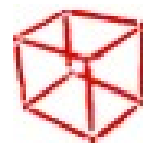




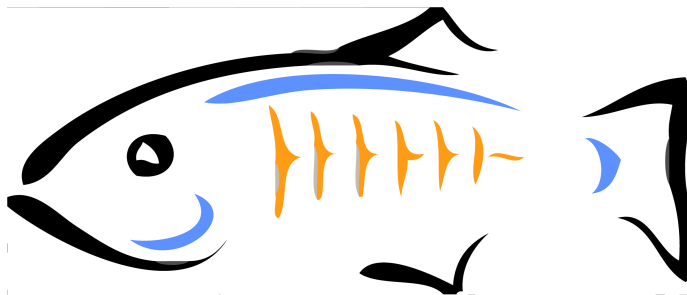
UI Test Automation

Alexandre (Shura) Iline
Java/FX SQE Tools Architect
SUN Microsystems
shura@sun.com

This presentation shares experience gotten from testing such products as



NetBeans



**Java FX
Designer**
No logo yet

Agenda

Test automation effectiveness

“To automate or not to automate”

UI specific

What's UI testing about

How to write tests

**Different approaches to encapsulate
logic into test library**

Tools

Requirements

Jemmy

The three things about automated tests

Inexpensive to execute.

That's good, but

Expensive to create

And, guess what ...

Require maintenance

'Cause ... they ... FAIL

Hmm ... need to optimize the overall value, then ...

Automation effectiveness constituents

T_M – time needed to run the tests manually

this is also a multiplier for other time values.

T_D – time needed for automated test development

which includes developing of tools, harnesses, etc.

T_S – time needed for automated test support

tests are needed to be updated if the product gets updated.

N_R – number of releases (test cycles)

vary a lot for different group of tests

N_C – number of tested configurations

OSs, external servers, etc.

Putting it all together ...
(next slide, please) ...

The formula

$$E_A = \frac{T_M * N_R * N_C}{T_D + T_S * N_R * N_C}$$

E_A – automation effectiveness

To be used for every particular product.

N_R and N_C are unique for a product.

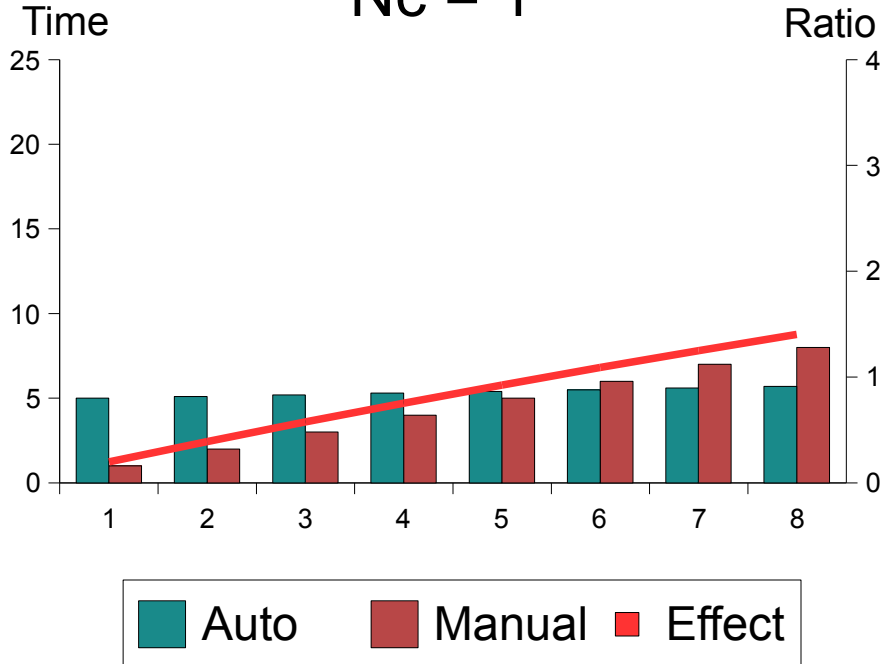
T_M is a characteristic of a test suite.

Smaller T_D and T_S - higher the E_A .

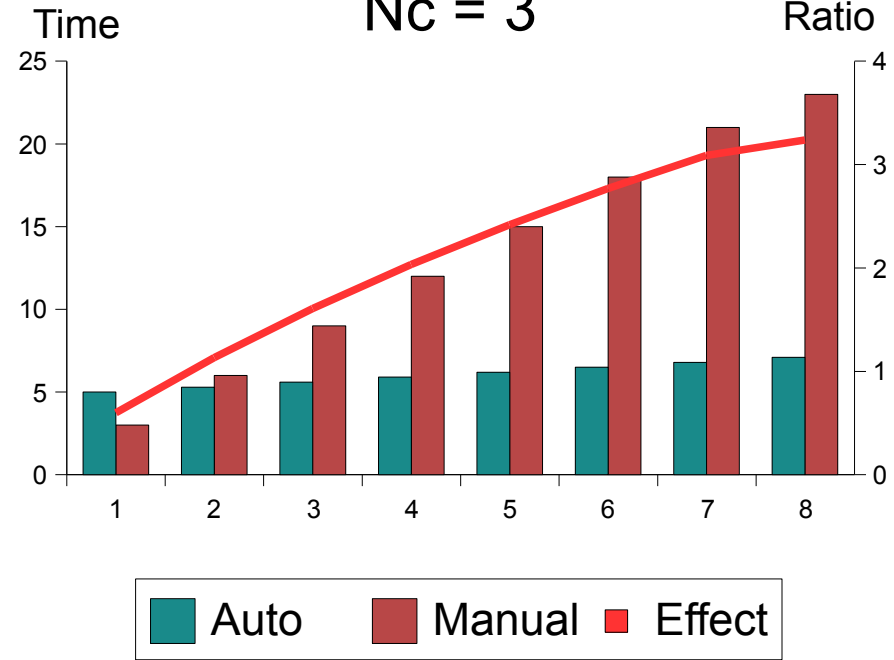
Some examples ...

Charts

$N_c = 1$



$N_c = 3$



Assumptions: $T_M = 1$ engineer*week

$T_S = 0.1 * T_M$ $T_D = 5 * T_M$ $N_R = 8$

T_d or T_s – what to minimize

T_s - if $(N_C * N_R)$ is big

Multi-platform

Compatibility with external products (servers, browsers, ...)

Long-living

Other words, when you want the testing to be repeated

T_D - if $(N_C * N_R)$ is small

Proof of concept

Preview

Just to run them several times for one release

Coding vs. recording

Scripting/coding

Tests are created manually.

T_D could be significant.

Record/Playback

User actions are recorded by some tool and stored for future use.

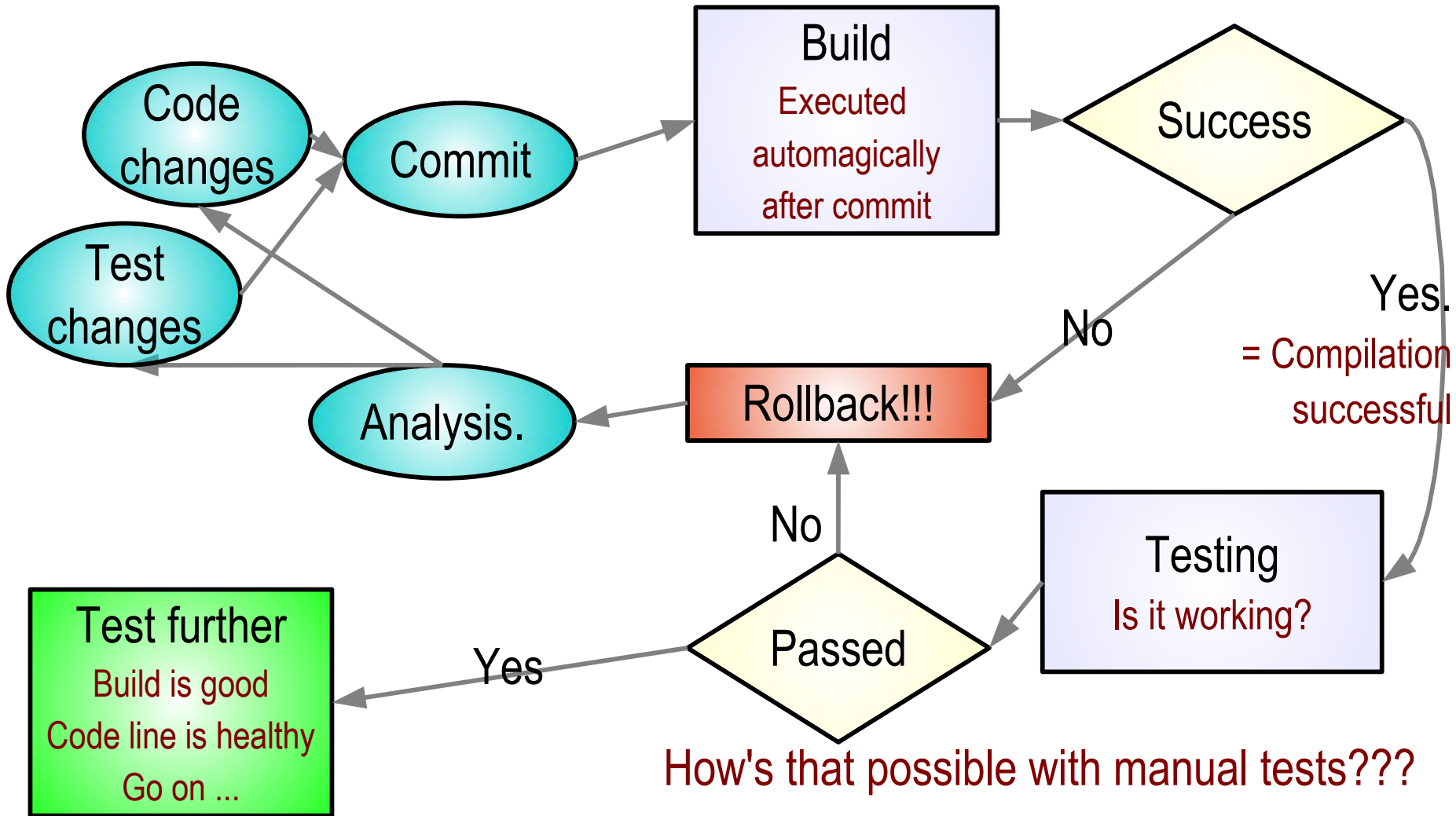
T_D is close to T_M , but T_S is usually big (for all tools known to authors).

Semi-automatic

Mix of the two above.

Is this all about ROI? **Not at all!**

Continuous build system:

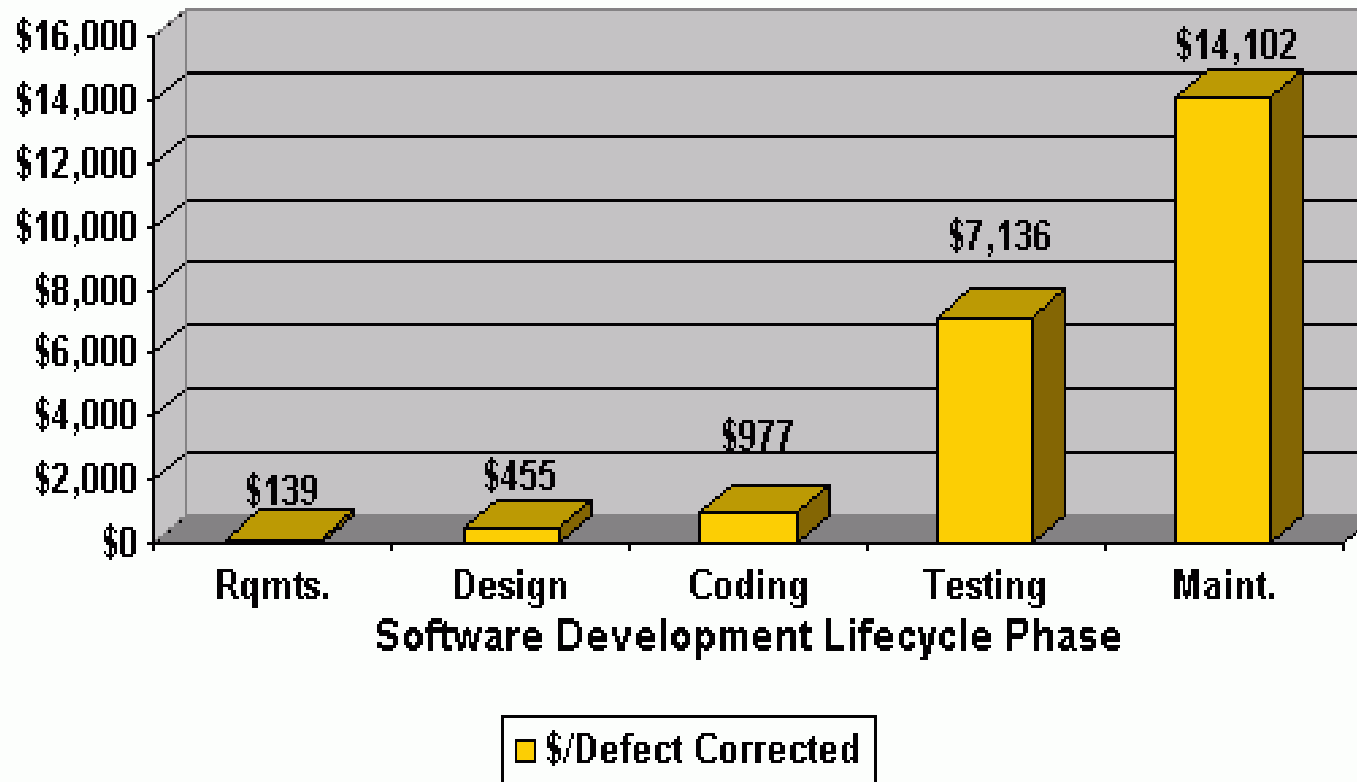


How's that possible with manual tests???

Earlier is better

Costs of Correcting Defects

Source: B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*



With continuous build, you find bugs at coding stage.

Performance && load tests

Are executed many times.

N_P – Number of times test is executed during performance/load test cycle

$$E_{AP} = E_A * N_P$$

What about UI



What's special about UI testing

UI test ...

Find the damn control!

Such as the “OK” button

Do something with it!

Push the damn thing

Check for the desired effect

Some other control is displayed

Some file changed

Something happened ...

Find the next control

Keep going ...

So, UI test automation is about

Control lookup

find that thingy over there

and

Human input simulation

scroll it or something

and

Result verification

check if scrolled

and

Waiting

For something to happen

Component lookup criteria

By ID

Easiest – may not be possible

By type

Most common

By index

Unavoidable

By toString()

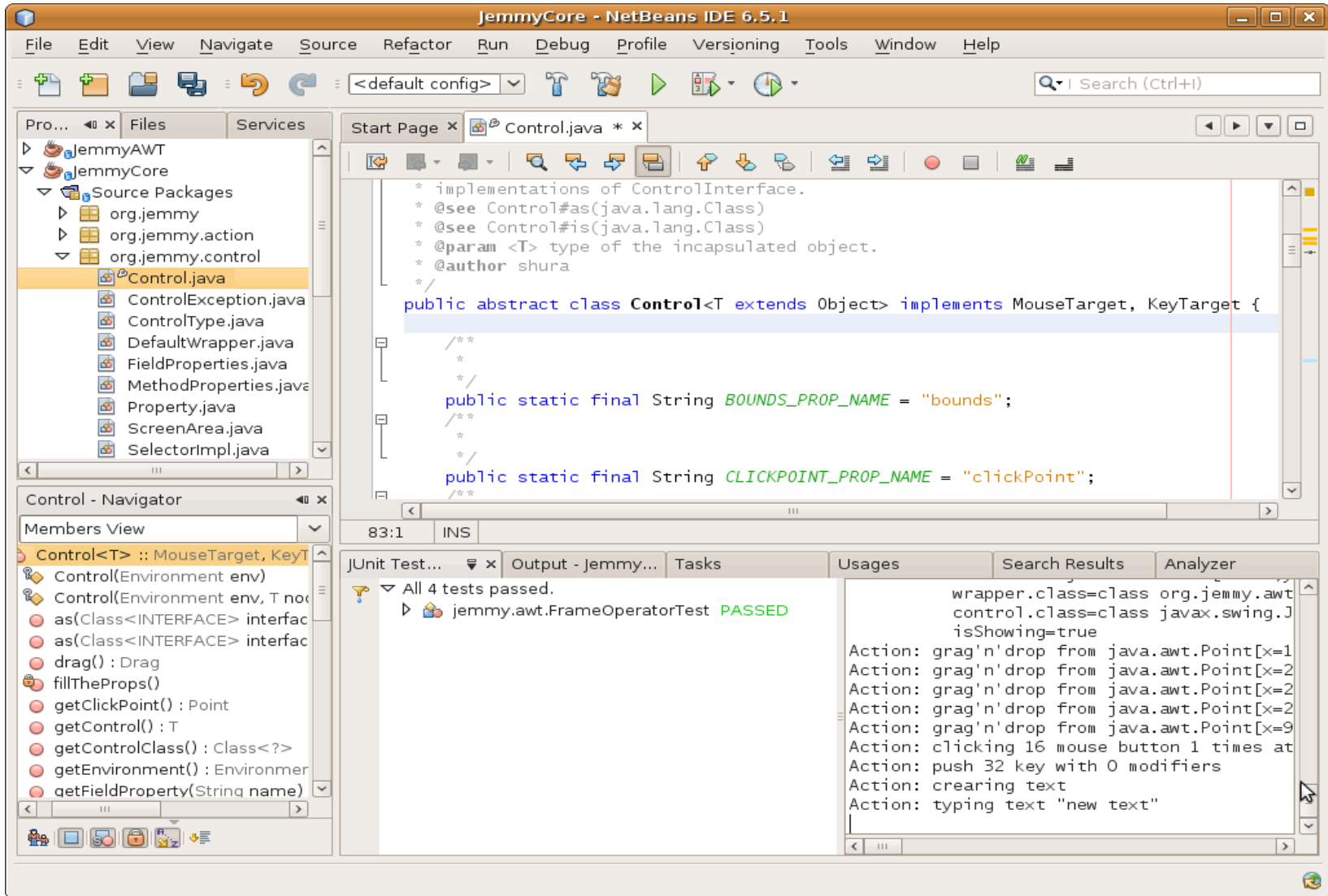
sucks

By text/tooltip/associated label

Best, if possible

*Are these all the search capabilities
you need?*

Oh, com'on!



The screenshot shows the NetBeans IDE interface for the 'JemmyCore' project. The main editor displays the source code for 'Control.java', which is an abstract class implementing 'MouseListener' and 'KeyListener'. The code includes annotations for documentation and two public static final strings: 'BOUNDS_PROP_NAME' and 'CLICKPOINT_PROP_NAME'.

```

* implementations of ControlInterface.
* @see Control#as(java.lang.Class)
* @see Control#is(java.lang.Class)
* @param <T> type of the encapsulated object.
* @author shura
*/
public abstract class Control<T extends Object> implements MouseTarget, KeyTarget {

    /**
     *
     */
    public static final String BOUNDS_PROP_NAME = "bounds";

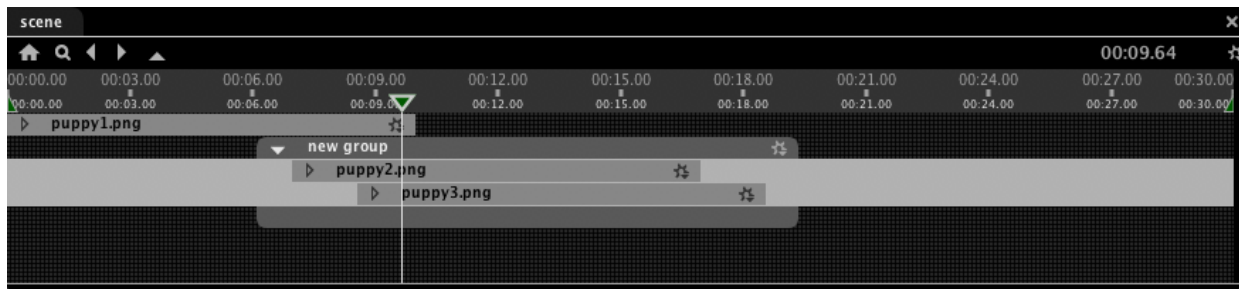
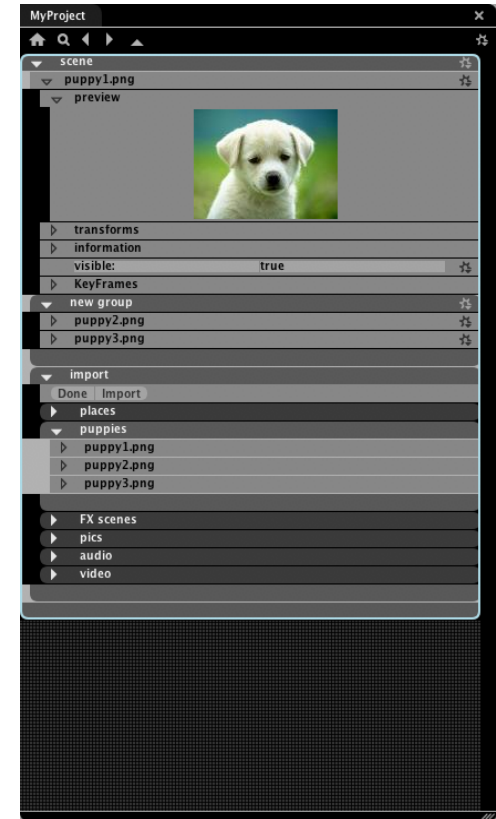
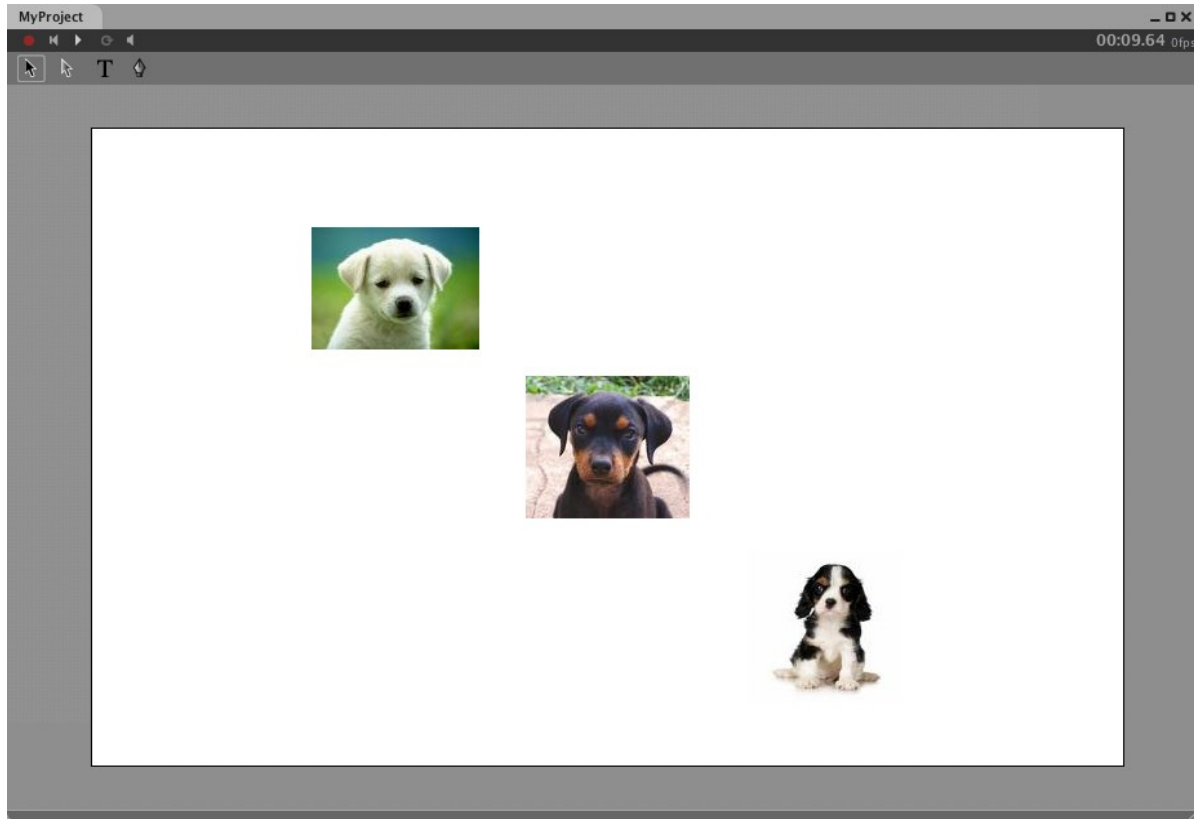
    /**
     *
     */
    public static final String CLICKPOINT_PROP_NAME = "clickPoint";

```

The 'Control - Navigator' window shows the 'Members View' for 'Control<T> :: MouseTarget, KeyTarget'. The members listed include methods like 'Control(Environment env)', 'as(Class<INTERFACE> interface)', 'drag(): Drag', 'fillTheProps()', 'getClickPoint(): Point', 'getControl(): T', 'getControlClass(): Class<?>', 'getEnvironment(): Environment', and 'getFieldProperty(String name)'.

The 'JUnit Test Results' window shows that all 4 tests passed, with a specific test 'jemmy.awt.FrameOperatorTest' marked as 'PASSED'. The 'Output - Jemmy...' window displays a log of actions performed during the test, such as 'Action: drag'n'drop from java.awt.Point[x=1]', 'Action: clicking 16 mouse button 1 times at', and 'Action: typing text "new text"'.

Oh, com'on! continues



And again ...

☕ **Java Store** alpha
catalog myapps about help

Featured Applications



SweetHome3D
SweetHome 3D



Face Bunny
Face Bunny Game



Duke Game
Duke Game



Installed
Puzzle
2D Puzzle Game



WhiteOut
WhiteOut



Duke Game
Duke Game



Calendar
Calendar



Calculator
Handheld Calculator



Installed
Beijing Opera
Beijing Opera Game



Video Puzzle
Video Puzzle

© 2009 Sun Microsystems, Inc.

Face Bunny



The Face Bunny game is a social networking game that let's you create your own customized bunny avatar. The game spawns new bunny faces each time you add a Facebook picture or messages...

Preview
Free

Install

Component lookup revisited

By .. whatever you want ...

```
class MySearchCriteria implements  
    SomeSearchCriteriaInterface {  
    public boolean theControlVerificationMethod  
        (ControlType control) {  
        return control.getSomeControlProperty().  
            equals(someThing);  
    }  
}
```

Input

Mouse

click(Point point, int button, int modifiers, int times)

press(int button, int modifiers)

release()

dragNDrop(Point from, Point to, int button, ...)

Keyboard

push(int keyCode, int modifiers)

press(int keyCode, int modifiers)

release()

Is this all you need?

Component specific

Text field

type(String text)

clear()

Check box

select(boolean)

Scroll bar

scroll(int value)

Table

select(int row, int column)

etc.

Verifying result

UI feedback

Dialog appeared

Text displayed

Object property

“white box”

External resource

file on disk

record in database

Timing ...

Test is executed in one thread

Application in ... many

Main application thread

Event queue thread(s)

Background threads

Modal dialogs

Loading something

etc

ANY test operation must be done with waiting!

Two options

Sleep **No good, 'cause**

Tests are slow

The timeouts are not big enough

Waiting

- For a control to
appear

- be ready for input

- have some property

- Event queue

- to be empty

- Some file on the disk to be updated

- etc.

UI test coding models



Remember the formula?

Application example

Different approaches

Summary

Remember the formula?

$$E_A = \frac{T_M * N_R * N_C}{T_D + T_S * N_R * N_C}$$

E_A – automation effectiveness

To be used for every particular product.

N_R and N_C are unique for a product.

T_M is a characteristic of a test suite.

Smaller T_D and T_S - higher the E_A .

Coefficient depend on the way you write your tests

T_s depends on the way tests are written

T_s mainly consists of time for test modification.

When product changes, tests need to be changed accordingly.

Many tests! Hundreds.

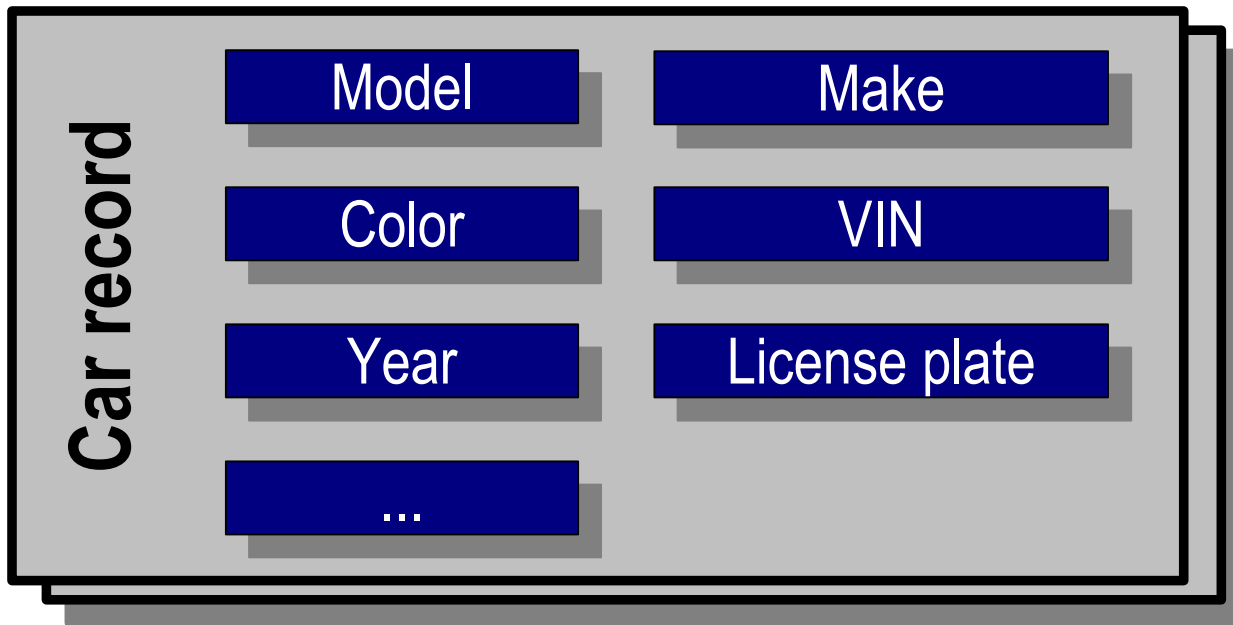
“Less changes of test code per a change in the product UI.”

Ideally ... no more than one.

But ... how? You are the coders – you know:

Move code to test library.

Application domain model



That's ... car catalog of some sort

Application UI

Product UI

Car records.

Make	Model	Year	Color	License Plate
Honda	Accord	2001	White	1abc234
Honda	Accord	2004	Black	5def678
Honda	Element	2006	Red	9ghi012
Subaru	Forester	2004	Green	3jkl456

Add Delete Properties

Subaru Forester 2004 Green 3jkl456

Make: Subaru

Model: Forester

Color: Green

Year: 2004 License Plate: 3jkl456

OK Cancel

Coordinates

```
click(134,32) //selects some record  
click(215,122) //hits "Properties"  
sleep(5) //sleeps to let dialog be painted  
click(64,182) //expands color combo  
click(235,182) //selects Gray  
click(235,212) //hit OK
```

Never tried, but ... $T_d \approx 1.1 * T_m$, $T_s \approx 1 * T_m$

Widgets

Find “Car records” frame

Find table

Select “1abc234” cell

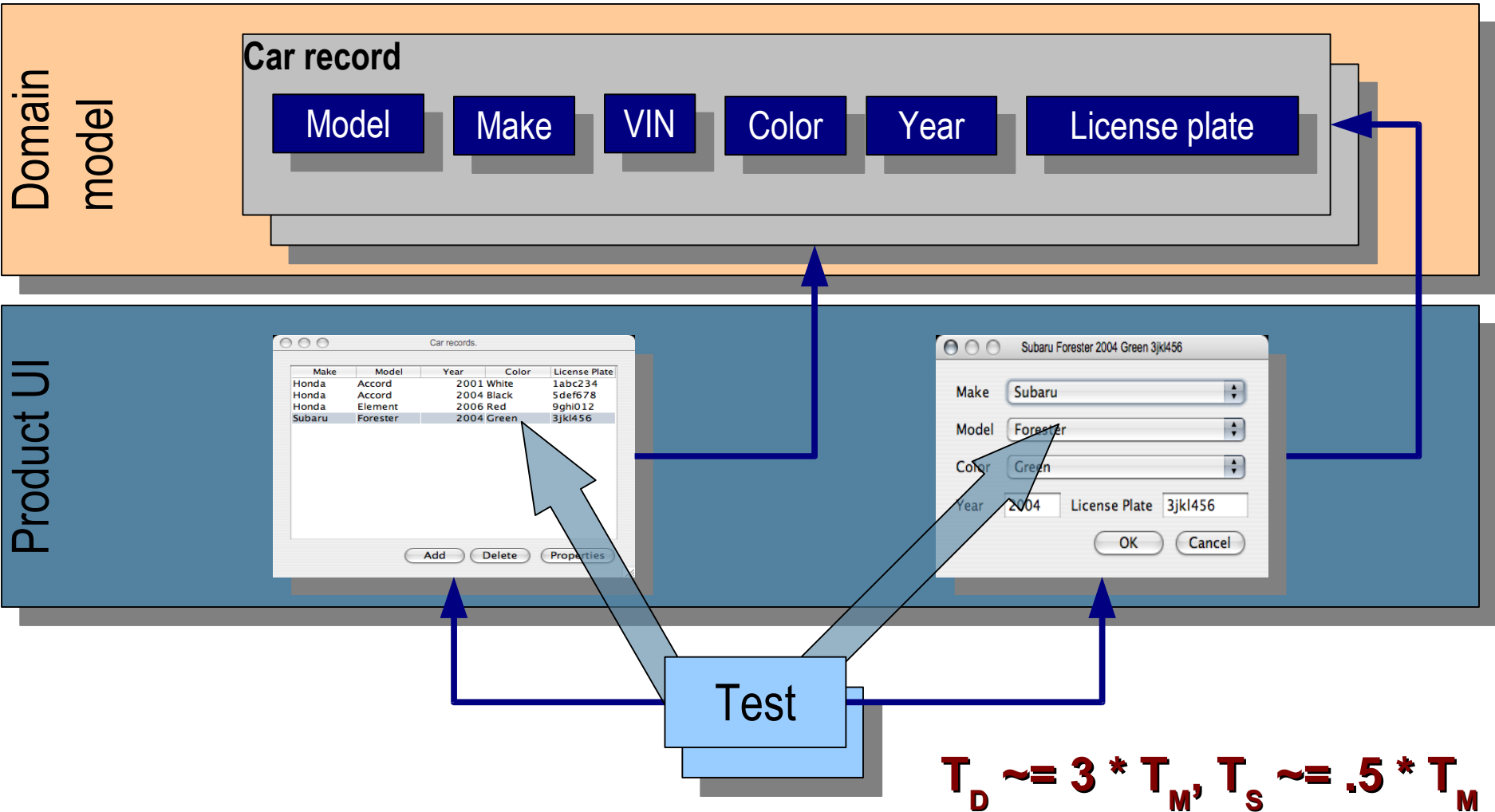
Push “Properties” button

Wait for “1abc234” dialog

Select “Gray” color in combo box

Push “OK”

Widgets or coordinates



UI Primitives

Find car list frame

```
CarListFrame list = new CarListFrame()
```

Open properties dialog for car "1abc234"

```
CarDialog propDialog =  
    list.carProperties("1abc234");
```

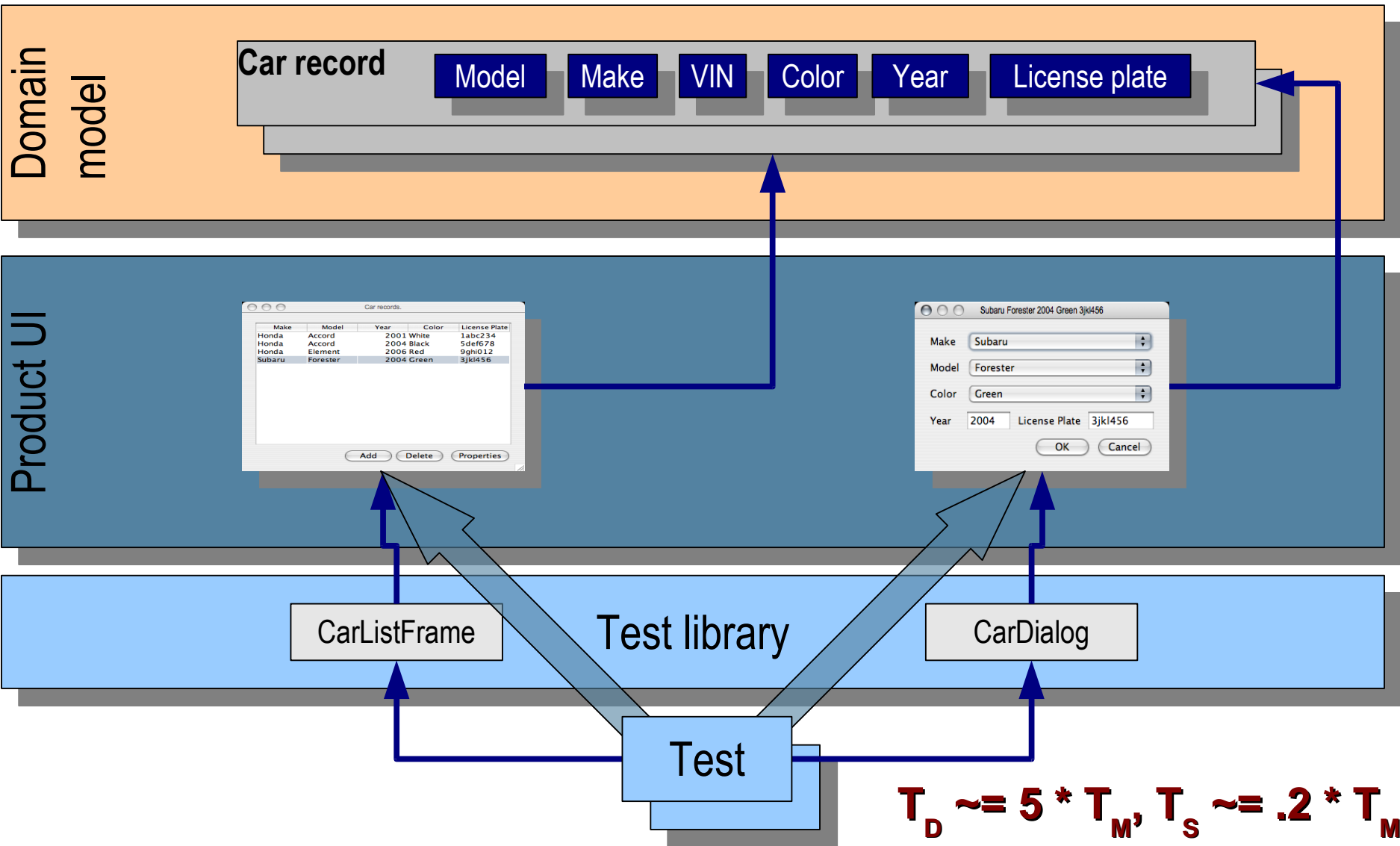
Set color to gray

```
propDialog.setColor(Color.GRAY);
```

Apply changes

```
propDialog.ok();
```

Library



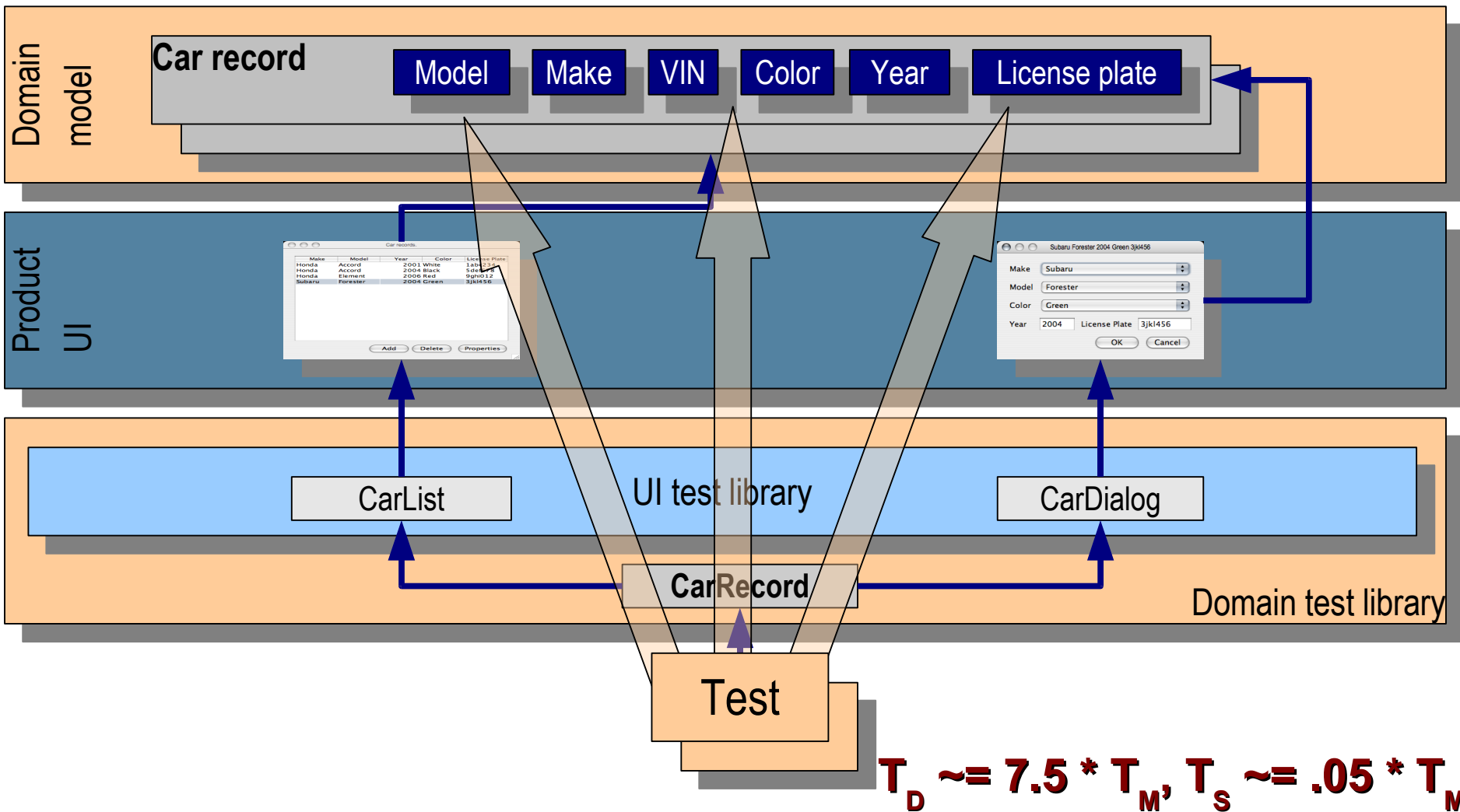
Domain model

```
Set color to gray for a car  
"1abc234"
```

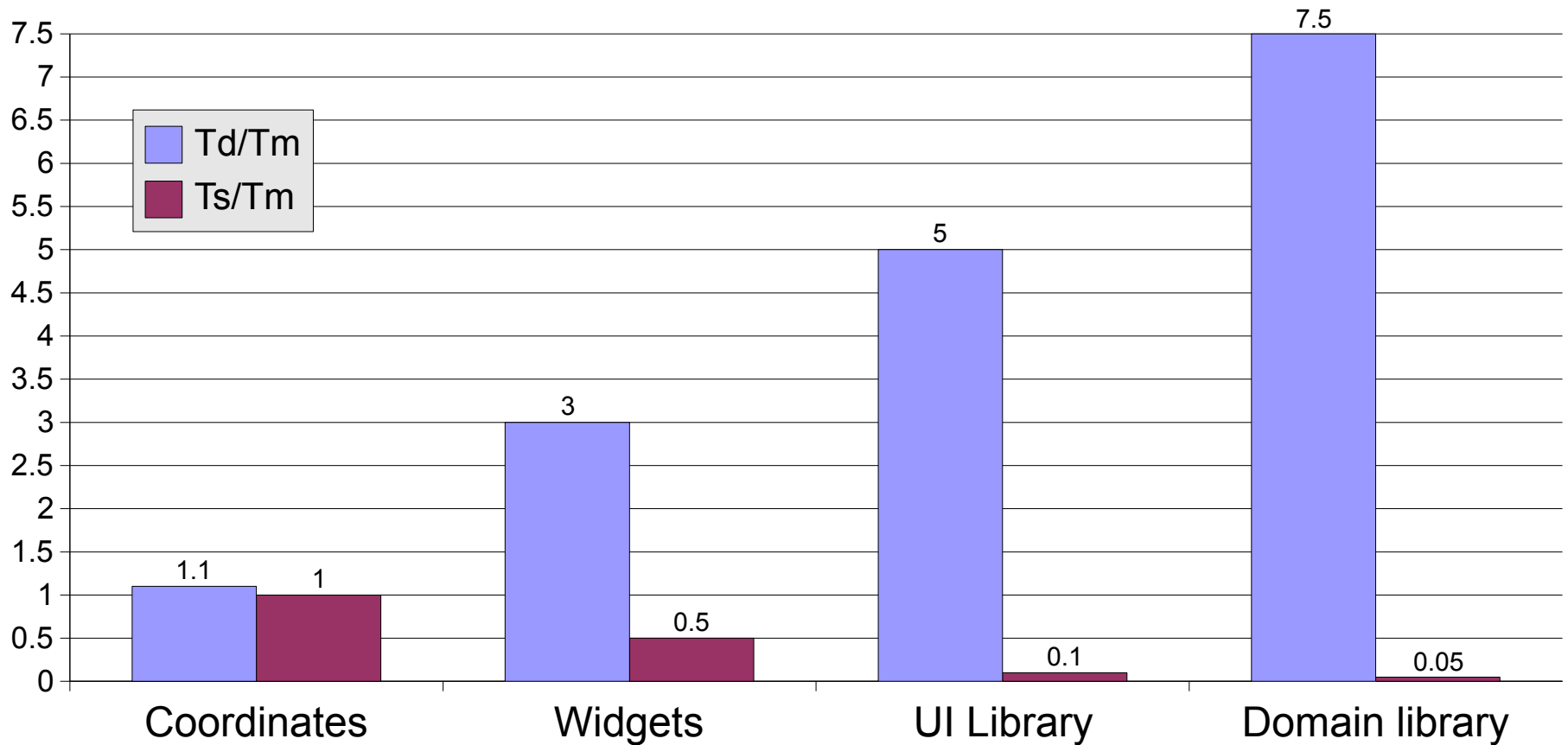
```
new CarRecord("1abc234") .  
    setColor(Color.GRAY);
```

Underneath the cover, **CarRecord** class does
all described earlier

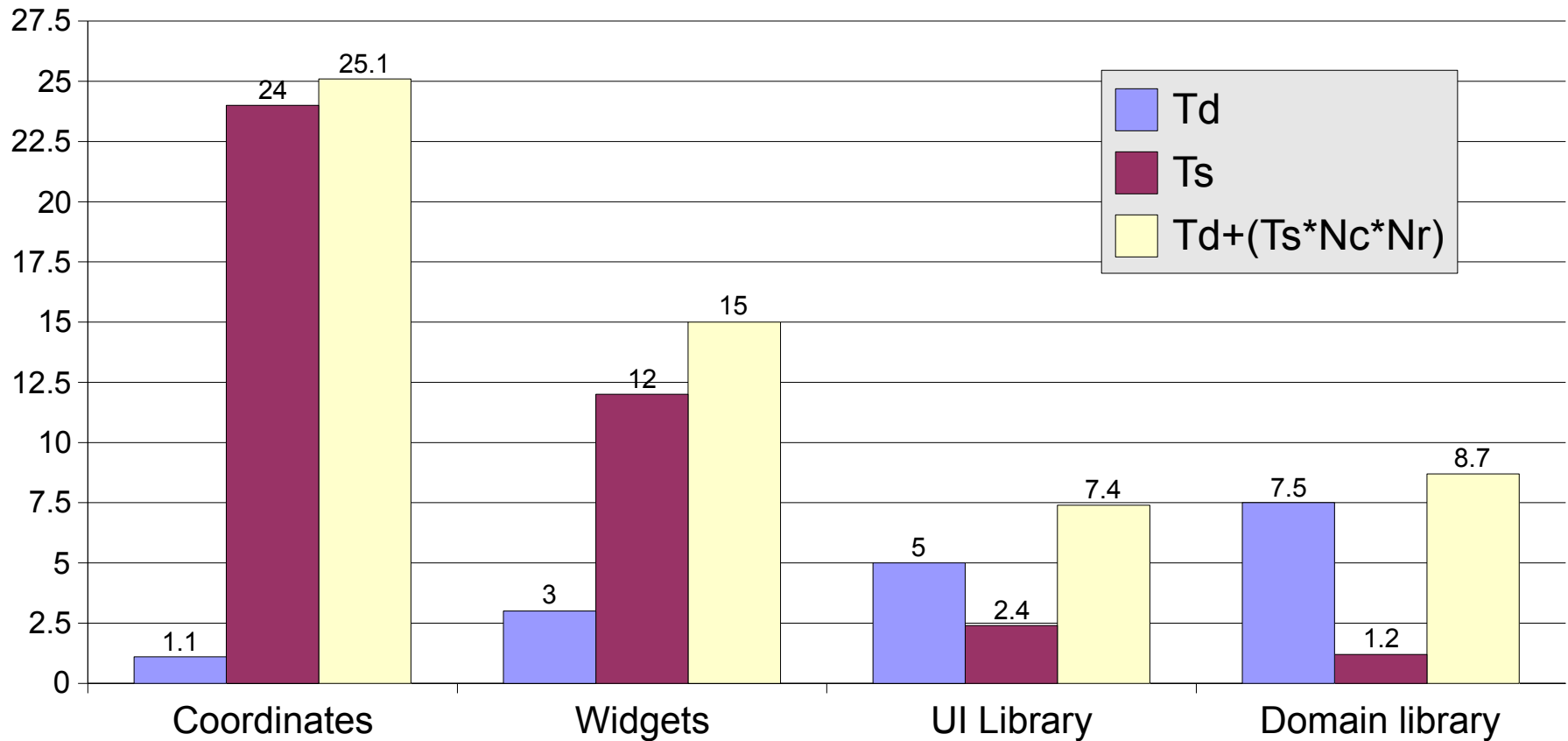
Domain library



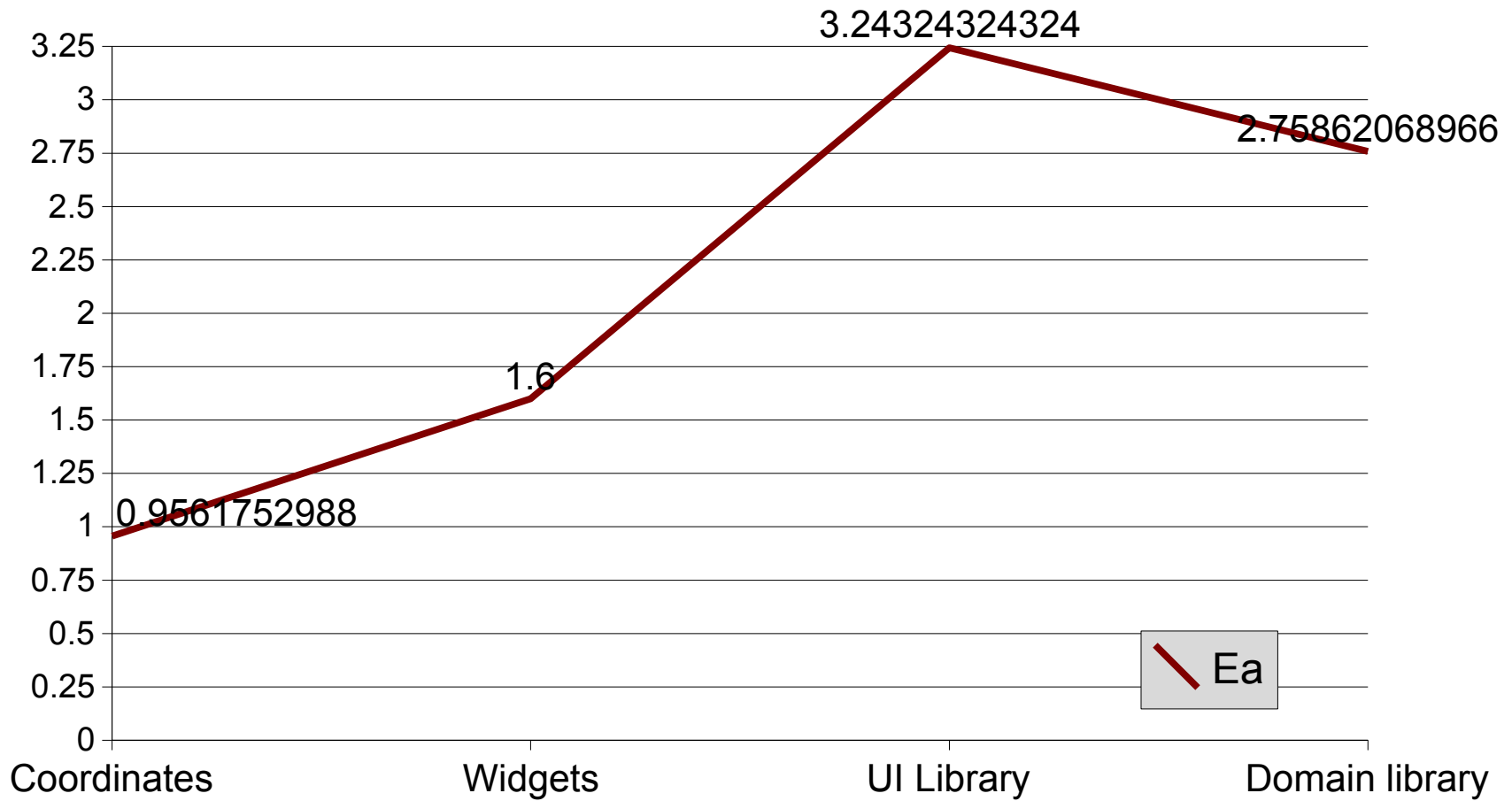
T_d and T_s together



T_D and T_S for $N_C=3$, $N_R=8$, $T_M=1$



E_A for $N_C=3, N_R=8$



Tools

Requirements

Jemmy



Tool requirements

Stability

If tests fail randomly, Ts is increasing for nothing

Power

You must be able to automate all the tests you need

Flexibility

You must be able to reuse tool API efficiently to build on top of it

Jemmy history

1999 Started as a tool for testing TeamWare UI

2000 Started to be used for NB modules

2000 Jemmy v2

2001 Accepted as NetBeans UI test tool

~2001 Made it's way to Java SQE

2002 Jellytools

2004 Java Studio Creator (Project Rave)

2008 Hosted on dev.java.net

2009 Jemmy v3

Jemmy v2 facts

Covers Swing/AWT

Tests written in Java

Used extensively within SUN

Open source

Very stable code

Operators in Jemmy v2

Wrapper around `java.awt.Component`
`public Component <?>Operator.getSource()`

Mimic Swing/AWT hierarchy

`JButtonOperator` extends `AbstractButtonOperator`
`JComponentOperator` extends `ComponentOperator`

Lookup constructors

ComponentOperator(ContainerOperator, ComponentChooser)

ComponentOperator(ContainerOperator, ComponentChooser, int)

JTextOperator(ContainerOperator, String)

JTextOperator(ContainerOperator, String, int)

JTableOperator(ContainerOperator, String, int, int)

JTableOperator(ContainerOperator, String, int, int, int)

JListOperator(ContainerOperator, String, int)

JListOperator(ContainerOperator, String, int, int)

etc

Convenient methods ...

ComponentOperator

clickMouse(...)

pushKey(...)

JTextOperator

type(String)

AbstractButtonOperator

push()

JTreeOperator

selectPath(String)

...

Jemmy 3

JemmyCore

UI library independent module

JemmyAWT

Small wrapper around Jemmy2

Reference implementation

JemmySG

Scenegraph

JemmyFX

Franca (limited)

Marina (coverage for `javafx.scene.control`)

Lookup

Interface LookupCriteria<CONTROL>

Boolean check(CONTROL)

ByTextLookup

To look by text

CoordinateLookup

By position

Parent and Lookup interfaces

Parent

Lookup<CONTROL> lookup(LookupCriteria<CONTROL>)

Search by criteria

Lookup<ST extends CONTROL> lookup(Class<ST>,
LookupCriteria<ST>)

Search by type and criteria

**Lookup<CONTROL> extends
Parent<CONTROL>**

void wait(int)

CONTROL get(int)

Control<CONTROL> control(int)

void dump(OutputStream)

int size()

Control<CONTROL> class

Wrapper

CONTROL getControl()

boolean is(Class<INTERFACE>)

Checks if the control could be treated as an interface

INTERFACE as(Class<INTERFACE>)

Treats the control as interface

Lookup

```
Parent<MyControl> parent = ...;
```

```
LookupCriteria<MyControl> lookup1 = ...;
```

```
//MySubControl extends MyControl
```

```
LookupCriteria<MySubControl> lookup2 = ...;
```

```
parent.lookup(lookup1).size(); //how many
```

```
parent.lookup(lookup1).get(0); //first
```

```
parent.lookup(lookup1).control(0); //wrap
```

```
//narrow down the search further.
```

```
parent.lookup(lookup1).lookup(MySubControl.class, lookup2).
```

```
control(0);
```

Interfaces

Parent – seen already

Mouse

click(...), press(...), release(...)

Keyboard

push(...), press(...), release(...)

Drag

dnd(...)

Scroll

For scroll bars, sliders, spinners

Selectable

For check boxes, radio buttons

Window

For anything resize-able and movable.

Interfaces cont.

```
Control<MyControl> control = ...;
```

```
//basic input is built in
```

```
control.mouse().click(); // same as control.as(Mouse.class).click();
```

```
control.drag().dnd(...);
```

```
//easy to use other functionality
```

```
control.as(Scroll.class).scroller().scrollTo(50);
```

```
//here it gets interesting ...
```

```
control.as(Parent.class, MyControl.class).lookup(lookup1).control(0).
```

```
    mouse().click();
```

Lookup revisited

```
Parent<Scene> allScenes = ...;
```

```
allStages.lookup(new ByTitleSceneLookup("My window")).
```

```
control(0).
```

```
as(Parent.class, Node.class).
```

```
lookup(CheckBox.class, new ByTextLookup("check box")).
```

```
control(0).
```

```
as(Selectable.class, Boolean.class).
```

```
selector().select(true);
```

Demo



Are you still alive?

Wanna know more?

<http://jemmy.dev.java.net>