Integration Testing

Wei-Tek Tsai

Department of Computer Science and Engineering

Arizona State University

Tempe, AZ 85287

1

Introduction

- Integration testing is critical to ensure the functional correctness of the integrated system.
- Integration testing can be divided into two categories:
  - Incremental integration testing: incremental testing expands the set of integrated modules prograessively.
  - Non-incremental integration testing: by non-incremental testing, software modules are combined and tested randomly.
- Integration testing is often the most time consuming and expensive part of testing.

2

E2E Testing

- Why E2E testing?
  - Testing is still the primary means for software quality assurance.
  - Integration and functional testing, considered the most effective techniques, remain weak.
    - Most are principles, lack of techniques
    - Restricted to particular application type
  - Evaluation and measurement are important to both the end product and the process that produces it.

3

E2E Testing

- What is E2E testing?
  - Verifies that a set of interconnected systems will perform correctly.
    - Individual subsystems and applications may have been tested and approved, but unobserved faults may still exist.
  - Focuses on system functionality exclusively from the end user’s point of view.
    - Different from integration testing, which can focus on any subset of systems.
    - Assumes that module and integration tests have been performed and approved.

4

E2E Testing

- Benefits of E2E testing
  - Extends test objective from fault detection and prediction to partition and integration test planning with overall system fault tolerance.
  - Independent of development techniques
  - Promotes concurrent engineering
    - Can be developed early in the life cycle so that test requirements can be carried out concurrently with the development process
  - Enforces test management
    - Data management, project management, change management, and reuse management
  - Improves test productivity and effectiveness

5

E2E Testing

- A structured and disciplined 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

6

E2E Test Planning

- 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.

7

E2E Test Planning

- 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 Critiria

8

E2E Test Planning

- 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 Test 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 Test 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.

11

E2E Test 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

12

E2E Test 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

13

E2E Test Design

- Thin thread definition:

a thin thread is A complete trace (E2E) of data/message using a minimally representative sample of external input data transformed through an interconnected set of system (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.

(U.S. Department of Defense Year 2000 Management Plan)

14

E2E Test 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
  - Button-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

15

E2E Test Design

- Conditions
  - Factors that affect the execution of the function identified by a thin thread
  - 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

16

E2E Test 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
  - Button-up abstraction and composition
- Condition relationships
  - Independent conditions
  - Mutually exclusive conditions
  - Trigger/Trigger-by conditions
  - Related conditions

17

E2E Test Design

- Test scenarios
  - Function description for a test session
  - Simple test scenario
    - A basic thin thread
    - Verifies system correctness with individual functions
  - Complex test scenario
    - A composition of thin threads
    - Tests complicated usage of the system with multiple functions triggered in various orders
    - Verifies system reliability, stability, recoverability, performance, and other non-functional requirements
- Test cases
  - Generate test data based on various test criteria

18

E2E Test 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.

19

E2E Test Design

- One way of ranking thin threads is to assign a rich 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.

20

E2E Test 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.

21

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.

22

E2E Test Execution

- E2E Testing in the Simulated Environment
  - Testing in the simulated environment includes there 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

23

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.

24

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.

25

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

26

E2E Test 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.

27

E2E Test 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

28

E2E Test 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 sued to maintain consistency of the thin-thread tree and condition tree.

29

E2E Test 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.

30

E2E Test Result Analysis

- Regression Testing
  - Regression testing, or rerunning the existing test cases on the modified software, is commonly used whenever a 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.

31

E2E Test 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

32

E2E Test 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 areads or features
    - Test the modified or new features if their requirements are changed

33

E2E Test 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 cased used

34

References

- [1]. 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
- http://149.169.25.9/e2e/E2E_testing.html
- [2]. Assistant Secretary of Defense for Command, Control, Communications, and Intelligence (ASD C3I) Investment and Acquisition Directorate, "End-to-End Integration Test Guidebook", Software Engineering Lab, Department of Computer Science and Engineering, Arizona State University.
- http://149.169.25.9/e2e/E2E_testing.html
- [3]. 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.
- http://149.169.25.9/e2e/E2E_testing.html
- [4]. 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. http://149.169.25.9/e2e/E2E_testing.html
- [5]. M. Dyer, "The Cleanroom Approach to Quality Software Development", Wiley, New York, New York, 1992.
- http://www.cleansoft.com/cleansoft_overview.html

35

References

- [6]. S. Kirani and W. T. Tsai, "Specification and Verification of Object-Oriented Programs", technical report, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, 1994. http://www.cs.umn.edu/tech_reports/1994/TR_94-64_Specification_and_Verification_of_Object-Oriented_Programs.html
- [7]. D. C. Kung, P. Hsia, and J. Gao, "Testing Object-Oriented Software", IEEE Computer Society Press, Los Alamitos, CA, 1999. http://lglwww.epfl.ch/Research/OO_Research/toos/
- [8]. G. J. Myers, "The Art of Software Testing", New York, Wiley Inter-science, New York, New York, 1979.
- http://swexpert.com/C9/SE.C9.AUG.00.pdf
- [9]. R.S. Pressman, "Software Engineering: A Practitioner's Approach", McGraw Hill, 5th edition, 2000.
- http://www.mhhe.com/engcs/compsci/pressman/index.mhtml
- [10]. W. T. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Hierarchical Scenario Templates", Proc. COMPSAC, 1999, pp. 166-171.
- http://asusrl.eas.asu.edu/papers/Design-Pattern-Testing.htm

36

References

- [11]. W. T. Tsai, V. Agarwal, B. Huang, R. Paul, “Augmenting Sequence Constraints in Z and its Application to Testing”, Proc. 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 41-48. http://asusrl.eas.asu.edu/papers/ZTestToSend3.htm
- [12]. W. T. Tsai, R. Paul, W. Shao, S. Rayadurgam, and J. Li, “Assurance-Based Y2K Testing”, 1999 Proc. 4th IEEE International Symposium on High-Assurance Systems Engineering, pp. 27-34.
- http://asusrl.eas.asu.edu/papers/abt-hase.htm
- [13]. W. T. Tsai, R. Mojdehbakhsh and F. Zhu, “Ensuring System and Software Reliability in Safety-Critical Applications”, Proceedings of IEEE Application-Specific Systems and Software Engineering Technology, 1998.
- http://asusrl.eas.asu.edu/papers/REA-asset98.htm
- [14]. W. T. Tsai, R. Paul, W. Shao, S. Rayadurgam, and J. Li, “Assurance-Based Testing”, Proc. of IEEE High-Assurance Software Engineering (HASE), 1999.
- http://asusrl.eas.asu.edu/abt/pracguide.asp

37