

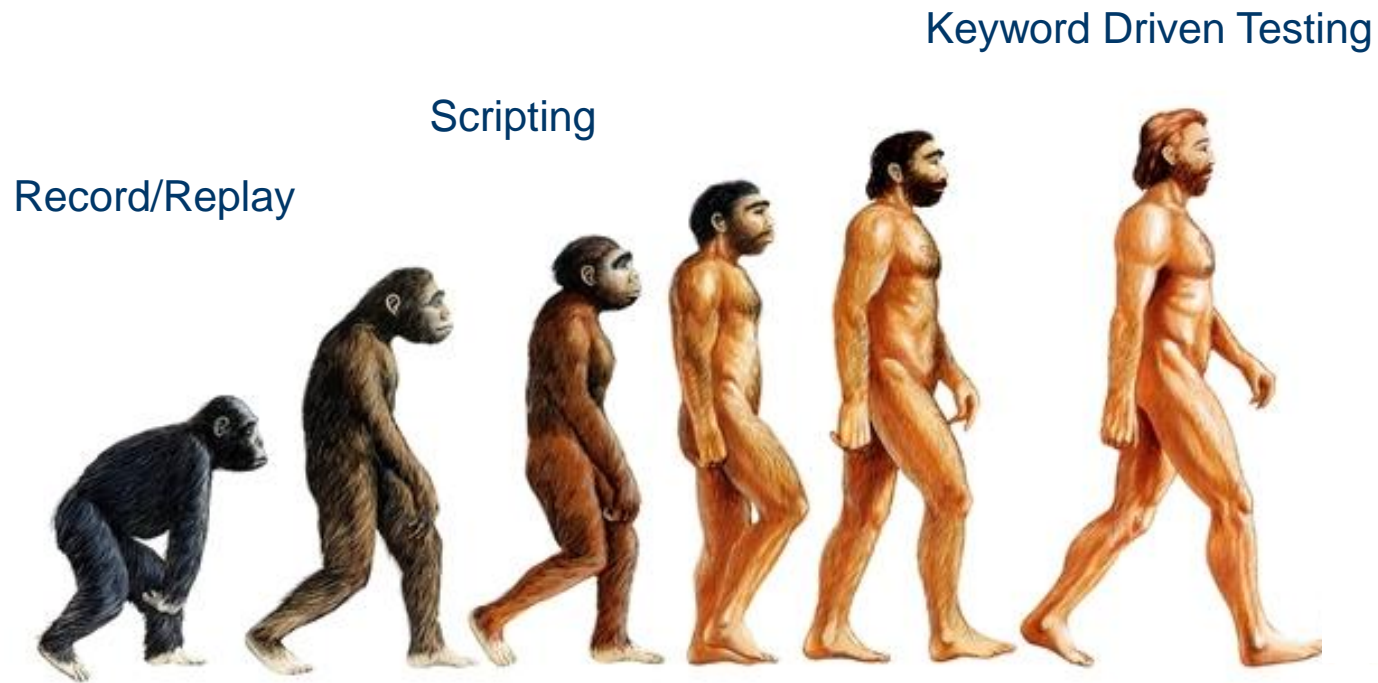


State-driven Testing: An Innovation in UI Test Automation

Dietmar Strasser
Sr. Manager, Testing - AMQ
dietmar.strasser@microfocus.com



Evolution of Test Automation





What is Keyword-driven Testing (KDT)

- KDT is a known and well accepted test automation technique that many mature development organizations use in order to overcome the disadvantages of simple record/playback test automation
- KDT provides a mechanism for mapping an Application Under Test (AUT) to a set of defined and well understood keywords
- Test designers, mostly non-technical people, assemble test cases using these keywords
- Test cases can be developed without programming knowledge



What is a Keyword?

- Keyword describes functional elementary actions

Low-level keyword (one action on 1 object), e.g. entering a username into a textfield.

Object	Action	Data
Textfield (username)	SetText	<username>

High-level keyword (a combination of other keywords in a meaningful unit), e.g. logging in.

Object	Action	Data
Textfield (domain)	SetText	<domain>
Textfield (username)	SetText	<username>
Textfield (password)	SetText	<password>
Button (login)	Click	One left click



Advantages of Keyword-driven Testing (KDT)

- KDT addresses the problem that Business Analysts usually do not have test automation expertise and Test Automation Engineers do not have domain knowledge
- Test scripts document the functionality of the AUT in a tabular format
- Separation of Test Design and Test Implementation
- Ease of maintenance
- Can be used for manual and automated testing
- Independent from UI driver



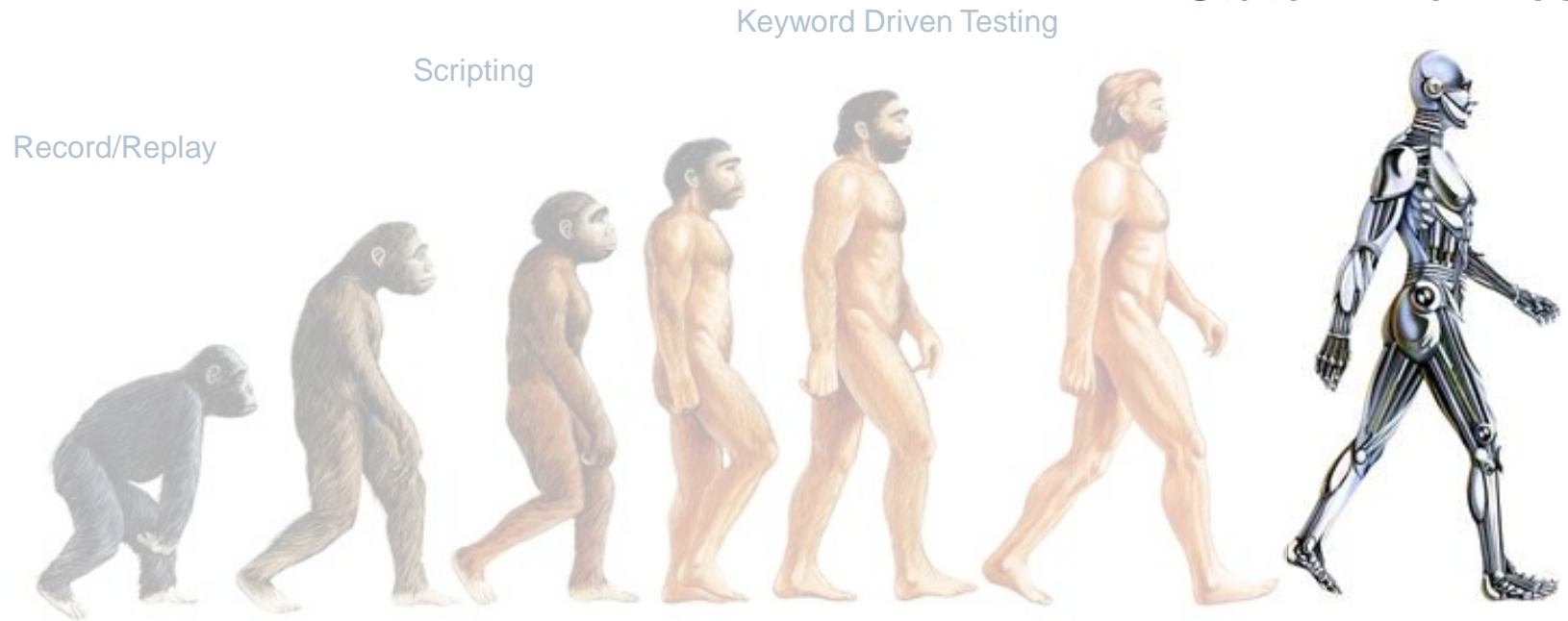
Shortcoming: Keyword-driven Testing

*“While keyword-driven sounds wonderful, it is not a magical methodology that will solve all automation problems and cure world hunger. I worked on a keyword-driven project while I was an employee of a big corporation. We had an elaborate in-house tool, that could compose the keywords into larger blocks of actions, which were also reusable in tests. **The project was a failure. The library of keywords became so huge that no one could figure out which keyword should be used in which context.**”*



Next Generation of Test Automation

State Driven Testing





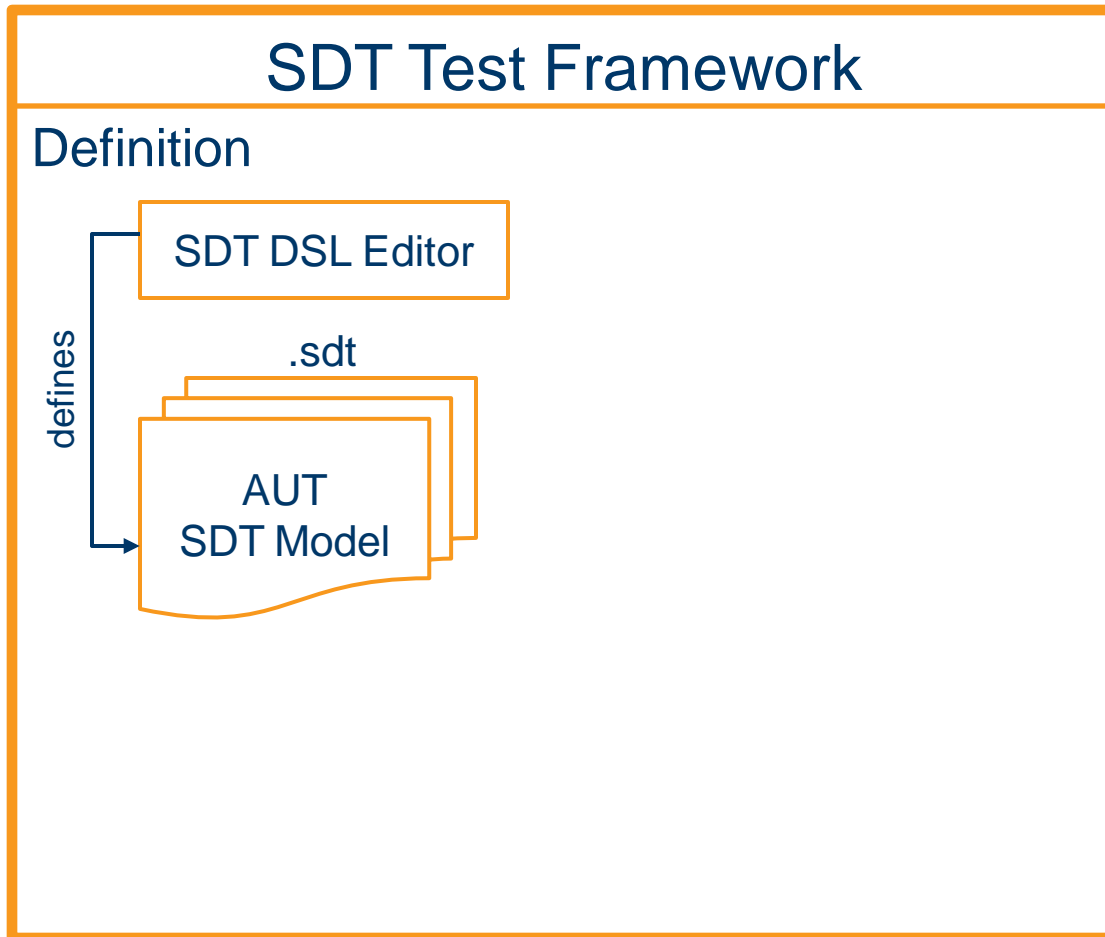
Overview

*State Driven Testing (SDT) =
Keyword Driven Testing + (UI) State Management*

- State Driven Testing (SDT) addresses exactly the issues of maintainability and complexity of Keyword Driven Testing by providing a **UI state transition model** as the core concept.
- By defining the state transitions of UI objects the set of allowed UI actions (keywords) at a specific point in a test script can be minimized to a few possible (tens) instead of all available (thousands).

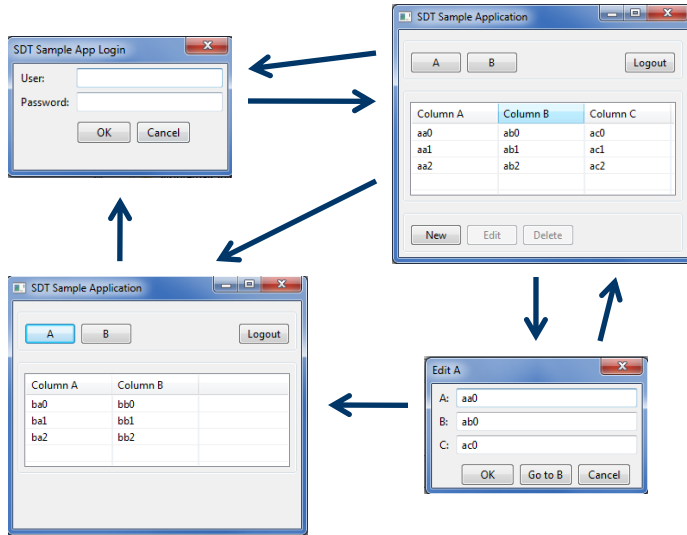


SDT Process – Framework Definition





AUT → SDT Framework Definition



```

TestObject Login
  DataField String mUserName = "admin"
  DataField String mPassword = "top_secret"

  TestMethod setUsernameAndPassword(String username="admin", String password="top_secret")
  TestMethod selectCancel() Returns Start
  TestMethod selectOk() Returns Main
  StateTransition
    NewAppState(Main)

TestObject Main
  TestMethod selectA() Returns AGrid
  StateTransition
    SetAppState(Main,AGrid)
  TestMethod selectB() Returns BGrid
  StateTransition
    SetAppState(Main,BGrid)
  TestMethod selectLogout() Returns Login
  StateTransition
    SetAppState(Login)
  
```

Define Test objects, test methods and state transitions using SDT DSL



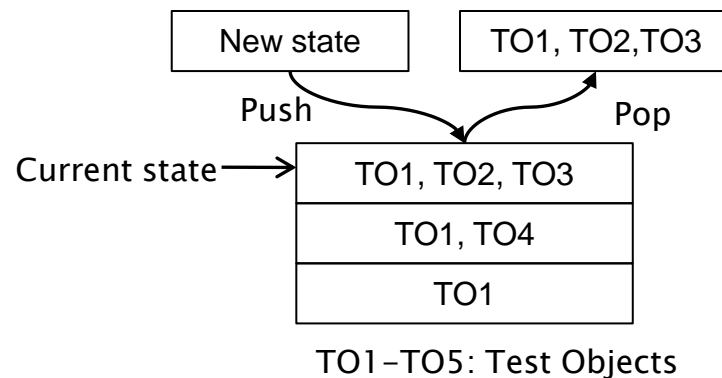
Definitions (1)

- Test object
 - A **Test object** consists of one to many Test methods that represent actions against the AUT (Application Under Test). Typically Test objects are used to structure the test framework so that all actions available for a specific UI container of the AUT like a dialog, a tree-view, a data-grid, a pane, a frame, or a menu are represented as methods of the Test object.
- Test method
 - Represents an action against an AUT like entering data, verifying response data or navigating in the application.
- State transition
 - A **State transition** is associated to a test method and defines how the accessible test objects (=application state) *change* after executing the test method. Multiple *state transition methods* can be used to change the application state.
- State transition methods
 - A **State transition method** can be used to change the application state.



Definitions (2)

- Application state
 - Represents the list of test objects that are accessible at a specific position in the test case.
- Application start state
 - Special application state represented by the list of Test objects which is used to describe the first possible interactions a user can do with an AUT.
- Application state stack
 - An **Application state stack** provides a mechanism to maintain multiple application states in a stack (a last in, first out (LIFO)). Therefore application states can be easily re-established to former states.





5 State Transition Methods

- **NewAppState (<list of test objects>)**
 - Creates a new application state representing the list of test objects provided as input, puts it on the application state stack and sets the current state to the newly created state.
- **RestoreAppState**
 - Removes the existing application state from the application state stack and activates the previous state.
- **AddAppState (<list of test objects>)**
 - Adds the list of test objects provided as input to the current application state. Former test objects of the current state are kept.
- **RemoveAppState (<list of test objects>)**
 - Removes the list of test objects provided as input to the current application state (if existing in the current application state)
- **SetAppState (<list of test objects>)**
 - Sets the current application state to the list of test objects provided as input. Former Test objects of the current state are deleted.

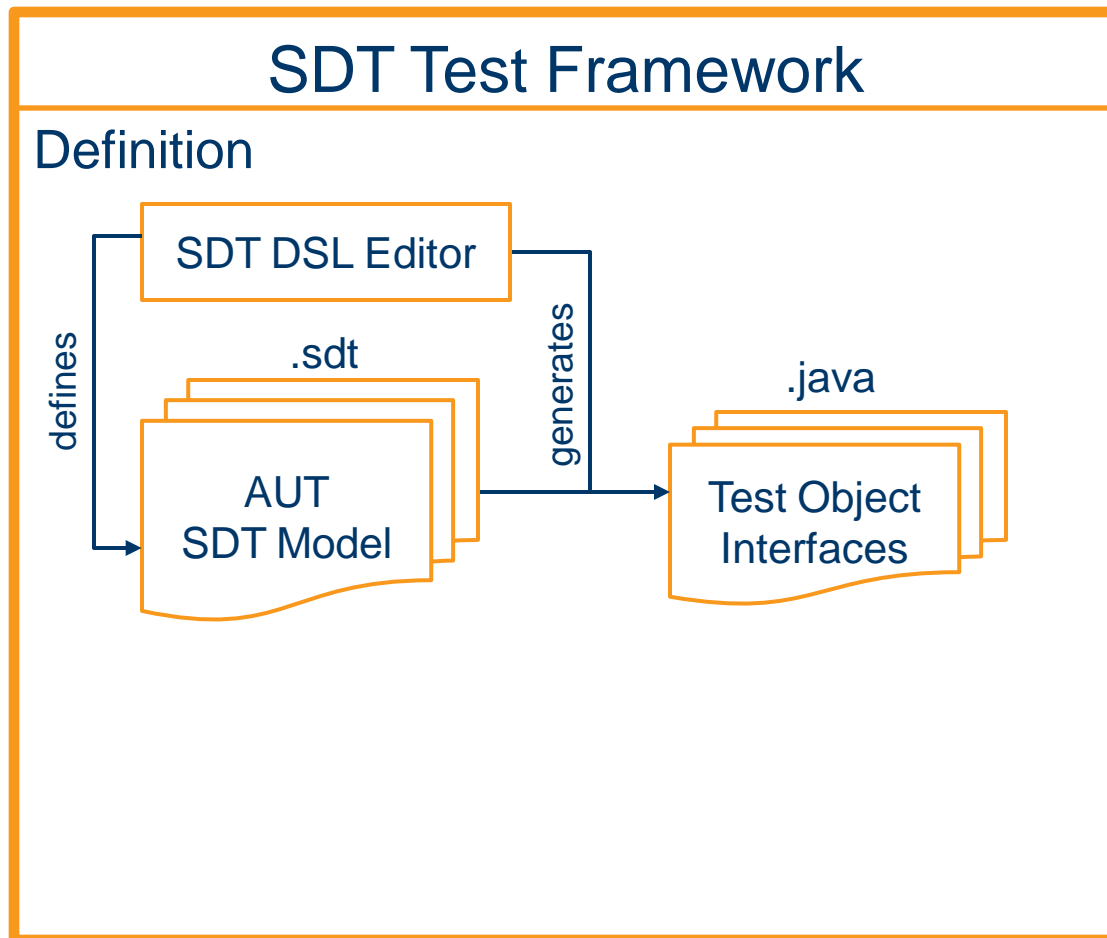


SDT Test Framework Structure

1 StartObject	
1..n TestObject	
0..1	Extends <Test Object>
0..n DataField	
	Data Types: Boolean, Integer or String
1..n TestMethod	
0..4	Parameters
	Data Types: Boolean, Integer or String; limited to <=4
0..1	Returns <Test Object> or <simple datatype value>
0..n StateTransition	
	5 different State Transition Methods (NewAppState, RestoreAppState, AddAppState, SetAppState or RemoveAppState)



SDT Process – Interface Code Generation





SDT Test Framework Code Generation

```
TestObject Login
  DataField String mUserName = "admin"
  DataField String mPassword = "top_secret"

  TestMethod setUsernameAndPassword(String username="admin", String password="top_secret")
  TestMethod selectCancel() Returns Start
  TestMethod selectOk() Returns Main
  StateTransition
    NewAppState (Main)

TestObject Main
  TestMethod selectA() Returns AGrid
  StateTransition
    SetAppState (Main, AGrid)
  TestMethod selectB() Returns BGrid
  StateTransition
    SetAppState (Main, BGrid)
  TestMethod selectLogout() Returns Login
  StateTransition
    SetAppState (Login)
```

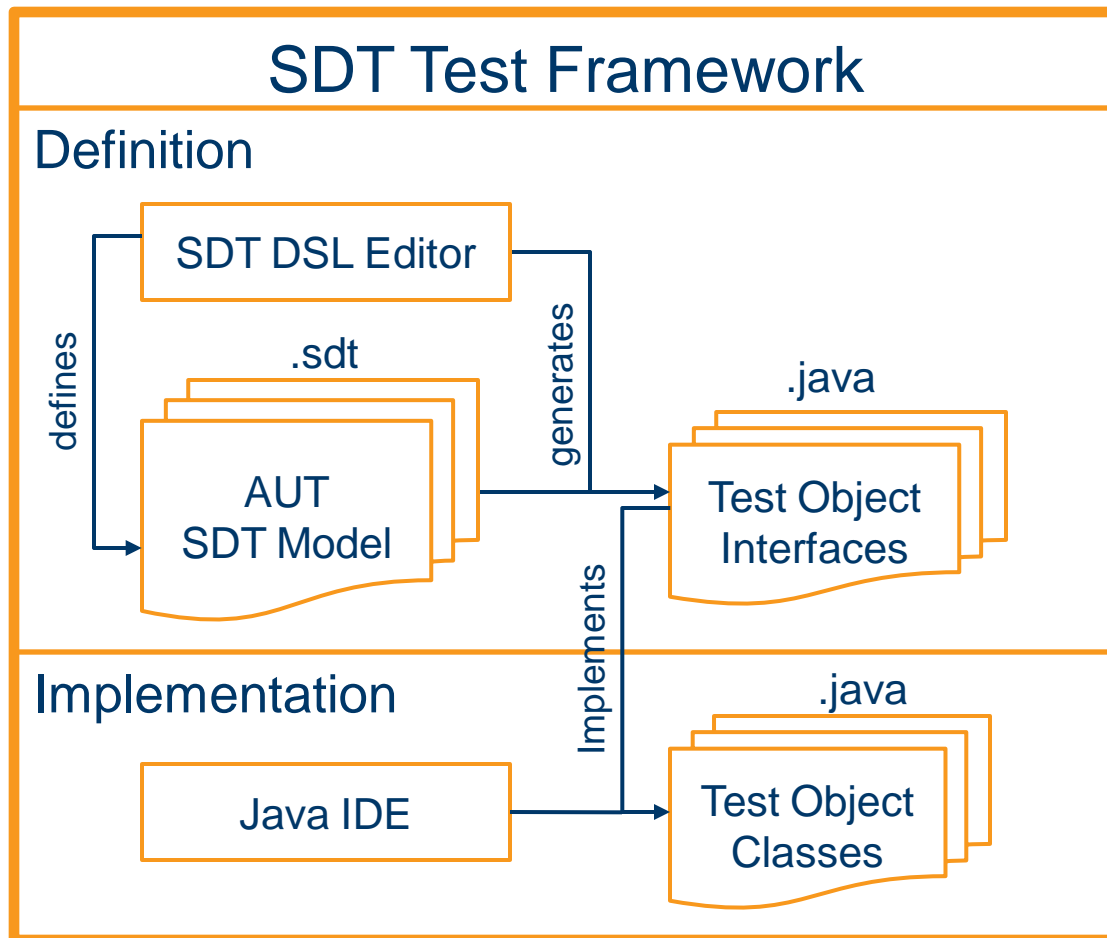


```
SDTTestFrameworkForSampleApp_v086
  src
    com.borland.silk.sdt.sample.testinterfacesDSL
      AAction.java
      AbstractDialog.java
      AEditDialog.java
      AGrid.java
      BGrid.java
      Login.java
      Main.java
      Start.java
      TestObjectsDSL.java
      Utils.java
    com.borland.silk.sdt.sample.testinterfacesDSL_impl
      AActionImpl.java
      AbstractDialogImpl.java
      AEditDialogImpl.java
      AGridImpl.java
      BGridImpl.java
      LoginImpl.java
      MainImpl.java
      SdtInit.java
      StartImpl.java
      UtilsImpl.java
```

SDT Framework Definition → Java Interface Classes

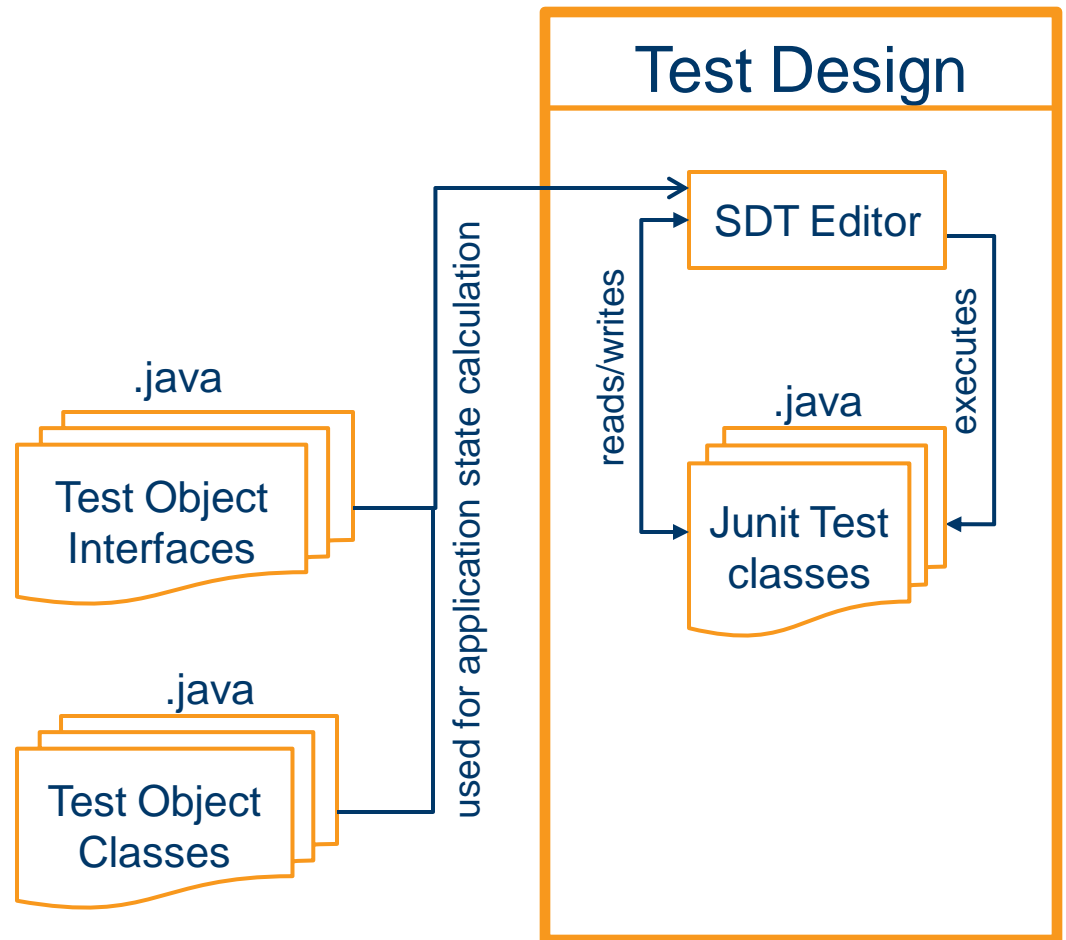


SDT Process – Framework Implementation





SDT Process – Test Design & Execution





Application State Engine

- An **Application state engine** calculates the application state (accessible Test objects) for a sequence of actions based on the application state transitions defined for the test methods in the test framework by using an application state stack.
- It calculates:
 - Which test objects are accessible when **appending** an action at the end of a sequence of actions.
 - Which test objects and test methods are accessible when **inserting** an action within a sequence of actions.
 - Which test objects and test methods can be **changed** for an existing test script step without breaking state transitions for succeeding actions.
 - Which consecutive sequence of actions can be **deleted** from a sequence of actions without breaking state transitions (a broken state transition causes actions that are not reachable through the state transitions of the predecessor actions).



SDT Test Design Editor

Test Script: testCase1

Script

No	Test Object	Test Method	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Return
1	Start	setUserNameAndPassword	user	pwd			
2	Start	selectOk					Main
3	Main	selectA					AGrid
4	AGrid	selectRow	1				AAction

Command

Line: 4

Test Object: AGrid

Test Method: Main

Row index: 1

Test Step Commands

Accessible Test Objects

+ Append - Delete Last 📄 Insert 🗑 Delete 🔍 Locate



SDT Test Case Example: “Create Manual Test”

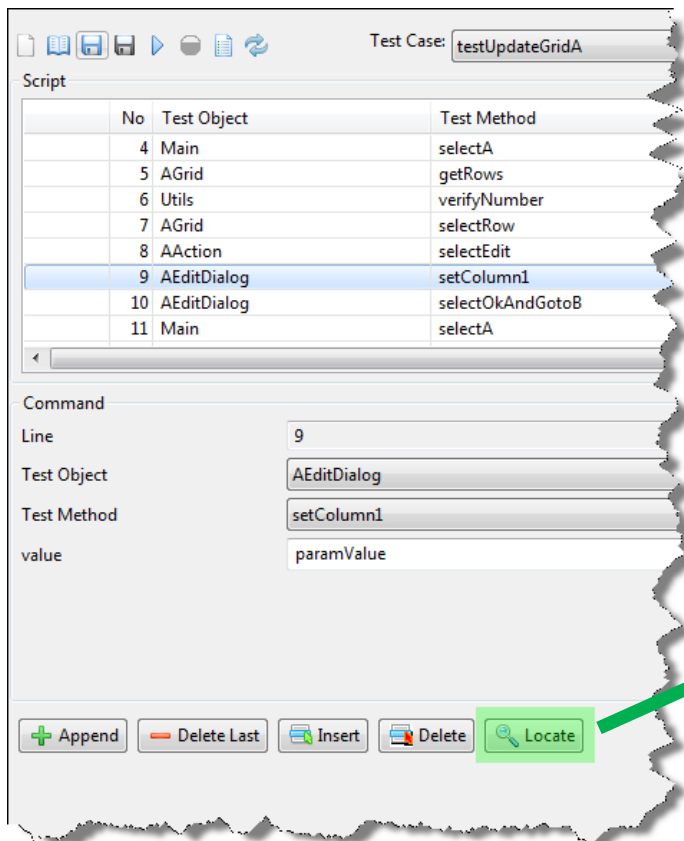
Test Case:

Script

No	Test Object	Test Method	Parameter 1	Parameter 2	Parameter 3	Parameter 4
✓ 1	TmStart	basestate_LoginAndSelectProj...	localhost:19120	admin	admin	Demo Project
✓ 2	TmMain	gotoTestPlan				
✓ 3	TpTree	selectFolderNodeByName	Application Access/Login			
✓ 4	TpFolder	openContextMenu				
✓ 5	TpFolderContextMenu	newChildTestDefinition				
✓ 6	TpTestDefinitionDlg	setName	MyManualTestDefinition1			
✓ 7	TpTestDefinitionDlg	setDescription	this is a description			
✓ 8	TpTestDefinitionDlg	nextManualTest				
✓ 9	TpManualTestPropertiesDlg	setPlannedTime	1:15			
✓ 10	TpManualTestPropertiesDlg	next				
✓ 11	TpNewTestStepDlg	setName	MyStepName1			
✓ 12	TpNewTestStepDlg	setActionDescription	My Action Description1			
✓ 13	TpNewTestStepDlg	setExpectedResults	My Expected Result			
✓ 14	TpNewTestStepDlg	ok				
✓ 15	TpTree	selectManualTestNodeByName	Application Access/Login/MyManual...			
✓ 16	TpManualTest	openContextMenu				
✓ 17	TpManualTestContextMenu	delete				
✓ 18	TmConfirmationDlg	yes				
✓ 19	TmMain	logout				



Press SDT Test Design Editor ,Locate' Button (Visual Test Object Location)



Test Case: testUpdateGridA

Script

No	Test Object	Test Method
4	Main	selectA
5	AGrid	getRows
6	Utils	verifyNumber
7	AGrid	selectRow
8	AAction	selectEdit
9	AEditDialog	setColumn1
10	AEditDialog	selectOkAndGotoB
11	Main	selectA

Command

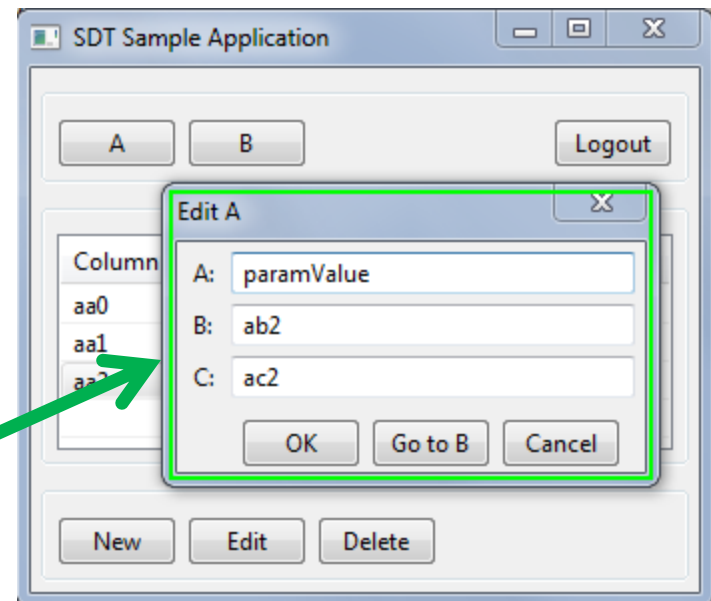
Line: 9

Test Object: AEditDialog

Test Method: setColumn1

value: paramValue

Buttons: Append, Delete Last, Insert, Delete, **Locate**



SDT Sample Application

Buttons: A, B, Logout

Dialog: Edit A

Columns: Column, aa0, aa1, aa2

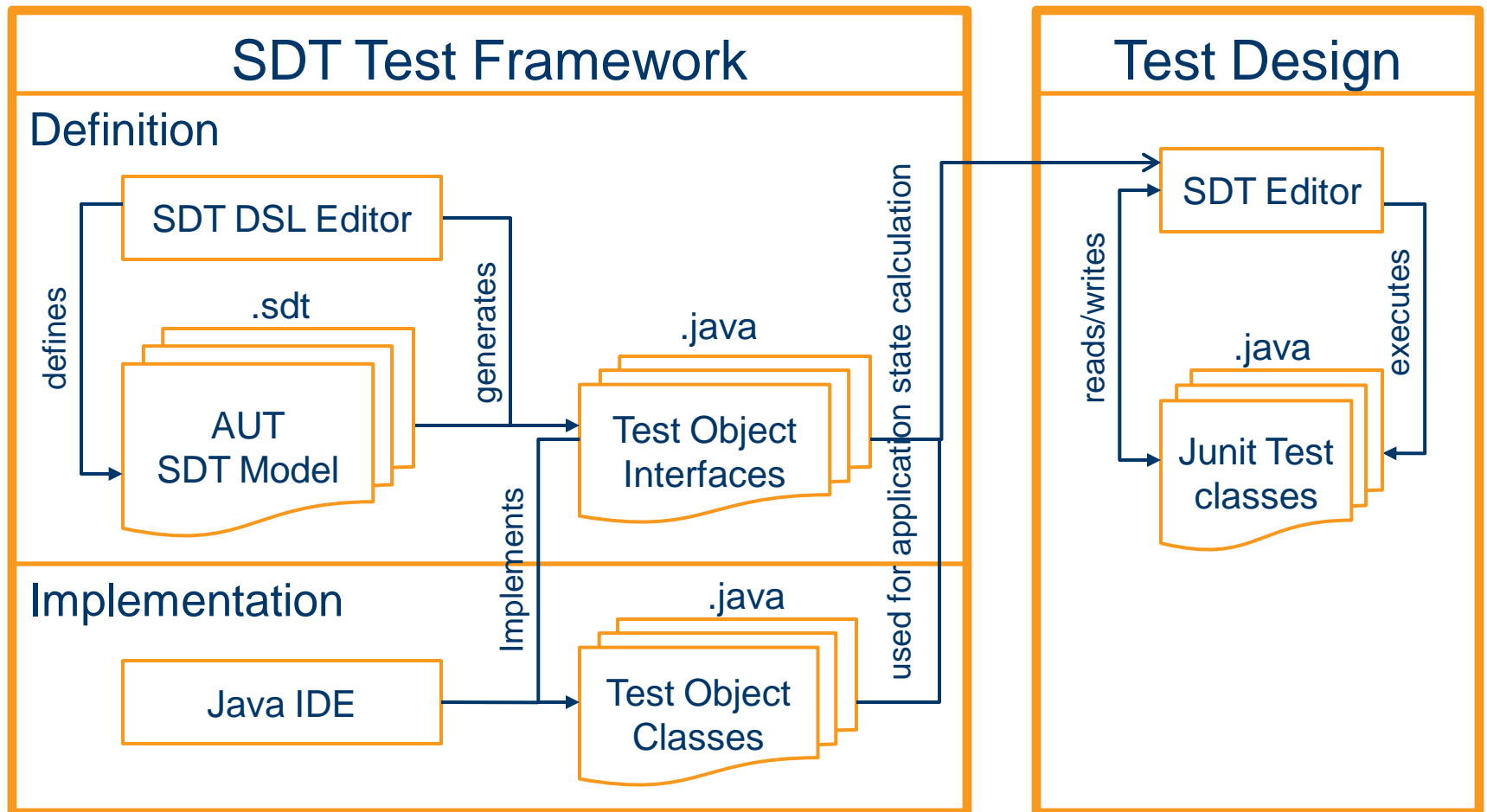
Fields: A: paramValue, B: ab2, C: ac2

Buttons: OK, Go to B, Cancel

Buttons: New, Edit, Delete

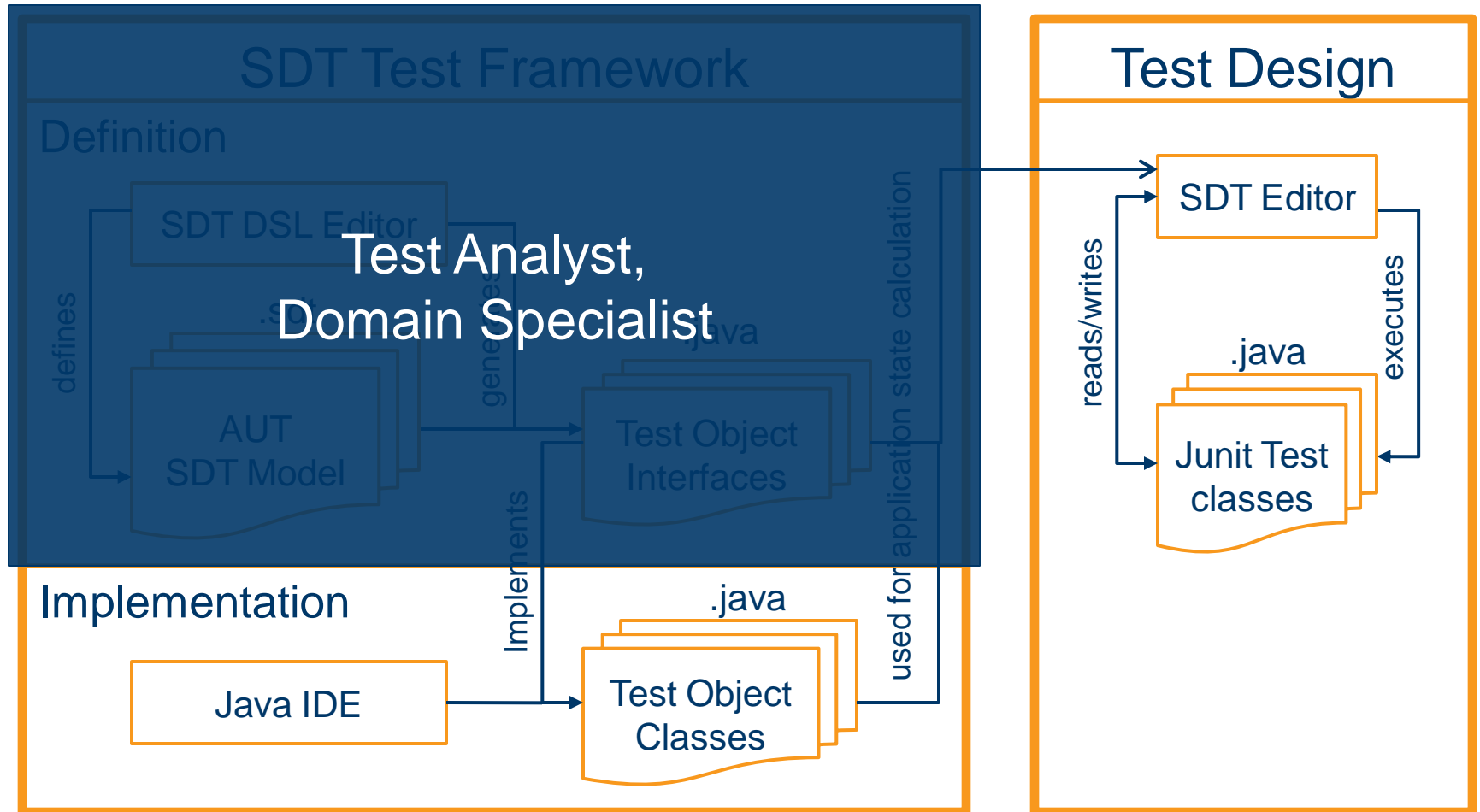


SDT Process



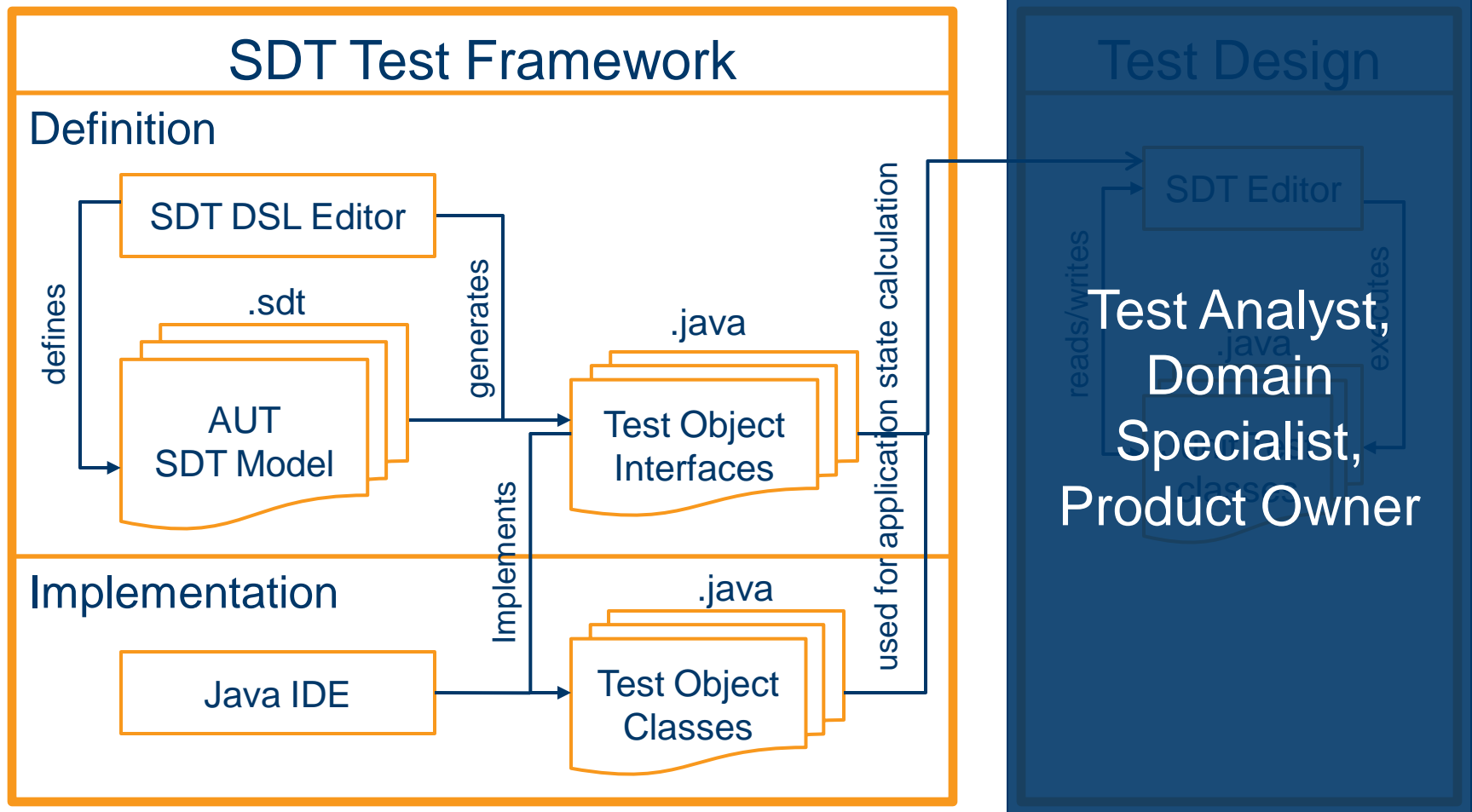


SDT Framework Definition - Persona



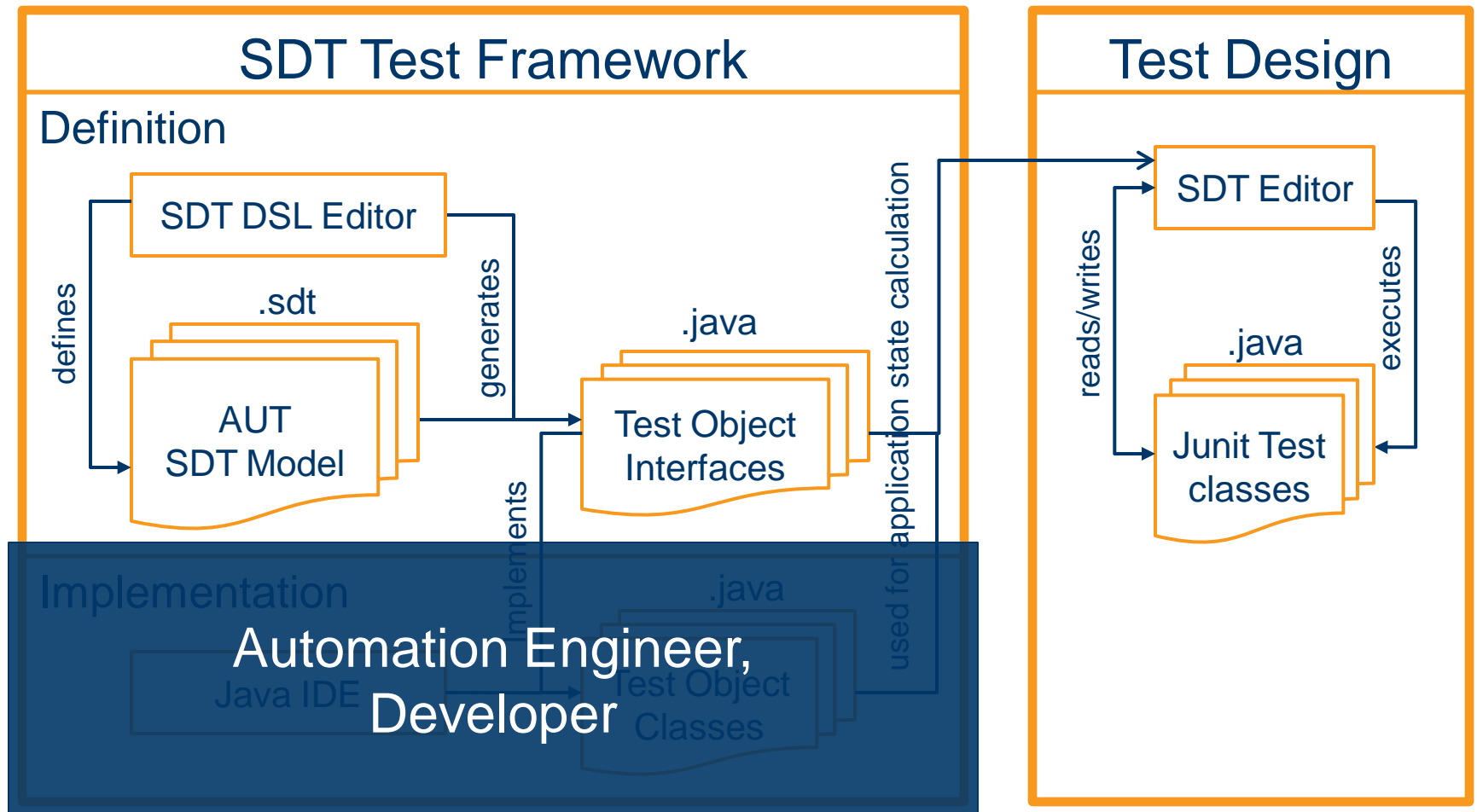


SDT Test cases - Persona





SDT Framework Implementation - Persona





10 Good Reasons to use SDT

1. Allows easy and efficient collaboration in a cross functional team, consisting of Business Analysts, developers and testers
2. Visual test object location mechanism allows easy extension of existing SDT Test Framework
3. Adaptable to different software development processes, like Agile, V-Model or pure waterfall
4. Clear, readable test cases usable as test documentation, which is always up-to-date
5. Possible to use same test case for manual, semi-automated and automated test execution
6. Supports test-driven development (create tests before GUI exists)
7. Consistent approach to create test cases on an integration, system and acceptance level
8. Test cases can be written by non-technical and technical people
9. Dramatically decreases costs of maintenance
10. Low or even no trainings effort to use SDT Test Framework for assembling test cases (just knowledge about AUT is needed)



Pilot Project Results

- Application Under Test
 - SilkCentral Test Manager (Test Management Suite)
 - Development effort spent for this product: > 50 person years
- SDT Framework
 - 370 Test objects
 - 3000 Test methods
 - ~90% of AUT functionality in SDT Test Framework defined
 - ~90% of all test objects implemented
 - Effort for Definition and implementation: 3 person months
- SDT Test suite
 - 103 test cases with code coverage of 53% running on 5 configuration (IE6, IE7, IE8, FF 3.0, FF3.5)



Q & A