7th International Conference on Software Engineering Advances (ICSEA 2012)

PANEL

Challenges in Testing and Validation of Hardware/Software Systems

Lisbon, November 19th , 2012

Feature Interaction Testing: An Industrial Perspective

Jameleddine Hassine

Department of Information and Computer Science

KFUPM, Kingdom of Saudi Arabia

jhassine@kfupm.edu.sa

The Feature Interaction Problem

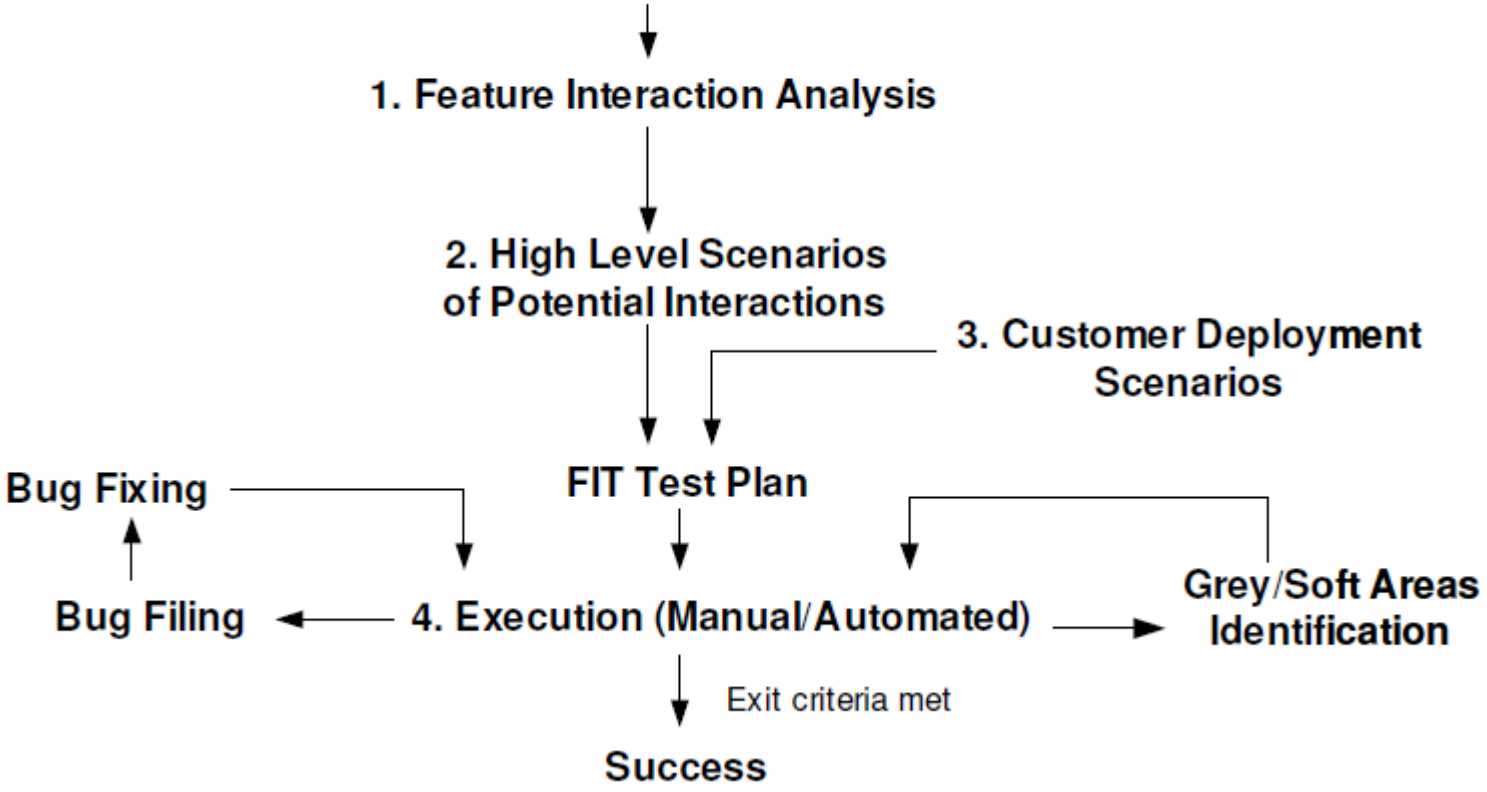
- Increased competition between Telecommunication service providers
- Pressure on network equipment vendors to rapidly develop and deliver innovative advanced features
- This challenge is hindered by the **Feature Interaction Problem**
- May severely damage feature development and deployment

The Feature Interaction Problem (cont.)

- Feature Interaction occurs when the integration of two **features** would modify the **behavior** of one or both features
- Recognized as an important problem in telecommunications since the early 1980s
- Large number of contributions from academia and industry:
  - Detection
  - Resolution
  - Avoidance
- Research trends:
  - Software engineering approaches (i.e., requirements, modeling, specification and design techniques)
  - Formal methods (applied during design-time of feature, called also off-line techniques)
  - On-line techniques (applied while the features are running)

Feature Interaction Testing (FIT) Process

The Goal is to Discover undesirable (harmful) interactions



Feature Interaction Testing (FIT) Process

1. **Feature Interaction Analysis (FIA)**
   - FIT suffers from the problem of combinatorial state explosion
   - Impractical to test all feature interactions
   - FIA Matrix reporting which feature may interact with which and to what extent (Low, Medium or High)
2. **High Level Feature Interaction Scenarios**
   1. Testbed Details: The type of setup (e.g., MPLS vs. IP), type of HW used (e.g., ATM, GE Linecard), routing protocols used (e.g., BGP, IS-IS)
   2. Feature Provisioning (Configuration flavors of the interacting features)
      - Feature configuration level: Apply the feature globally on the router or on a specific LC or interface
      - Feature flavors and options: Many variants with different configuration parameters
   3. Feature Interactions Triggers
      - Feature provisioning and permutations (order of applying the feature)

Feature Interaction Testing (FIT) Process

2. High Level Feature Interaction Scenarios (cont.)

4. Network traffic patterns
   - Features may treat data traffic differently (e.g., in different ASICs and queues, in different SW components) which may cause feature interactions in the data plane (e.g., packets drops, corrupted packets)
   - Traffic stream: Protocol (e.g., IPv4/IPv6, UDP, RTP, Multicast, etc.), packet size, inter-packet gap, jitter, etc.
5. Network events
   - Shut/unshut physical interfaces, Reloading network LC, Physically removing/putting back LCs
6. Stress testing
   - Continuous addition/removal of configurations, restarting feature specific processes

Feature Interaction Testing (FIT) Process

3. Deployment Scenarios

- Explores the potential customer deployment scenarios involving the new set of features (network architecture, configurations, scalability numbers (one-dimensional and multidimensional), etc.).
- Would help detect quickly the issues that customers are likely to face in their live networks
- More efficient and cost-effective FIT coverage

Feature Interaction Testing (FIT) Process

4. FIT Test Plan Execution

- FIT test plan is executed using different network topologies with different scale numbers (for different customers)
- Some parts of the FIT test plan can be automated
- Test plan execution may be shifted to cover the identified grey areas (high rate bugs) first
- A risk prioritization process that aims to choose the 10% to 15% most critical test cases from the high level scenarios
- Risk-driven activity that requires an assessment of the impact of such priority shifting
- Exit criteria: 100% execution completed, pass rate is above 95 %, and no high severity bugs are still open

FIT Process Challenges

- When to start FIT cycle?
  - After UT/IT?
  - In parallel with individual feature testing (short-time to market) ?
- FI Analysis
  - Subject matter experts (SME) input ? High-level View?
  - Low level View (static/dynamic dependencies analysis)?
- Desired vs. Undesired Feature Interaction
  - Is it a Feature or a Bug ?
- Risk-driven activities
  - Selection of deployment scenarios
  - Shift the execution to gray areas
- Automation
  - How much automation is needed ?
  - Physical network events cannot be automated

Thank You

PANEL Topic:

Challenges in Testing and Validation of Hardware/Software Systems

SoftNet 2012
November 18-23, 2012 – Lisbon, Portugal

Author: Fabien PEUREUX
Contact: peureux@smartesting.com

Is Business Process model adequate to capture and validate business requirements?

- BPM allows transparency, efficiency, predictability and more flexibility by offering a systematic and standardized approach to manage business processes
- BPM bridges the gap between IT and business requirements/needs by creating a common language and a consistent documentation for the involved stakeholders
- BPM establishes a new communication era since the IT teams can now efficiently collaborate with the business experts through modeling to achieve a strong alignment of quality objectives
- BPM standardization allows to ensure a better traceability between business processes and implemented business rules, and makes it possible to (semi-)automate the validation process

Is Model-Based Testing Compatible with Agile Development?

- Model-Based Testing consists in deriving (semi-)automatically test cases from a test model. In concrete terms, if a requirement changes, the model should be updated and tests can be quickly reproduced. A high-level of abstraction is required in order to manage the complexity. Moreover, firstly it helps to handle the requirements on a non-ambiguous and synthetic format, second it provides a systematic and highly productive way for test case creation and maintenance. By keeping solely the test model aligned with the customer (client or product owner) requirements at all times, it offers the guaranty of an up-to-date test repository.
- The generated test case is typically a sequence of high-level SUT actions, with input parameters and expected output values for each action. These generated test sequences are similar to the high-level user scenarios that would be designed manually to drive agile development. They are easily understood by humans and are complete
- The generated test cases can also be used as acceptance tests (at the API level) that are written by the agile development team or the product owner with the help of the final client. They check that the application fits the requirements (functional, performance, security, etc.). The high-level abstraction of the generated test cases allow the customers to easily review and validate them.

Structural testing vs functional testing What is the optimal strategy?

- Structural and functional test cases complement each other. The optimal strategy thus requires to use the both since each test family checks different types of errors:
  - Structural testing allows to avoid code errors such as infinite loops, null pointer exceptions,… (it checks that the program does the things right)
  - Functional testing allows to avoid inadequate services regarding end-user needs and requirements (it checks that the program does the right things)
- Nevertheless, due to some constraints, if only one test family should be used, functional testing has to be maintained since:
  - Functional test cases are often less fragile than structural test cases (code source changes more than API)
  - Fatal code errors can sometimes be discovered by functional testing
  - Functional testing directly allows to ensure the customer confidence and satisfaction