Workshop:
Modeling Scenarios Using Data

Fiona Charles
BCS SIGIST 22-Sept-2009
Who I Am:

Fiona Charles - Test Consultant and Test Manager, President and Principal Consultant of Quality Intelligence, Inc.

I teach organizations to match their software testing to their business risks and opportunities

With 30+ years experience in software development and integration projects, I have managed testing and QA, and consulted on testing on many projects for clients in retail, banking, financial services, health care, and telecommunications.

My articles on testing and test management appear frequently in Better Software Magazine and on StickyMinds.com. I edited The Gift of Time (Dorset House, 2008), and am co-founder and host of the Toronto Workshop on Software Testing, an annual peer conference for senior test practitioners.

I advocate, design, implement and teach pragmatic, humane practices to deliver software worth having—in even the most difficult project circumstances
Today’s menu

Test scenarios & modeling

Modeling based on data

Scenario modeling exercise

Conclusions
Scenario testing is a powerful way to find bugs that matter to stakeholders

- Exercise a system with real sequences of transactions and events
- Find problems in the end-to-end processing that deep functional testing can miss
- Represent various stakeholders’ points of view
- Give stakeholders a test they can relate to
What do we mean by “scenario”?

An outline or model of an expected or supposed sequence of events


“A scenario is a hypothetical story, used to help a person think through a complex problem or system...A scenario test is a test based on a scenario.”

Cem Kaner, An Introduction to Scenario Testing, 2003
Modeling for scenario testing operates at two principal levels

- The overall model for the test of the system or solution (and its systems context)
- The scenarios that encapsulate the tests to be evaluated
Why “modeling”?

• In every test, we make choices
  – What to include
  – What to leave out

• Consciously modeling a test gives us a way to direct, control, examine, and explain those choices

“Every model is ultimately the expression of one thing...we hope to understand in terms of another that we do understand.”

Gerald M. Weinberg, An Introduction to General Systems Thinking
Some useful model types for scenario testing

• Business operations
  – “Day (week, month, mock-year) in the life”
  – Model office or business

• Entity lifecycles or flows
  – Customer experience
  – Lifecycle of a bank account

• Data
  – Static
  – Semi-static
  – Dynamic

• Functional or organizational decomposition
  – Functional areas within the system or business (Ordering, Inventory Management, Billing, etc.)
  – Processes in each area (Order capture, provisioning, etc.)
  – Functions within each process (Enter, edit, cancel order, etc.)
It’s a good idea always to combine 2 or more models

- Each model type can act as a source of test ideas

- Different models can act as cross-checks on each other, generating ideas for complementary scenarios, e.g.,
  - Combining the real-world view of a business operations model with the systems view of a model based on the data
A conceptual framework for modeling scenarios based on categories of data
What does a model framework do?

- Scenarios are stories modeling some aspect of the system
- A conceptual framework gives you a way to structure the stories
  - Analyze the test requirements:
    - powerful tool for driving out variations to feed scenarios
  - Design the test model
- You can create building blocks for reuse in different combinations
- Helps to ensure you don’t miss anything important
You can construct a conceptual framework to facilitate developing any model

- Ask “what are the key constituent elements of this model?”
- Create a spreadsheet or other structure to order the elements
- Use the elements to capture variations and apply the model
A data-driven model focuses on the data and data interfaces

- Inputs, outputs and reference data
- Points of entry to the system or integrated solution
- Frequency of changes to data
- Who or what initiates or changes data (actors)
- Data variations (including actors)
Benefits of a data-driven model

- Represents a systems view that can be rounded out using models based on a business view
- Identifies principal components of the execution model for the test:
  - Setup data
  - Entry points
  - Verification points
  - Events to schedule
- Easy to structure and analyze
A high-level framework based on data is useful for modeling a test of a transactional system (or multi-system solution).
At the most basic level, we want scenarios to test the outcome of system transactions.

**Transactions** drive dynamic data (i.e., data that changes in the course of the test).

Transactions represent the principal business functions the solution is designed to support, e.g. sales, returns, loyalty point redemptions, customer service adjustments.
A transaction could be initiated by a human actor or by the system.

There may be more than one actor involved in a transaction, e.g., a system user and a customer.
Many scenarios will have multiple transactions.

Subsequent transactions can affect the outcome by acting on the dynamic data created by earlier related transactions.
Transactions operate in a context of test bed data

Reference test bed data is static (doesn't change in the course of the test), e.g., user profiles, frequent shopper (loyalty) points awarded per dollar spent, sales tax rates.

Deciding which data should be static is a test strategy decision.
The test bed also contains semi-static data, which can change the context and affect the outcomes of transactions.

**Semi-static data** changes occasionally during testing, as the result of an event. Examples include items for sale, prices and promotions.

Determining which data will be semi-static, and how frequently it will change, is a test strategy decision.
Events affect transaction outcomes—by changing the system or data context, or by acting on the dynamic data.

**Events** can represent:

- periodic or occasional **business processes**, e.g., rate changes, price changes, weekly promotions, month-end aggregations
- **happenings**, such as system interface failures
- “external” **business exceptions**, such as a shipment arriving damaged, or a truck getting lost
Scenarios also operate within a context of date and time.

The date/time a transaction or event occurs can have a significant impact on a scenario outcome.
Categorizing the data determines each type’s role in the test, and gives us the overall model

- Reference data sets the context for scenarios and their component transactions
- A scenario begins with an event or a transaction
- Transactions have expected results
- Events operate on transactions and affect their results
  - A prior event changes transaction context, e.g., an overnight price change
  - A following event changes the scenario result and potentially affects the transaction, e.g., product not found in the warehouse
- Actors influence expected results
  - User privileges
  - Customer discount or tax status
Example from Point-of-sale System test
Scenario modeling exercise
Exercise and Debrief
Conclusions
Situations where you should consider scenario testing

• As part of an overall test strategy that includes different kinds of tests
• For Acceptance tests of business systems
  – UAT
  – Vendor acceptance
• End-to-end systems integration tests
• When you don’t have detailed knowledge to do under-the-covers testing and have no way to get it
  – Inadequate documentation
  – Restricted access to people who wrote the software
  – Time pressures
Critical success factors for scenario testing

• Domain knowledge
  – Testers with experience in the domain
  – Business input and scenario review
  – Industry books (Cem Kaner’s suggestion)

• A model with a framework that fits the type of application
  – A data-driven model works well for transactional systems
  – For other types of applications, e.g., a desktop publishing system, you would need to create a different model and framework, such as one based on usage patterns

• Structured analysis

• Building-block concept: design and build varied and complex tests from elements
  – Begin testing with the simplest cases before adding complexity
Scenario testing risks

- Miss important bugs you could find with under-the-covers or deep function testing
- Difficult to diagnose bugs, especially if you go too complex too soon
- Choosing a model that is too restrictive
- Choosing a model that is too expansive (and expensive)
- Falling in love with one model type and missing the benefits of others
Further reading


http://www.stickyminds.com
Wrap-up questions and discussion