with an 'ACTION' 'EOIRSendReconInfoToWebCopForceview (=66)' and another 'ACTION' 'WebCopForceviewSendReconInfoToCrewActor (=67)'. This 'THEN' block contains an 'ELSE' block with an 'ACTION' 'DoNothing (=30)'. The main 'IF CLAUSE' also contains an 'ELSE' block with an 'ACTION' 'DoNothing (=30)'. On the right side of the editor, there is an 'Insert' toolbar with various icons for adding elements, including a right-pointing arrow, a cross, a left-pointing arrow, a blue arrow with '0' and '1', a blue circle with '1', '&&||', 'true false', a globe with 'Go!', and a dashed line icon. At the bottom of the editor, a status bar shows 'Type= C-Scenario; ID= 58; Name= SurfaceTargetSubSCNR2'.

Name	Value	Time(ms)
ATOMIC SCENARIO		
IF CLAUSE		
IF		
CONDITION COND ITEM	CommandToLocateTargetBeDelivered (=63)	
THEN		
ACTION	EOIRLocateTarget (=64)	
IF CLAUSE		
IF		
CONDITION COND ITEM	ReconInfoNeededToBeSentToCrewActorDirectly (=68)	
THEN		
ACTION	EOIRSendReconInfoToCrewActor (=65)	
ELSE		
IF CLAUSE		
IF		
CONDITION COND ITEM	ReconInfoNeededToBeSentToWebCopForceview (=69)	
THEN		
ACTION	EOIRSendReconInfoToWebCopForceview (=66)	
ACTION	WebCopForceviewSendReconInfoToCrewActor (=67)	
ELSE		
ACTION	DoNothing (=30)	
ELSE		
ACTION	DoNothing (=30)	

Type= C-Scenario; ID= 58; Name= SurfaceTargetSubSCNR2

Ripple Effect Analysis using E2E Tools

- Following is the REA result for the selected thin thread.
- Different dependent level are denoted by different colors.
- REA indicates the parts that are affected by the change and need to be tested again.

Thin Thread Specification					
REACT	REA	Name	Creator	Created	Modifier
<input checked="" type="checkbox"/>	0	NarrativeStep3_SubSCNR4(Tr...	admin	2003-03-08 03:02:29	admin
At Condition: IF (McCainReceivesOPORDERFromP3) IF (McCainReceivesOPORDERFromP3)(TAODecidesToFire) Do Action: Router/RadioOnMcCainPostsOPORDEROnGCCS_COP TAOReceivesOPORDERFromWeb_COP TAOOrderWeaponSysToFire Wea					
<input checked="" type="checkbox"/>	1	NarrativeStep3_SubSCNR4(Tr...	admin	2003-03-08 03:02:29	admin
At Condition: IF (McCainReceivesOPORDERFromP3) NOT IF (McCainReceivesOPORDERFromP3)(TAODecidesToFire) Do Action: DoNothing					
<input checked="" type="checkbox"/>	2	NarrativeStep3_SubSCNR4(Tr...	admin	2003-03-08 03:02:29	admin
At Condition: NOT IF (McCainReceivesOPORDERFromP3) Do Action:					

DOD REA Results using Scenario Tools

- The following is the REA result screen shot of DOD projects.
- In regression testing, if changes should be made to the scenario with blue color, those scenarios with red color also need to be checked, because they depend on the scenario with blue color.

The screenshot shows the SCNRTools interface with the following data:

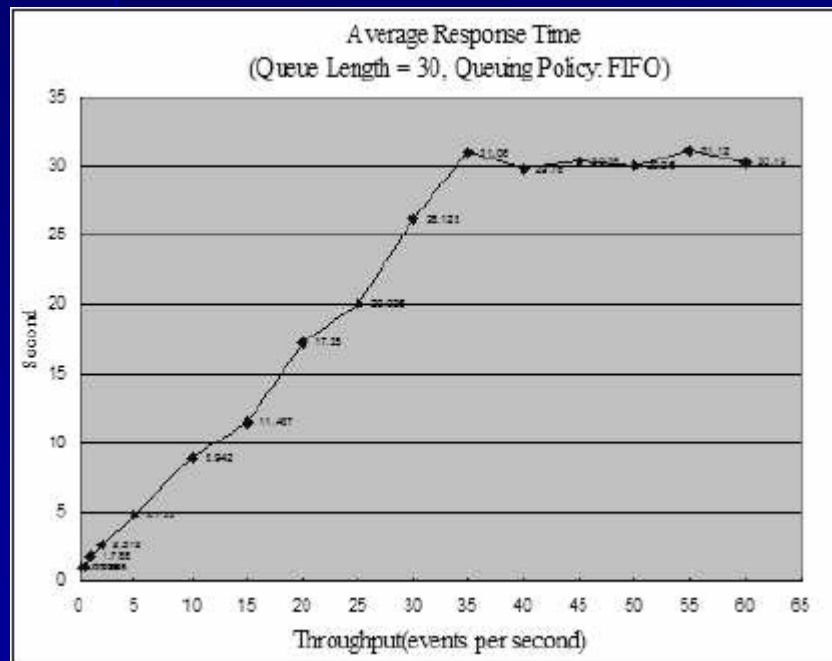
REA	Name	Creator	Created	Modifier	Last modified
0	NarrativeStep3_SubSCNR1	admin	2003-03-08 03:02:29	admin	2003-03-08 03:02:42
1	NarrativeStep3_SubSCNR2	admin	2003-03-08 03:02:29	admin	2003-03-08 03:02:49
1	NarrativeStep3_SubSCNR3	admin	2003-03-08 03:02:29	admin	2003-03-08 03:02:55
1	NarrativeStep3_SubSCNR4	admin	2003-03-08 03:02:29	admin	2003-03-08 03:03:00

REA	Name	Creator	Created
	CrewAwareOfTargetFromRadar	admin	2003-03-08 02:05:59
	TargetImageIdentifiedByEO/IR	admin	2003-03-08 02:06:52
	CrewAcquireTargetImageFrom...	admin	2003-03-08 02:08:08
	CrewConnectMCNetwork	admin	2003-03-08 02:08:40
	RadarTrackDataSentViaMCN...	admin	2003-03-08 02:10:09
	TargetImageSentViaMCNetw...	admin	2003-03-08 02:10:47
	MCNetworkPostRadarTrackDa...	admin	2003-03-08 02:12:29

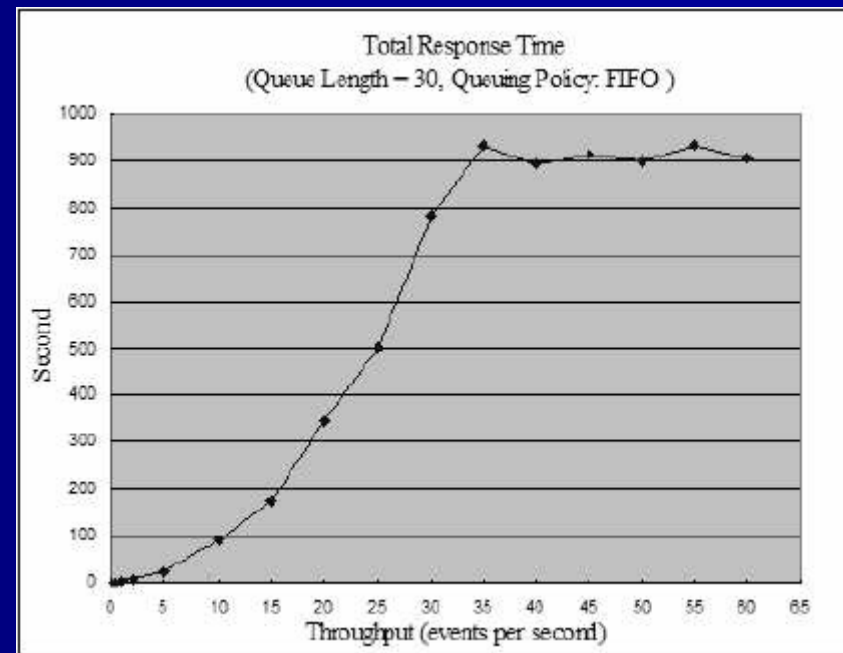
Name	Creator	Created	Modifier	Last modified
NarrativeStep8_...	admin	2003-03-08 07:...	admin	2003-03-08 07:...
NarrativeStep5_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep7_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep7_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep7_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep7_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep6_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep5_...	admin	2003-03-08 05:...	admin	2003-03-08 05:...
NarrativeStep4_...	admin	2003-03-08 05:...	admin	2003-03-08 05:...
NarrativeStep6_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep6_...	admin	2003-03-08 06:...	admin	2003-03-08 06:...
NarrativeStep5_...	admin	2003-03-08 05:...	admin	2003-03-08 05:...

Car Alarm System Simulation Data

- The following figures show the car alarm system's average response time and total response time when a queue with capacity of 30 events was applied



2003 Spring Semester



CSE494/598

118

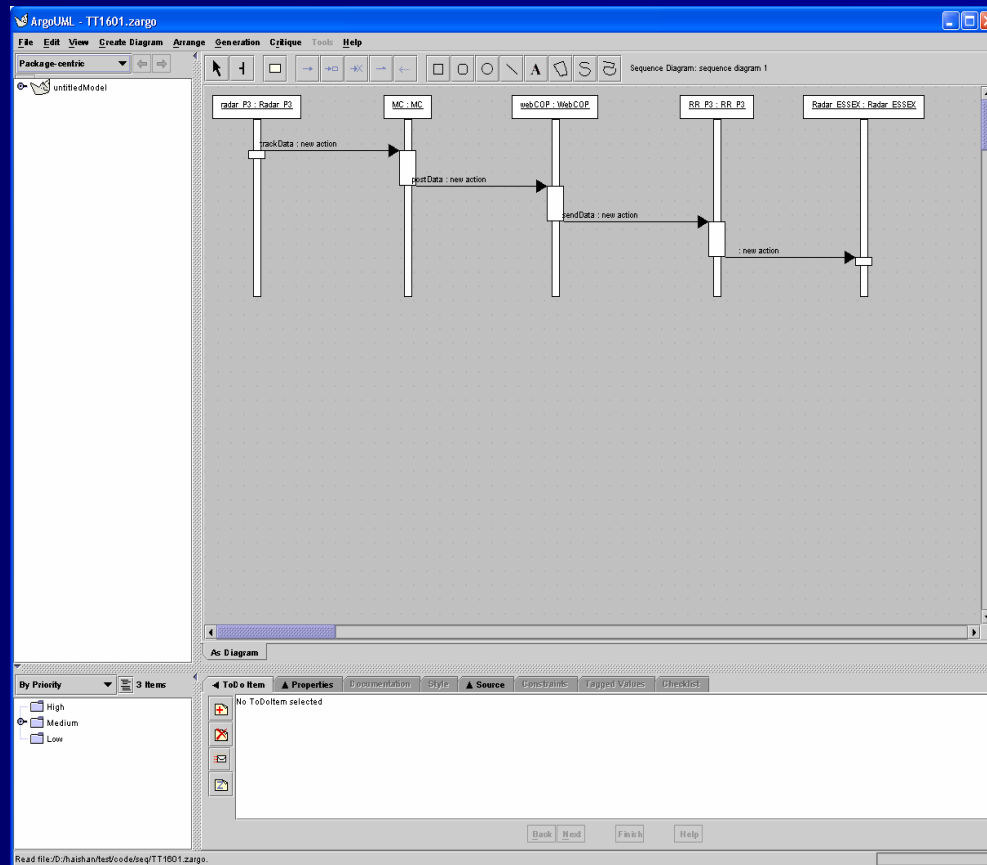
Timing Analysis

- Timing is important in both designing and testing.
- In E2E simulation, external timing constraints specify how the system should response to external events in terms of timing
- During simulation, E2E calculates the minimum and maximum time the system needs to handle each event and compare the results with the specification to check if the system probably fails to handle certain events.
- If more detailed timing requirements of LOE are supplied, all the timing constraints can be analyzed with E2E simulation tools

E2E Timing Analysis -- Timing Conflicts Detection

- Timing conflicts detection can be done at run-time in system simulation
- Given a global state, system simulation will simulate all possible system behaviors
- Two scenarios will conflict if and only if two scenarios with same pre-conditions have two different timing constraints.

UML Sequence Diagram Generation



ETEX – E2E Testing Execution

- ETEX is a distributed testing execution process based on the information provided by ETC. It provides the following testing methods:
 - Remote Testing: Testers can test the SUT remotely
 - Scheduled Testing: Testers specify testing plan and then ETEX schedules the registered testing center to execute test
 - Stress Testing: Multiple test centers send test scripts to a particular SUT in order to test SUT performance under heavy load
 - Agent Based Testing: Agents accept testing command, translate test scripts to SUT and collect test results.

Test Case Generation

- We define value set for each datum that will be used in the system under testing.
- Once we get the thin threads, we can attach different data to them to generate the test cases.
- There are different strategies to choose different value from the data value set to do:
 - Random testing
 - Partition testing
 - Boundary value testing

Example Code of Using Patterns

LogicPattern class:

```
public abstract class LogicPattern
    public void runLogic() // template method
        // initialize
        doLogic();
        // clean
    abstract void doLogic(); // primitive method
    abstract String genCode();
```

Logic{M} class:

```
public class LogicM extends LogicPatternN
    public boolean performActionA()
        // override this method
        // implementation of action A
        ...
    public boolean performActionB()
        // implementation of action B
        ...
```

LogicPattern{N} class:

```
public class LogicPatternN extends BizPattern
    public LogicPatternN()
    public void doLogic() // primitive method for BizPattern and also template method
        if (performActionA() == true) // the steps are implemented following business
            if (checkConditionB()) // logic pattern.
                displayMessage();
                // continue logic...
            else
                displayMessage();
                performActionB();
        else
            stopProcess();
    // every action must be defined as primitive method
    public abstract boolean performActionA()
    public boolean performActionB() ... // primitive class may have default implementation
                                        // only different parts are implemented at sub-classes
    public String genCode()
    ...
```

Case Study -- Scenario Tool Overview

The screenshot displays the SCNRTools application window with the following components:

- Scenario Specification - Intel:** A table listing sub-scenarios.

RE A	Name	Creator	Created	Modifier	Last modified
	PerformActionA_SubScenario	admin	2003-04-19 01:00:16	admin	2003-04-19 01:00:16
	PerformActionZ_SubScenario	admin	2003-04-19 01:00:48	admin	2003-04-19 01:00:48
	ManageConditionB_SubScenario	admin	2003-04-19 01:01:27	admin	2003-04-19 01:01:27
	PerformActionCAndD_SubScen...	admin	2003-04-19 01:02:33	admin	2003-04-19 01:19:14
- Condition Specification - Intel:** A table listing conditions.

RE A	Name	Cr
	ActionAPass	ad
	EventTHappen	ad
	ConditionBMet	ad
	ActionCPass	ad
	ActionDPass	ad
	ProblemHappenMoreThenThre...	ad
	EventTNotHappen	ad
	ActionDCompleteEventNotHap...	ad
	OutputFromActionAlsAvailable	ad
	MessageOfConditionBHasBeen...	ad
- Configuration Specification - Action:** A table listing actions.

RE A	Name	Creator	Created	Modifier
	PerformActionA	admin	2003-04-19 00:53:31	admin
	PerformActionZ	admin	2003-04-19 00:54:12	admin
	DisplayMsg	admin	2003-04-19 00:55:10	admin
	PerformActionB	admin	2003-04-19 00:55:22	admin
	PerformActionC	admin	2003-04-19 00:55:39	admin
	PerformActionD	admin	2003-04-19 00:55:50	admin
	WaitForActionDCompleteEvent	admin	2003-04-19 00:56:35	admin
	CorrectTheProblem	admin	2003-04-19 00:56:54	admin
	ProduceOutputFromActionA	admin	2003-04-19 01:04:48	admin
	StopTheProcessAndQuit	admin	2003-04-19 01:05:25	admin
	Wait	admin	2003-04-19 01:09:32	admin
	End	admin	2003-04-19 01:11:38	admin
- Scenario Dependency:** A table at the bottom showing dependencies.

Name	Creator	Created	Modifier	Last modified
PerformActionCA...	admin	2003-04-19 04:...	admin	2003-04-19 04:...
WholeProcess_C...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
WholeProcess_C...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformActionCA...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformAction_S...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformActionCA...	admin	2003-04-19 04:...	admin	2003-04-19 04:...
PerformActionCA...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformAction_S...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformAction_S...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
ManageConditio...	admin	2003-04-19 01:...	admin	2003-04-19 01:...
PerformActionZ...	admin	2003-04-19 01:...	admin	2003-04-19 01:...

Scenario and Code

The screenshot shows an XML Editor window titled "XML Editor" with a tab labeled "C-Scenario". The window is divided into two main sections. The top section displays a tree view of a scenario, and the bottom section displays the corresponding Java code.

Name	Value	Time(ms)
SUB_SCENARIO		
ACTION	PerformActionC (=17)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionCPass (=10)	
THEN		
ACTION	PerformActionD (=18)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDCompleteEventNotHappen (=33)	
THEN		
ACTION	Wait (=31)	

```
public class IntelExample {
    public static void main(String [] args) {
        ActionBControlCenter insActionBControlCenter = new ActionBControlCenter();
        WholeProcessControlCenter insWholeProcessControlCenter = new WholeProcessControlCenter();
        ActionDControlCenter insActionDControlCenter = new ActionDControlCenter();
        ActionCControlCenter insActionCControlCenter = new ActionCControlCenter();
        insActionCControlCenter.PerformActionC();
        if (insActionCControlCenter.ActionCPass())
        {
            insActionDControlCenter.PerformActionD();
            if (insActionDControlCenter.ActionDCompleteEventNotHappen())
            {
                insWholeProcessControlCenter.Wait();
            }
        }
        else
        {
        }
    }
}
```

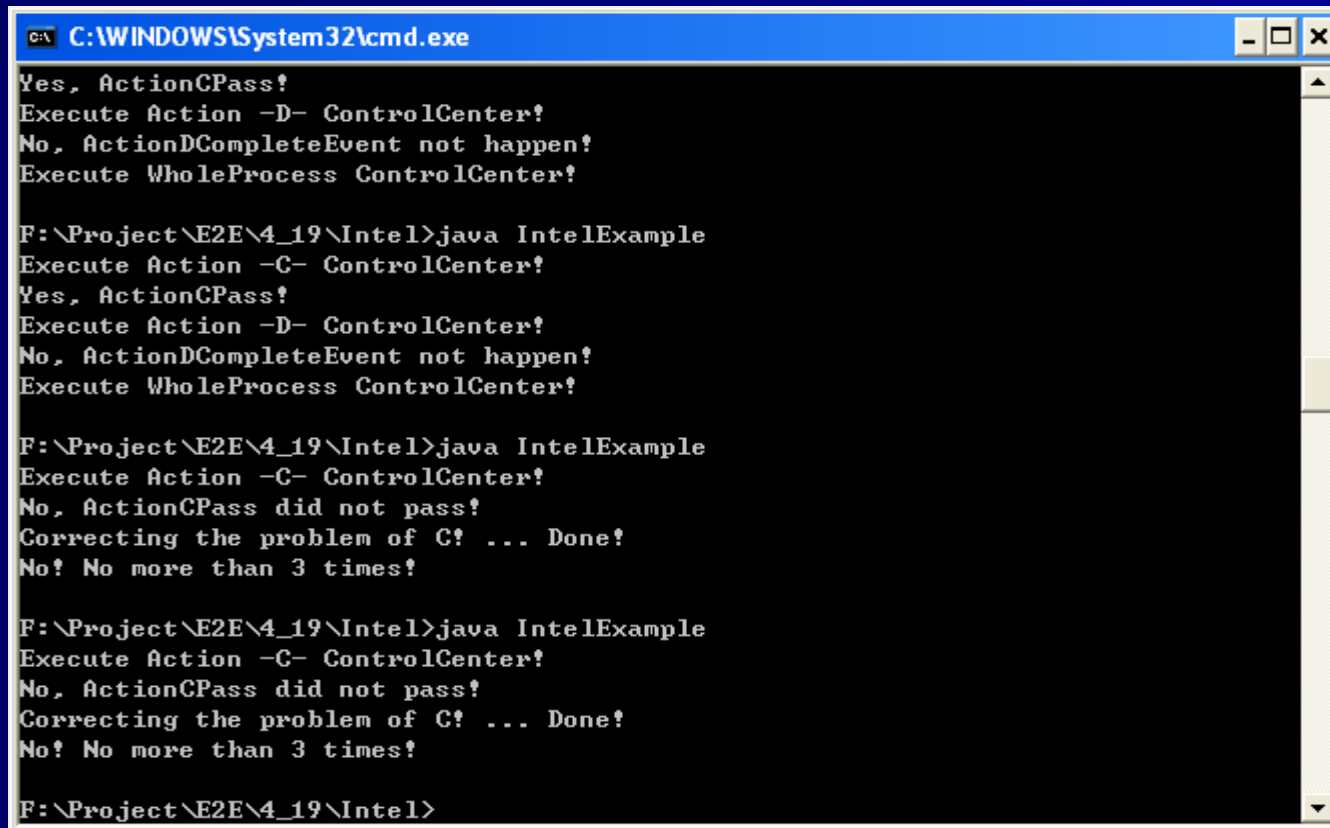
Scenario

Name	Value	Time(ms)
SUB_SCENARIO		
ACTION	PerformActionC (=17)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionCPass (=10)	
THEN		
ACTION	PerformActionD (=18)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDCompleteEventNotHappen (=33)	
THEN		
ACTION	Wait (=31)	
ELSE		
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDPass (=11)	
THEN		
ACTION	End (=32)	
ELSE		
ELSE		
ACTION	CorrectTheProblem (=20)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ProblemHappenMoreThanThreeTimes (=12)	
THEN		
ACTION	DisplayMsg (=15)	
ACTION	PerformActionB (=16)	
ACTION	End (=32)	
ELSE		

Code

```
C:\IntelExample.java
2      public static void main(String [] args) {
3          ActionBControlCenter      insActionBControlCenter = new ActionBControlCenter ();
4          WholeProcessControlCenter  insWholeProcessControlCenter = new WholeProcessControlCenter ();
5          ActionDControlCenter      insActionDControlCenter = new ActionDControlCenter ();
6          ActionCControlCenter      insActionCControlCenter = new ActionCControlCenter ();
7          insActionCControlCenter.PerformActionC ();
8          if (insActionCControlCenter.ActionCPass ())
9          {
10             insActionDControlCenter.PerformActionD ();
11             if (insActionDControlCenter.ActionDCompleteEventNotHappen ())
12             {
13                 insWholeProcessControlCenter.Wait ();
14             }
15             else
16             {
17                 if (insActionDControlCenter.ActionDPass () )
18                 {
19                     insWholeProcessControlCenter.End ();
20                 }
21             }
22         }
23         else
24         {
25             insActionCControlCenter.CorrectTheProblem ();
26             if (insActionCControlCenter.ProblemHappenMoreThanThreeTimes ())
27             {
28                 insWholeProcessControlCenter.DisplayMsg ();
29                 insActionBControlCenter.PerformActionB ();
30                 insWholeProcessControlCenter.End ();
31             }
32         }
33     }
34 }
```

Execution Result



```
C:\WINDOWS\System32\cmd.exe
Yes, ActionCPass!
Execute Action -D- ControlCenter!
No, ActionDCompleteEvent not happen!
Execute WholeProcess ControlCenter!

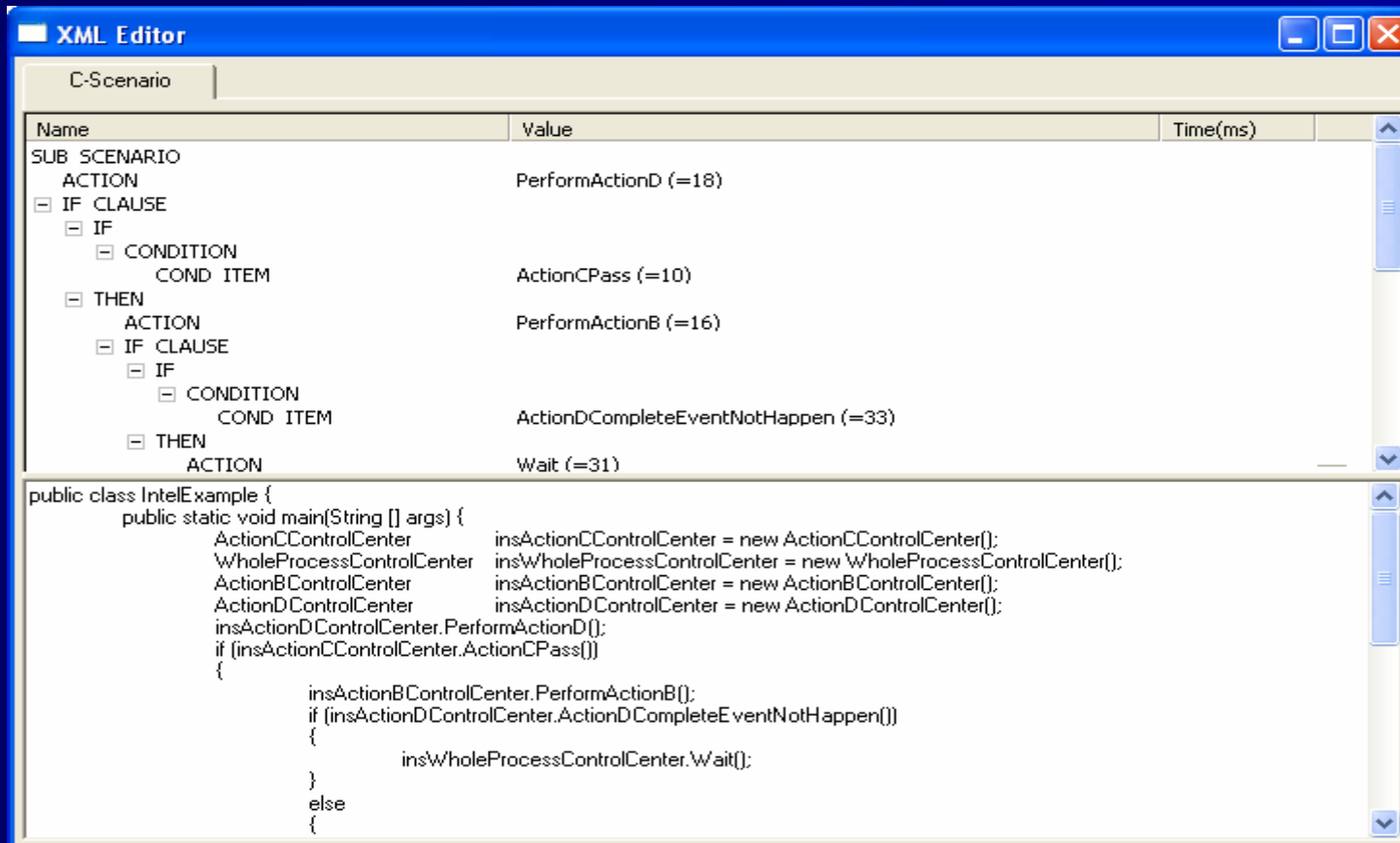
F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -C- ControlCenter!
Yes, ActionCPass!
Execute Action -D- ControlCenter!
No, ActionDCompleteEvent not happen!
Execute WholeProcess ControlCenter!

F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -C- ControlCenter!
No, ActionCPass did not pass!
Correcting the problem of C! ... Done!
No! No more than 3 times!

F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -C- ControlCenter!
No, ActionCPass did not pass!
Correcting the problem of C! ... Done!
No! No more than 3 times!

F:\Project\E2E\4_19\Intel>
```

Scenario and Code after Change



The screenshot shows an XML Editor window titled "XML Editor" with a tab labeled "C-Scenario". The window is divided into two panes. The top pane displays a tree view of a scenario, and the bottom pane displays the corresponding Java code.

Name	Value	Time(ms)
SUB SCENARIO		
ACTION	PerformActionD (=18)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionCPass (=10)	
THEN		
ACTION	PerformActionB (=16)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDCompleteEventNotHappen (=33)	
THEN		
ACTION	Wait (=31)	

```
public class IntelExample {
    public static void main(String [] args) {
        ActionCControlCenter insActionCControlCenter = new ActionCControlCenter();
        WholeProcessControlCenter insWholeProcessControlCenter = new WholeProcessControlCenter();
        ActionBControlCenter insActionBControlCenter = new ActionBControlCenter();
        ActionDControlCenter insActionDControlCenter = new ActionDControlCenter();
        insActionDControlCenter.PerformActionD();
        if (insActionCControlCenter.ActionCPass())
        {
            insActionBControlCenter.PerformActionB();
            if (insActionDControlCenter.ActionDCompleteEventNotHappen())
            {
                insWholeProcessControlCenter.Wait();
            }
        }
        else
        {

```

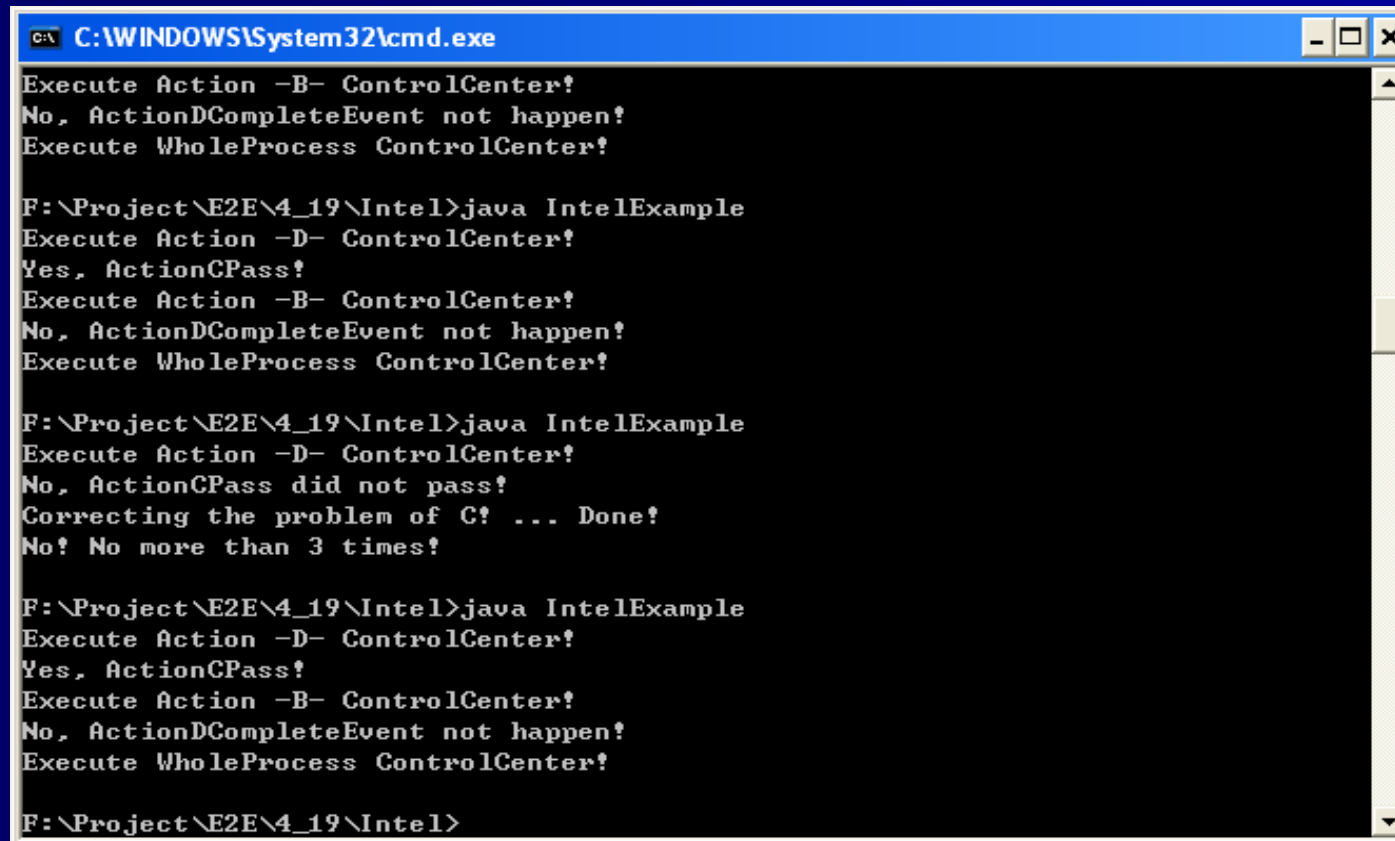
Scenario After Change

Name	Value	Time(ms)
SUB SCENARIO		
ACTION	PerformActionD (=18)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionCPass (=10)	
THEN		
ACTION	PerformActionB (=16)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDCompleteEventNotHappen (=33)	
THEN		
ACTION	Wait (=31)	
ELSE		
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ActionDPass (=11)	
THEN		
ACTION	End (=32)	
ELSE		
ELSE		
ACTION	CorrectTheProblem (=20)	
IF CLAUSE		
IF		
CONDITION		
COND ITEM	ProblemHappenMoreThanThreeTimes (=12)	
THEN		
ACTION	DisplayMsg (=15)	
ACTION	PerformActionC (=17)	
ACTION	End (=32)	
ELSE		

Code After Change

```
C:\IntelExample.java
2      public static void main(String [] args) {
3          ActionCControlCenter      insActionCControlCenter = new ActionCControlCenter();
4          WholeProcessControlCenter insWholeProcessControlCenter = new WholeProcessControlCenter();
5          ActionBControlCenter      insActionBControlCenter = new ActionBControlCenter();
6          ActionDControlCenter      insActionDControlCenter = new ActionDControlCenter();
7          insActionDControlCenter.PerformActionD();
8          if (insActionCControlCenter.ActionCPass())
9          {
10             insActionBControlCenter.PerformActionB();
11             if (insActionDControlCenter.ActionDCompleteEventNotHappen())
12             {
13                 insWholeProcessControlCenter.Wait();
14             }
15             else
16             {
17                 if (insActionDControlCenter.ActionDPass() )
18                 {
19                     insWholeProcessControlCenter.End();
20                 }
21             }
22         }
23         else
24         {
25             insActionCControlCenter.CorrectTheProblem();
26             if (insActionCControlCenter.ProblemHappenMoreThanThreeTimes())
27             {
28                 insWholeProcessControlCenter.DisplayMsg();
29                 insActionCControlCenter.PerformActionC();
30                 insWholeProcessControlCenter.End();
31             }
32         }
33     }
34 }
```

Execution Result After Change



```
C:\WINDOWS\System32\cmd.exe
Execute Action -B- ControlCenter!
No, ActionDCompleteEvent not happen!
Execute WholeProcess ControlCenter!

F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -D- ControlCenter!
Yes, ActionCPass!
Execute Action -B- ControlCenter!
No, ActionDCompleteEvent not happen!
Execute WholeProcess ControlCenter!

F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -D- ControlCenter!
No, ActionCPass did not pass!
Correcting the problem of C! ... Done!
No! No more than 3 times!

F:\Project\E2E\4_19\Intel>java IntelExample
Execute Action -D- ControlCenter!
Yes, ActionCPass!
Execute Action -B- ControlCenter!
No, ActionDCompleteEvent not happen!
Execute WholeProcess ControlCenter!

F:\Project\E2E\4_19\Intel>
```

References:

- R. Paul, “End-to-End Integration Testing”, Quality Software, 2001. Proceedings. Second Asia-Pacific Conference on , 10-11 Dec. 2001
Page(s): 211 -220
- W. T. Tsai, X. Bai, R. Paul, W. Shao, and V. Agarwal, “End to End Integration Test Design”, to appear in IEEE Proc. of COMPSAC, 2001
- X. Bai, W. T. Tsai, R. Paul, K. Feng, L. Yu, “Scenario-Based Business Modeling”, Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85287, 2001.
- X. Bai, W.T. Tsai, R. Paul, T. Shen and B. Li, “Distributed End-to-End Testing Management”, to appear in IEEE Proc. EDOC, 2001.
- M. Dyer, “The Cleanroom Approach to Quality Software Development”, Wiley, New York, New York, 1992.

References:

- Shari Lawrence Pfleeger, *Software Engineering Theory And Practice*, 2nd, 2001. Prentice Hall, Upper Saddle River, New Jersey