Paradigm shift in UI Testing

ivan.inozemtsev@xored.com
Traditional test case structure

- test case = actions + assertions
- action – some operation on a system
- assertion – 1 bit of information about system state
Implicit state management

Each test relies on a state from previous test

- False negative results
- Can’t run a random subset of tests
Explicit state management

- Each test case brings a system into an expected state
- Typically this is done by “clean everything and create a state from scratch” approach
Test case structure

- action
- action
- action
- action
- action
- action
- assert
- assert
- assert
- assert

Setup initial state
Perform test actions
Verify final state
Test case structure

Initial state description

Actions

Final state verification
State as a model

- Actions describe ‘how’
- Models describe ‘what’
State as a model

- Models can be reused
- Models can be captured
State projections

Full state description is

- Big
- Complex
- Too custom
- Not necessary
State Projections

- Workspace state
- Workbench state
- Preferences
- Launch configurations
State Projections

- Simpler models
- Fine-grained reuse
- Set only relevant state
  - Don’t care about workspace when testing
  - About dialog
Dependent Projections

Not all states are orthogonal

- Problems view depends on workspace
- Outline view depends on an active editor
- Variables view depends on a current breakpoint
- Editor colors depend on preferences
State verifications

- Model some part of a final state
- Calculate model diff between expected and actual
- Don’t need to be able to apply
  - So we can test UI projections of internal state
- Can test state delta (timestamp, error log)
Tests without actions

- Set some part of a system state
- Verify dependent part

Examples

- Workspace → Problems
- Java Editor → Outline
Demo
Thank you!

Visit http://q7.xored.com for more information