UT-BATTELLE

Unit Testing, Sequence Diagrams

There is no free lunch.

July 2006

Kay-Uwe Kasemir

kasemirk@ornl.gov

SNS SPALLATION NEUTRON SOURCE

OAK RIDGE NATIONAL LABORATORY
U. S. DEPARTMENT OF ENERGY

"Regression" or "Unit" Tests

- **Good Stuff!**
  - **Initially:**
    Know that your code works to at least some degree.
  - **After "improvements":**
    Re-run tests to verify that nothing broke.

How to perform a Test

1. **Have something to test**

   Example: A text parser, String => Integer.

2. **Implement the test**

   Parse "10",
   compare with expected 10, …

3. **Run the test**

4. **Check output, fix, try again**

Test-Driven Design, "Extreme" Programming

1. Implement the test

Design the API while you're actually trying to use it!

2. Have something to test

Implement what the test requires.

3. Run the test

4. Check output, fix, try again

Keep coding until tests pass.
+ Implement what's really needed.
+ Small, testable code units typically more re-usable than big, monolithic applications.

… Have Something To Test …

- **For this presentation: Simple Number Parser**
  - **java.lang.Integer.parseInt(String)**

.. Implement the Test ..(JUnit Example)

```
public class NumberParserTest extends TestCase
{
    public void testNumbers()
    {
        Assert.assertEquals(0,      Integer.parseInt("0"));
        Assert.assertEquals(2,      Integer.parseInt("2"));
        Assert.assertEquals(2545,   Integer.parseInt("2545"));
        Assert.assertEquals(-2,     Integer.parseInt("-2"));
        Assert.assertEquals(-34564, Integer.parseInt("-34564"));
    }

    public void testRandom()
    {
        for (int i = 0; i < 1000; i++)
        {
            int num = (int)(Math.random()*10000-5000);
            Assert.assertEquals(num, Integer.parseInt(String.valueOf(num)));
        }
    }

    public void testErrors()
    {
        try
        {
            Integer.parseInt("No Number");
            Assert.fail("Should not have parsed that....");
        }
        catch (NumberFormatException e)
        {
            Assert.assertNotNull("No Exception?", e);
        }
    }
}
```

- Check
  - known cases,
  - random cases,
  - Errors
- JUnit, CppUnit, MatUnit, … even ANSI C lib offer similar

  assert(…)

  mechanisms.

.. Run the Test ... (Build Test Suite)

NumberParserTest.java | Main.java

```
package test;

import junit.framework.TestSuite;

public class Main
{
    public static void main(String[] args)
    {
        TestSuite suite = new TestSuite();
        // Make sure that you add all your test classes here:
        suite.addTestSuite(NumberParserTest.class);
        // ------------------------------------------------
        junit.textui.TestRunner.run(suite);
    }
}
```

Console

```
<terminated> Main [Java Application] /System/Library/Frameworks/JavaVM.framework/V
...
Time: 0.046

OK (3 tests)
```

.. Run the Test, find errors ...

NumberParserTest.java | Main.java

```
package test;

import junit.framework.Assert;
import junit.framework.TestCase;

/** A test of the number parser. */
public class NumberParserTest extends TestCase
{
    public void testNumbers()
    {
        Assert.assertEquals(0,      Integer.parseInt("0"));
        Assert.assertEquals(2,      Integer.parseInt("2"));
        Assert.assertEquals(2545,   Integer.parseInt("2545"));
        Assert.assertEquals(-1,     Integer.parseInt("-2"));
        Assert.assertEquals(-34564, Integer.parseInt("-34564"));
    }
```

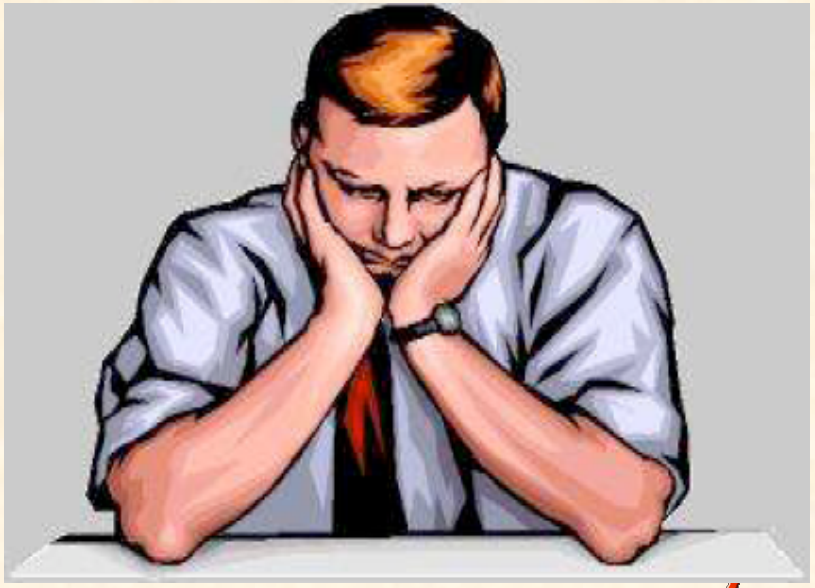
Console

```
<terminated> Main [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (Jul 5,
.F..
Time: 0.058
There was 1 failure:
1) testNumbers(test.NumberParserTest)junit.framework.AssertionFailedError: expected:<-1> but was:<-2>
        at test.NumberParserTest.testNumbers(NumberParserTest.java:14)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at test.Main.main(Main.java:13)

FAILURES!!!
Tests run: 3,  Failures: 1,  Errors: 0
```

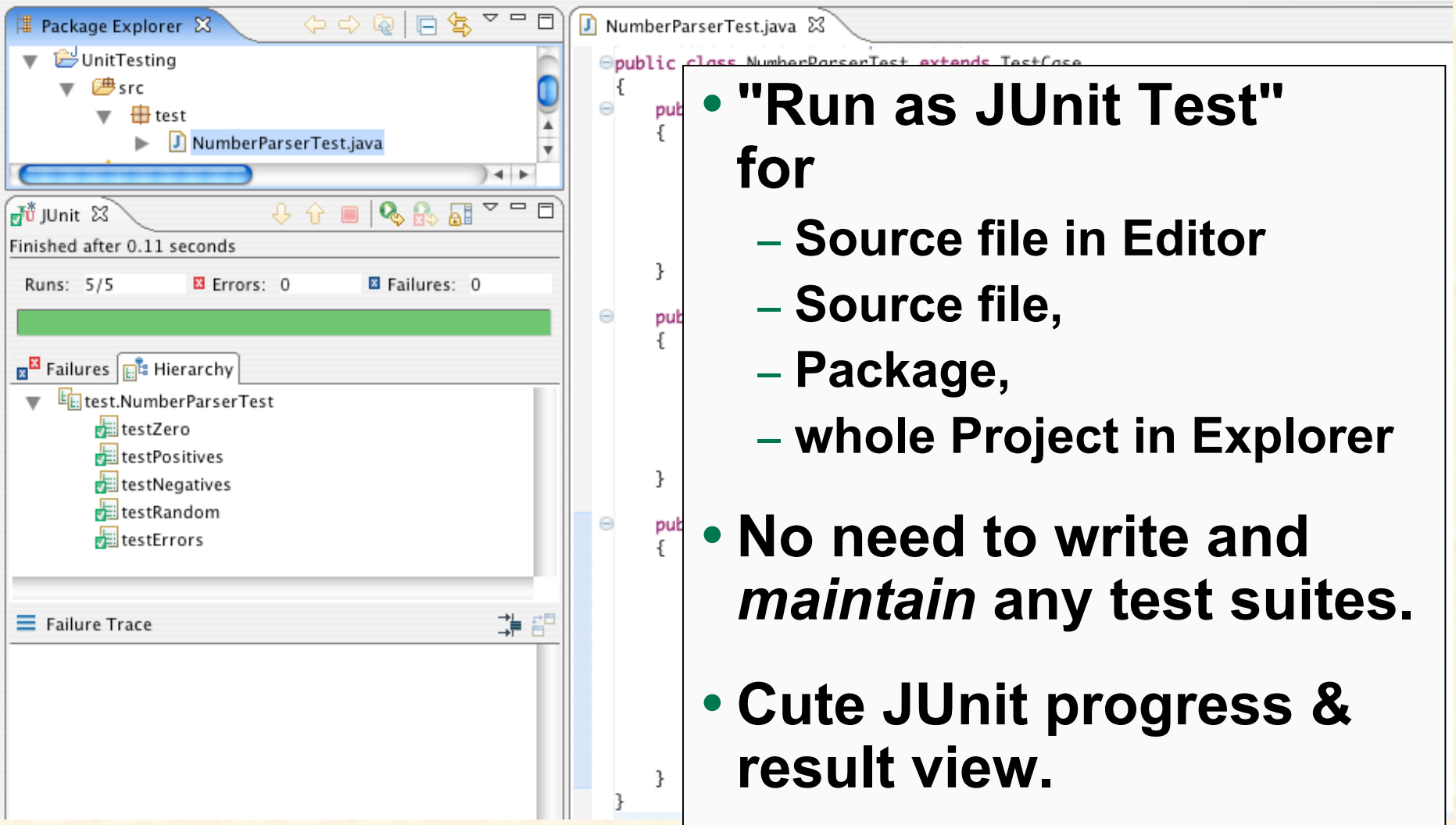
.. Keep the Tests Running?

- What if I forget to add a newly created test case to the suite?
- What if I want to run only one of the test cases?
  - … and then forget to re-enable "the rest" after fixing that one test case?

```
public class Main
{
    public static void main(String[] args)
    {
        TestSuite suite = new TestSuite();
        // Make sure that you add all your test classes here:
        suite.addTestSuite(NumberParserTest.class);
        // ------------------------------------------------
        junit.textui.TestRunner.run(suite);
    }
}
```

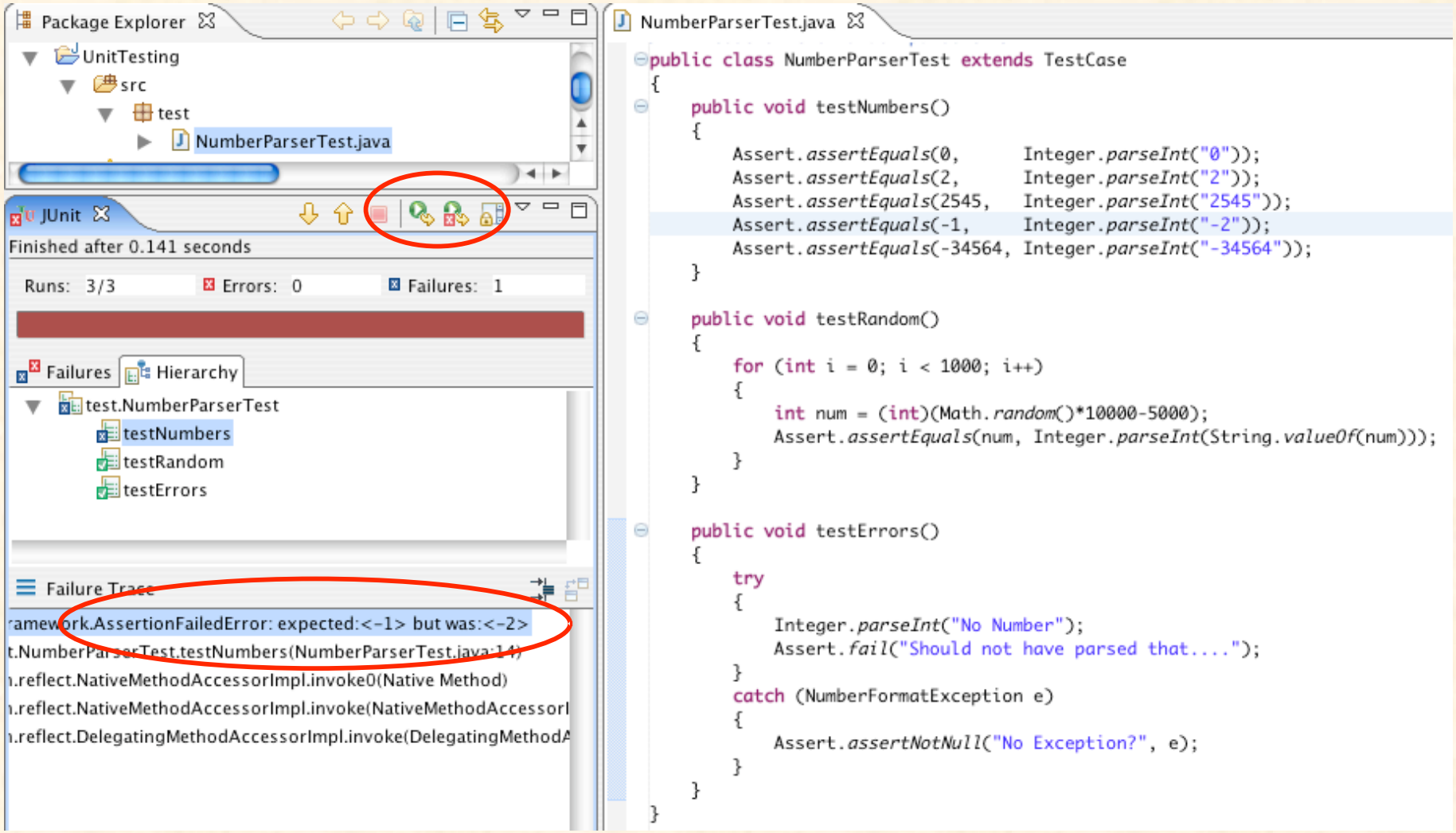


JUnit Integration into Eclipse



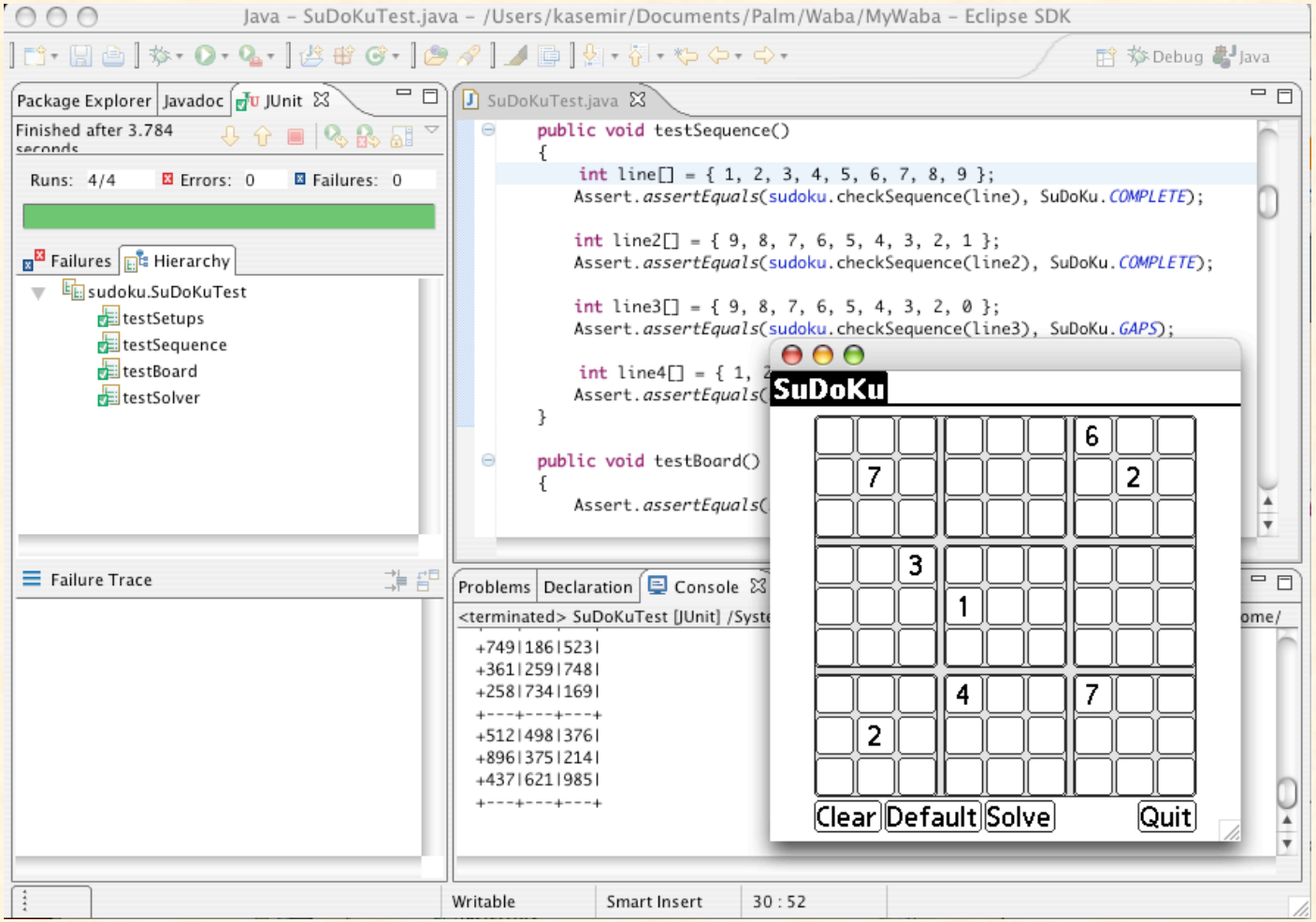
- "Run as JUnit Test" for
  - Source file in Editor
  - Source file,
  - Package,
  - whole Project in Explorer
- No need to write and *maintain* any test suites.
- Cute JUnit progress & result view.

Example of Failed Test



Great Stuff!

- … if you develop for Java



- How can I use that at work (LLRF, Archiver) with C, C++, Matlab, Sequencer, …?

Test support in other Languages?

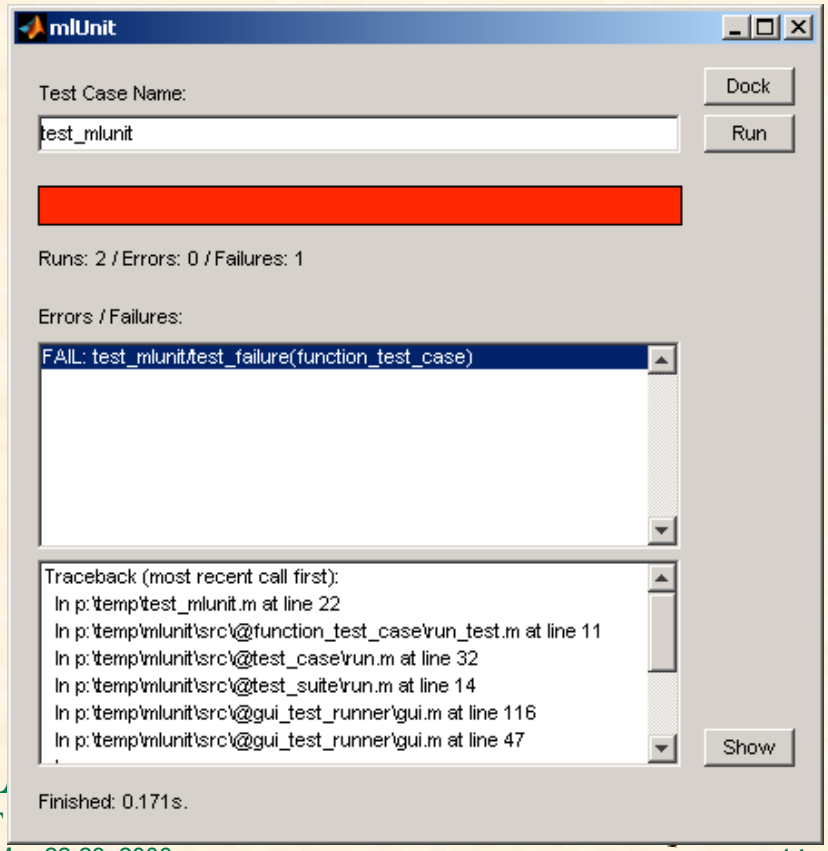
- JUnit & Eclipse benefit from introspection
  - Query JVM for the available tests…
- CUnit, CppUnit:
  Must manually add test cases and their methods.

```
CppUnit::TestSuite suite;
CppUnit::TestResult result;
suite.addTest( new CppUnit::TestCaller<ComplexNumberTest>(
                        "testEquality",
                        &ComplexNumberTest::testEquality ) );
suite.addTest( new CppUnit::TestCaller<ComplexNumberTest>(
                        "testAddition",
                        &ComplexNumberTest::testAddition ) );
suite.run( &result );
```

- How important is *easy invocation of tests* vs. number of specialized "assert" calls, progress bar GUI, html & pdf test reports, …?
  - To me: extremely!

Matlab Test Libraries

- mlUnit, MUnit, MATUnit are all semi-automatic:
  - Given an *.m file name, they locate the tests in there.
    - They don't find the *.m files that contain tests.
  - mlUnit handles test cases implemented as…
    - functions in plain *.m
    - MatLab objects
  - mlUnit has cute GUI runner
- But only MATUnit runs on all the Matlab versions that I encounter at the SNS.



My Favorite: MATUnit

- **Two lines boilerplate turn .m file into "Test"**

```
function unit_test

% Standard boilerplate for any test .m file
tests = str2func(suite([mfilename '.m']));
[passres, failres, warnres] = runner(tests, 1);

function testAdd
assertEquals('Add', 4, 2+2);

function testCompare
assertTrue('Compare', 10 > 5);
```

- **Minimalist reports**

```
>>  unit_test
....
Time: 0.002s
OK (2 tests)
```

```
>>  unit_test
...F
Time: 0.014s
There were 1 failures and 0 errors:
FAIL: testCompare: Compare
    at /ade/epics/iocTop/R3.14.7/linac/llrf/dev/auxLLRFApp/src/unit_test.m (testCompare):11
    at ../../ThirdParty/matunit/runner.m:38
    at /ade/epics/iocTop/R3.14.7/linac/llrf/dev/auxLLRFApp/src/unit_test.m:5
FAILURES!!!
Tests run: 2,  Failures: 1,  Errors 0
```

My C++ Favorite

- Didn't find any.
- Wrote perl script that looks for *Test.cpp with content like this:

```
#include "UnitTest.h"

TEST_CASE foo()
{
    TEST(1 == 2-1);
    TEST(strlen("Hi") == 2);
    TEST_OK;
}
```

- "make test"
  - invokes that perl script, which creates
    - UnitTest.cpp with main() routine and
    - EPICS make system include file,
  - compiles and runs the UnitTest executable:

```
Unit xxxTest:
------------------------------------------------------------------

foo:
  OK  : 1 == 2-1
  OK  : strlen("Hi") == 2
==================================================================
Tested 1 unit, ran 1 test, 1 passed, 0 failed.
Success rate: 100.0%
```

- Only options: Run all tests, run single test.

My Summary on Testing

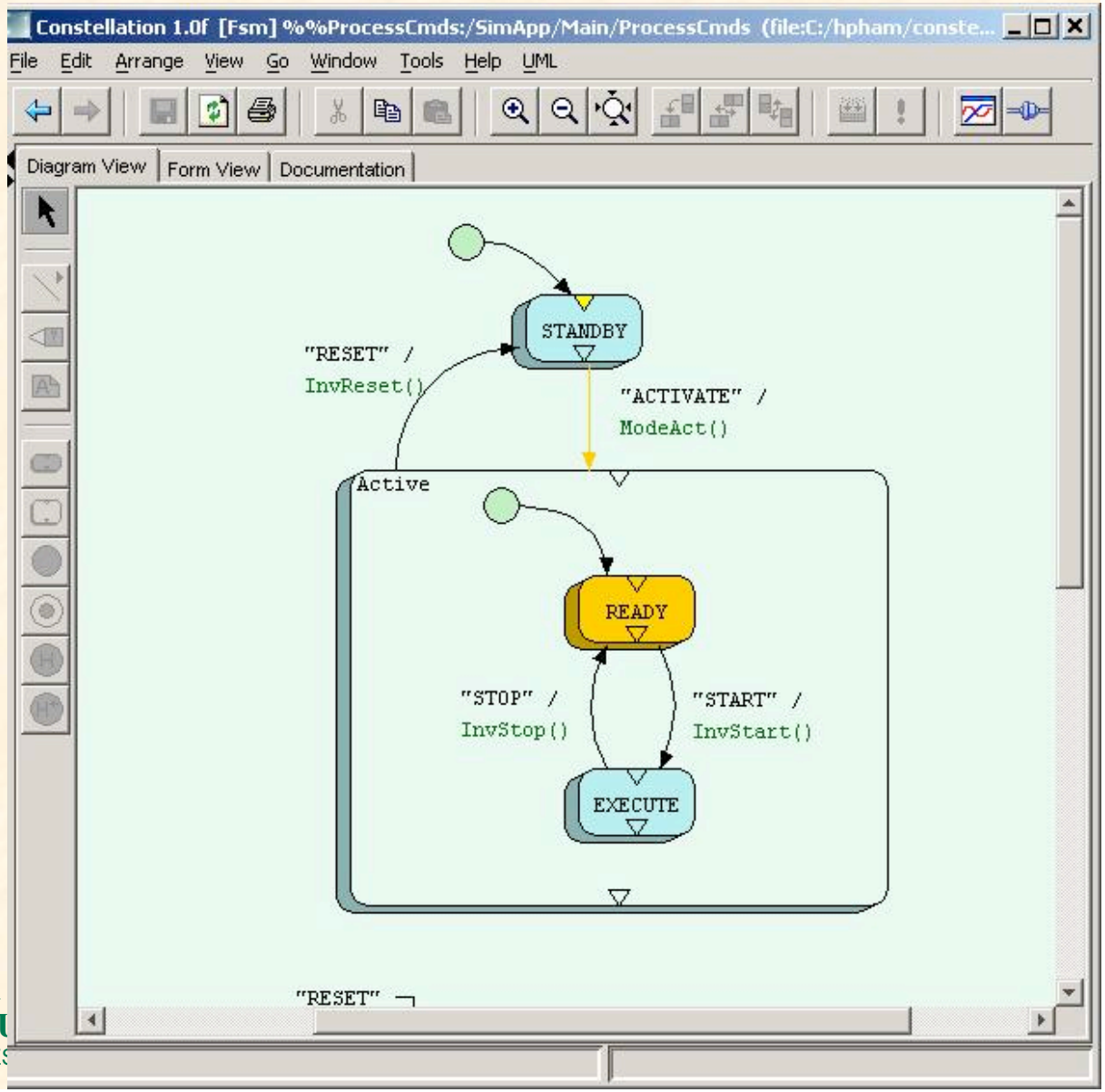
- Test-driven design very useful for new work.
- Found JUnit and Eclipse support outstanding
  - .. but have no use for Java in my day job.
  - Have not found anything for C++ that beats a simple perl script.
  - Andrew Johnson is adding test helpers to EPICS base
    - Idea:
      Executables print a list of "ok" or "not ok" following the Perl module Test::More.
- Experience from refactored ArchiveEngine
  - Split existing code into smaller, unit-testable pieces.
  - 35k changed into 45k lines of code,
    ~7k of which are tests.
  - Found (and fixed, I think) many errors.
  - Finally, several weeks of test runs where all observed errors were in CA client library.

The EPICS Sequencer/State Machine

- Compared to Doocs, LabVIEW, RTI Control Shell/Constellation, … EPICS lacks a graphical editor for the sequences.
- 'SNL' state machine code not ideal for documenting existing functionality or discussing changes

RTI Constellation (ex "Control Shell") FSM editor/viewer

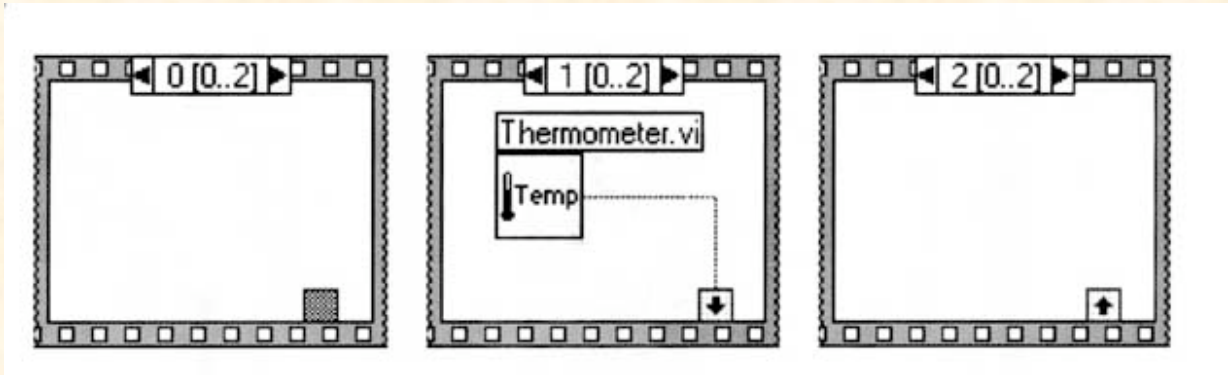
- Show Structure
  - States,
  - Transitions
- 'Click' to see Functionality
  - Code
- Doocs has the same idea
  - but more ugly



This statement represents the personal opinion of the author and is does in no way represent the organization for which he works nor the EPICS community as a whole.

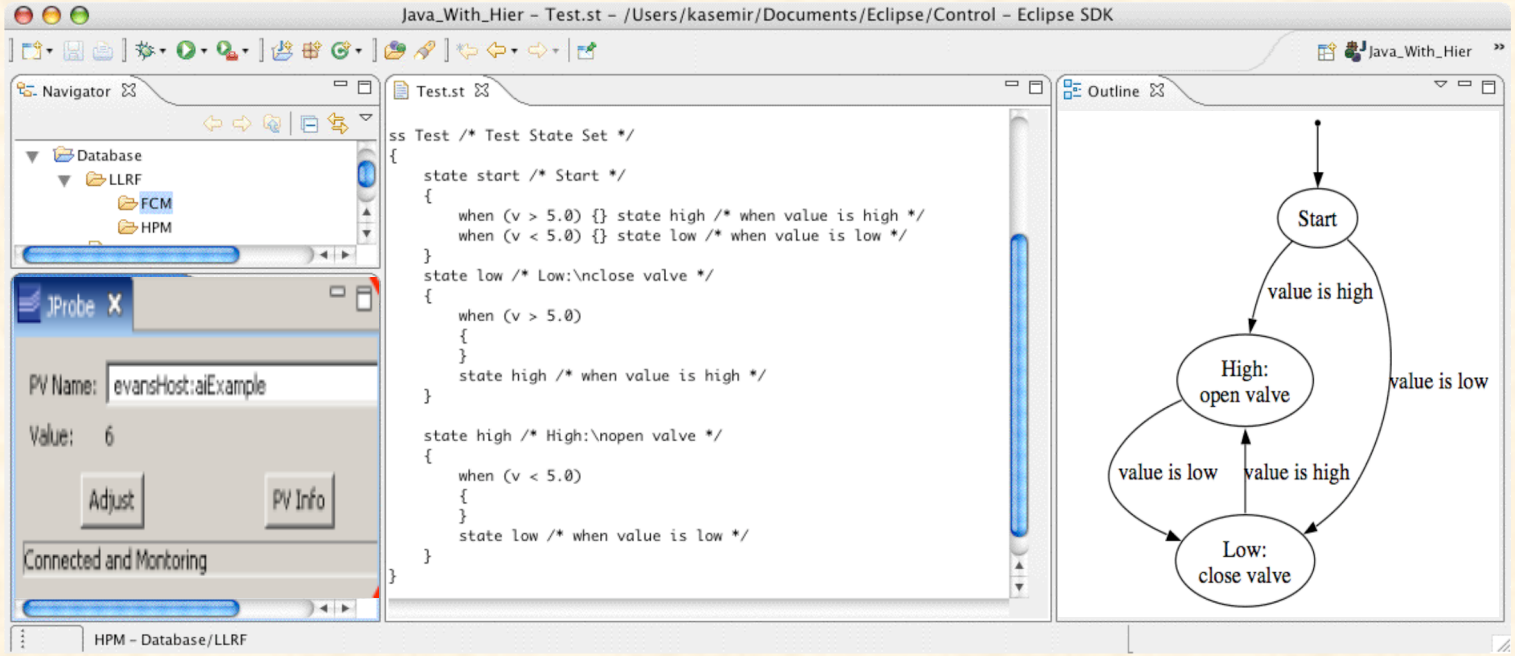
LabVIEW "Sequence"

- Shows Functionality for one state



- "Movie" idea is cute, but Structure is less obvious
  - Image is montage of subsequent frames
  - I can only see one frame at a time, and spend a lot of time clicking back and forth
- LabVIEW "case" statement is similar.
  - .. and maybe more suitable for "State machine"
  - .. but even more confusing when trying to see Structure.

How About This?

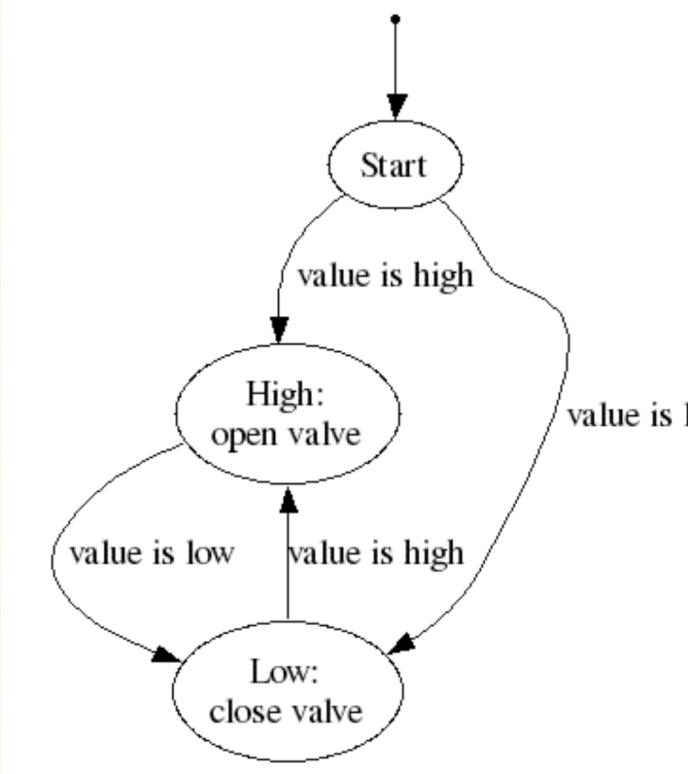


- **SNL support as part of IDE**
  - **Browse EPICS databases**
  - **Run 'probe' on a live record**
  - **Edit the SNL code**
  - **Life update of the sequence diagram**

State Diagram: Bubbles + Arrows?

- Easy to understand overview of states and transitions.
- Implemented perl script that parses SNL code for
  - States
  - Transitions

  … which are marked by special comments, creating a Graphviz 'dot' file



- http://ics-web1.sns.ornl.gov/~kasemir/snl_diags

SNL with 'magic' Comments

Comments provide some control over what gets diagrammed and how.

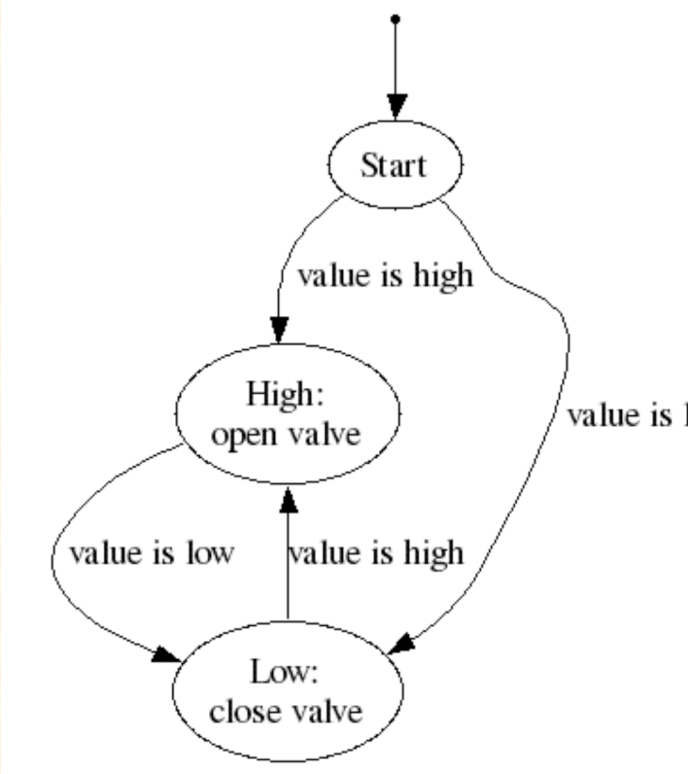
snl_dot.pl -h

gives usage info and basic SNL comment syntax.

```
/* Demo sequence.
 *
 * Run this to create dot file:
 *
 *   perl snl_dot.pl Test.st
 */
program test

double v;

ss test /* Test State Set */
{
    state start /* Start */
    {
        when (v > 5.0) {} state high /* when value is high */
        when (v < 5.0) {} state low /* when value is low */
    }
    state low /* Low:\nclose valve */
    {
        when (v > 5.0)
        {
        }
        state high /* when^ value is high */
    }

    state high /* High:\nopen valve */
    {
        when (v < 5.0)
        {
        }
        state low /* when^ value is low */
    }
}
```

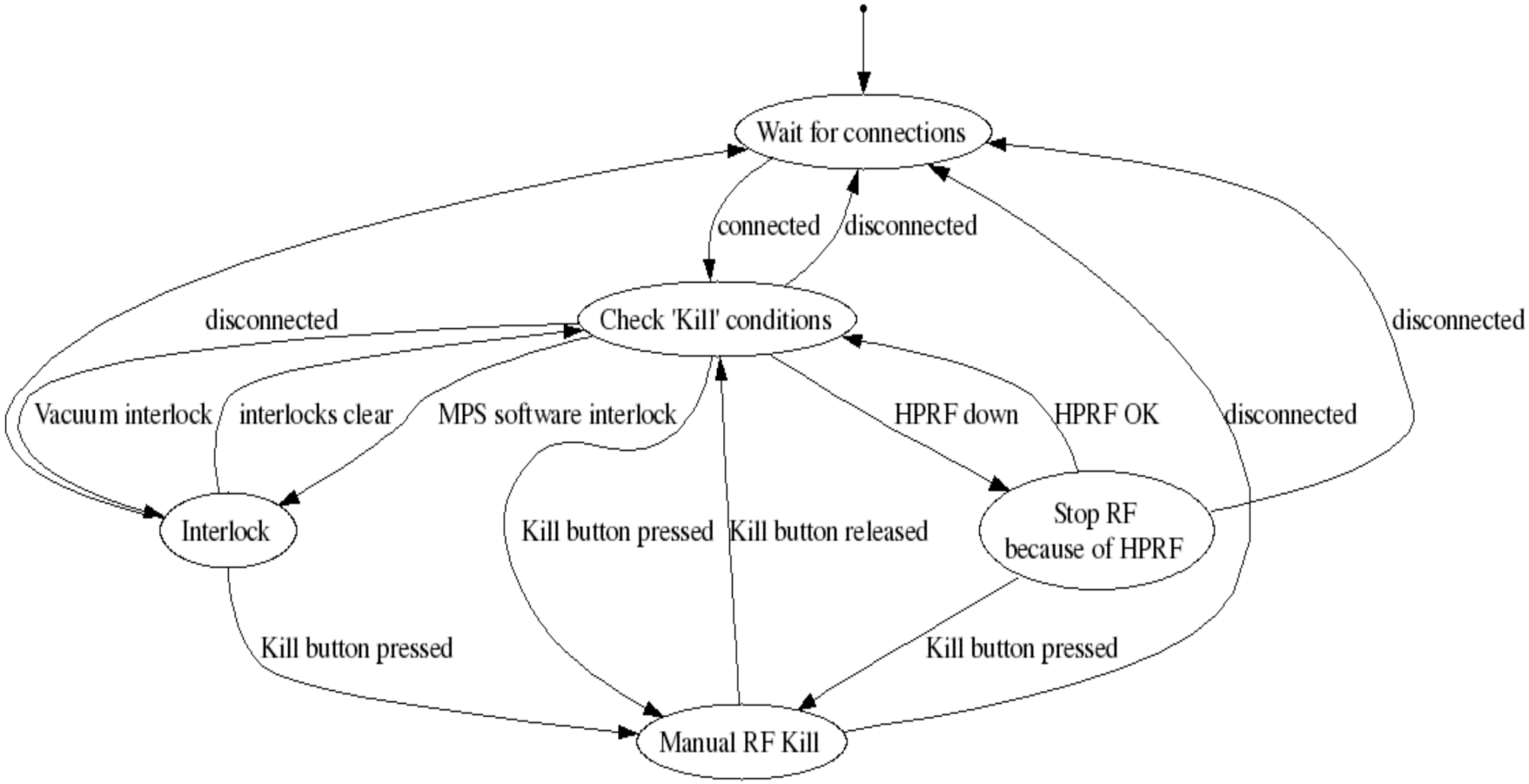
Limitations

- Not shown:
  - PVs that are behind SNL variables
  - Code that's executed on transitions
- No interactive editor.
- Little control over layout beyond descriptive text for states and transitions.
- Comments used for diagram can be descriptive and helpful, but also wrong.
- Bottom Line:
  Minimal effort, easy to use, and the diagram is 'in sync' with the SNL.

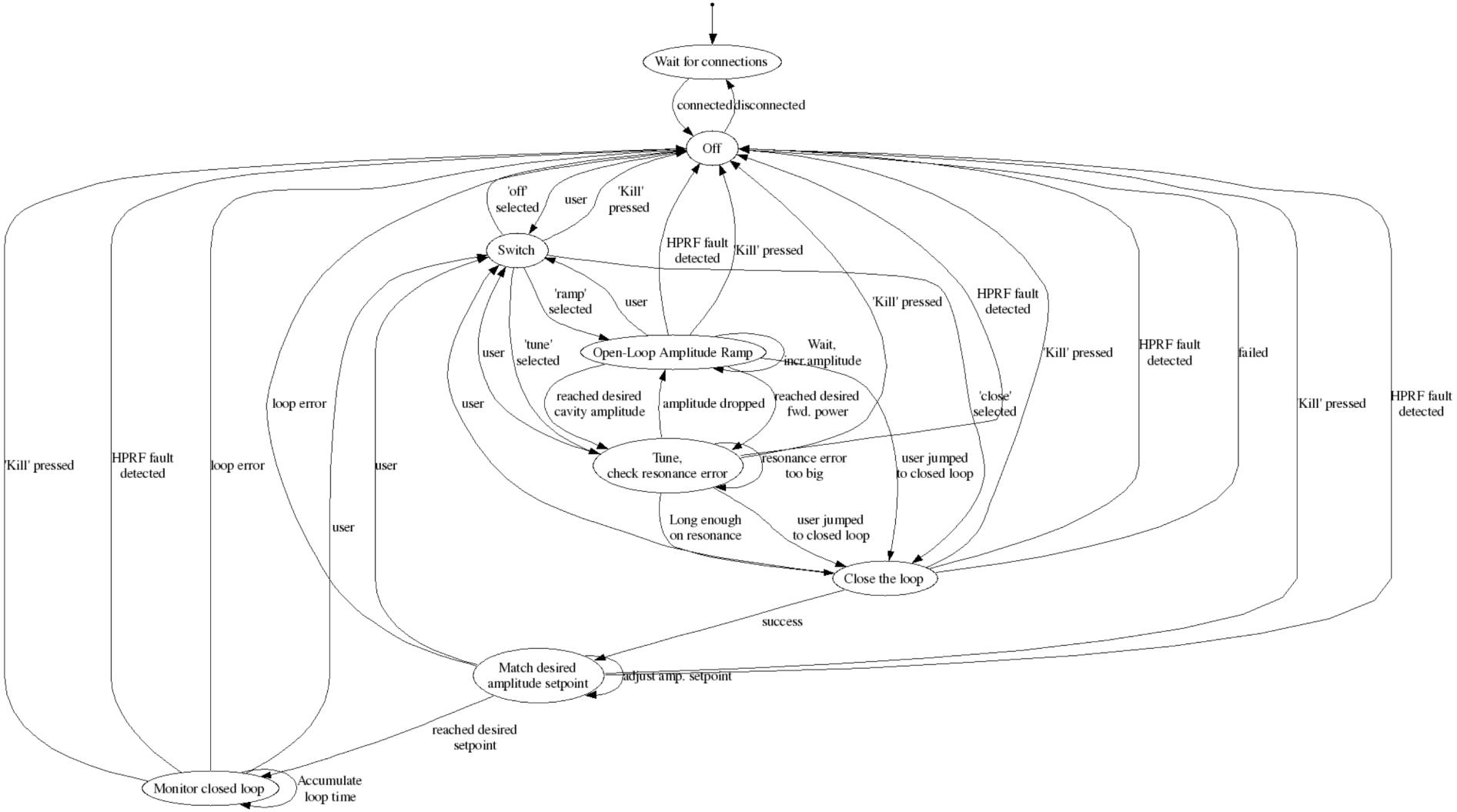


State Diagram Limitations

- **Even "simple" FSM complicated by**
  - **Error conditions ("disconnected")**
  - **Manual interaction.**

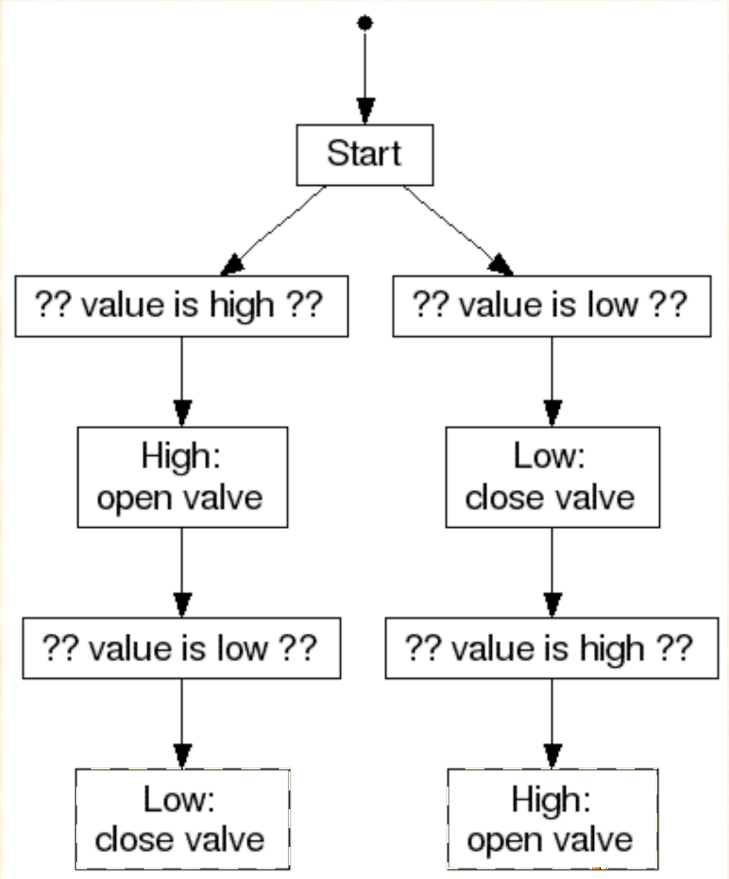


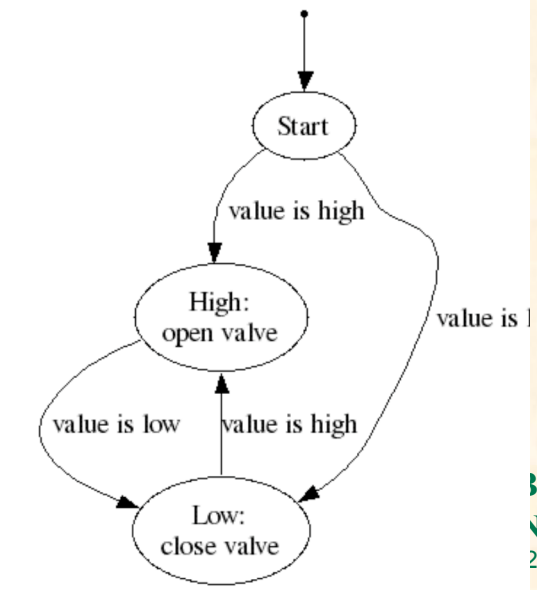
A Real-Word Example



Dromey Behavior Tree

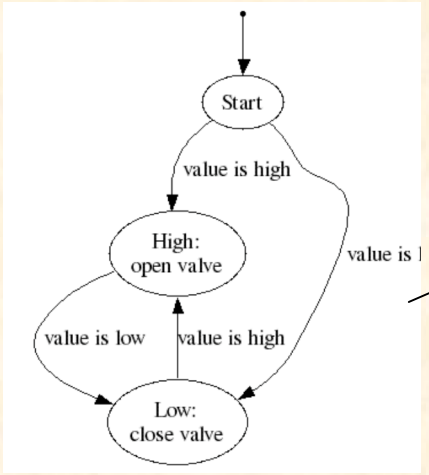
- Circular state transitions, though desired, cause messy diagrams.
- Dromey (http://www.sqi.gu.edu.au/gse) developed a mechanism for handling complex requirements in "Behavior Tree" notation.
- Carl Lionberger (LBL) developed some EPICS sequence code based on DBTs in a manual process.
- Neat idea: Show that a certain state is re-entered without drawing the transition "back" to the original state.

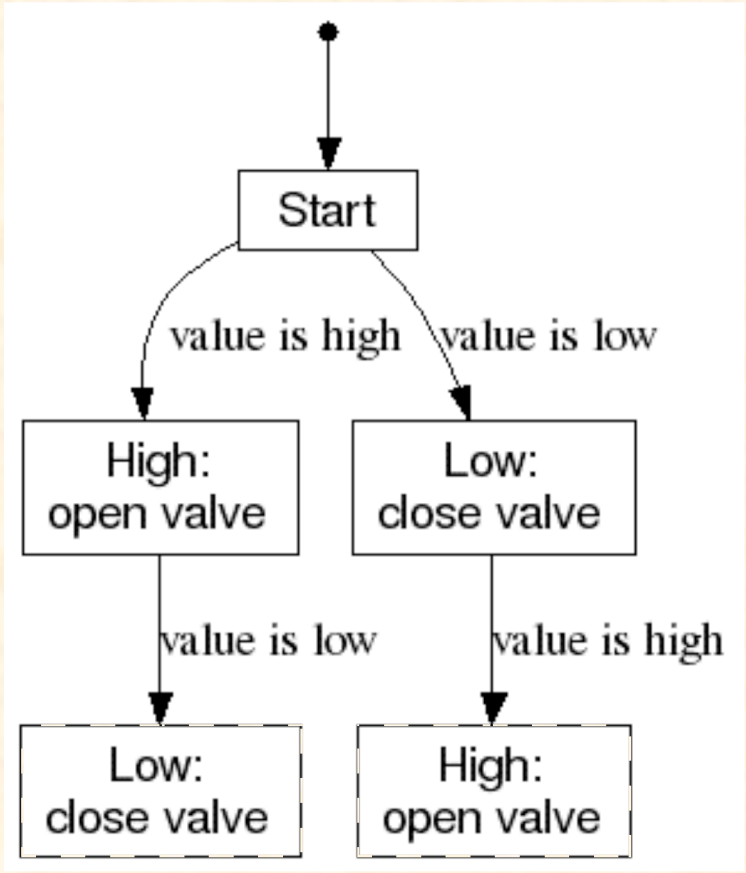




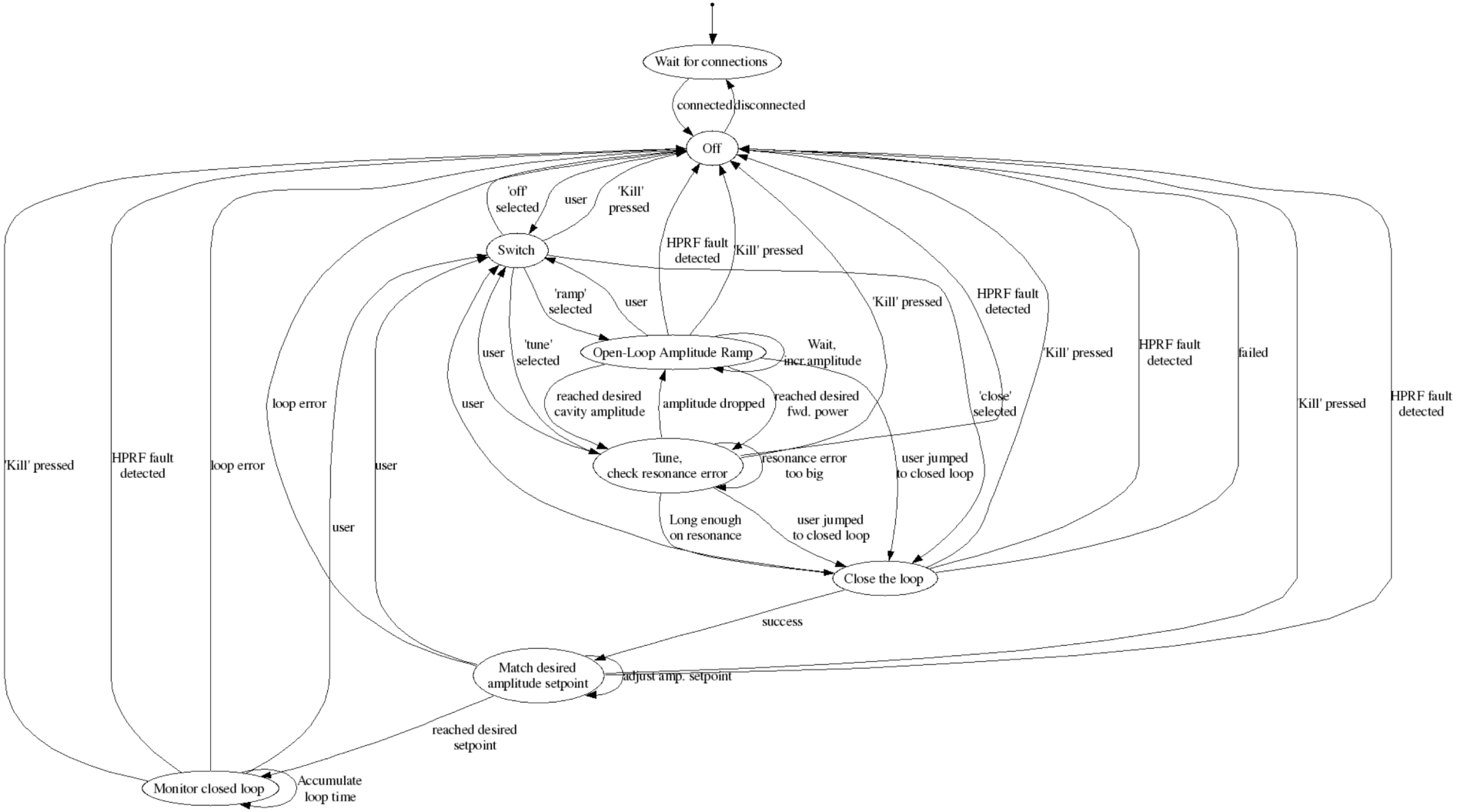
Mix of ideas

- **Basically the old state diagram, but with Dromey's idea for representing cycles.**
- **Perl tool that can create dot files for**
  - **a) Original state diagram**
  - **b) Sort-of-Dromey Behavior Tree**
  - **c) The mix shown here**
  - **from the original sequencer source file,**
  - **aided by special comments in the code.**

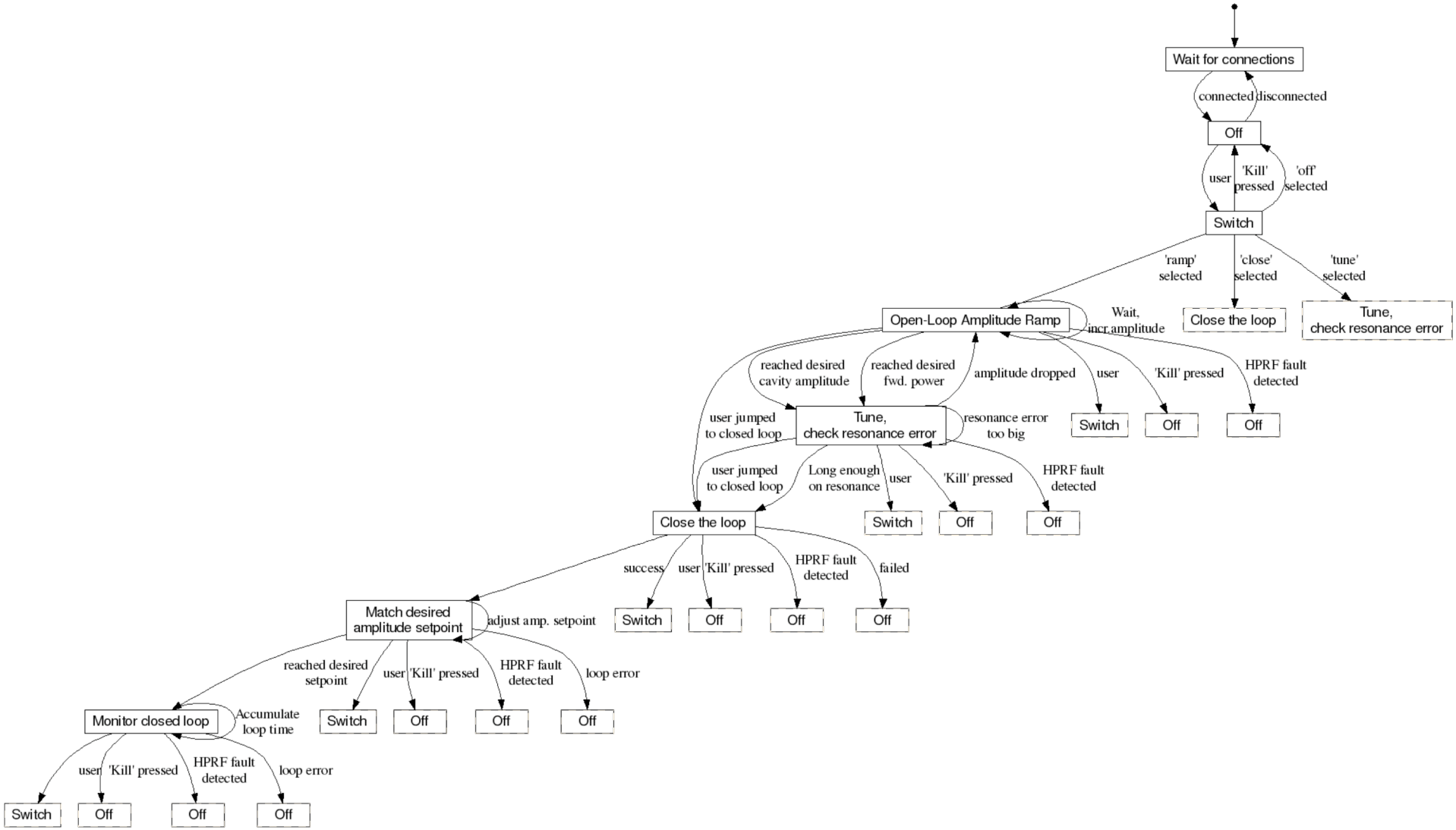




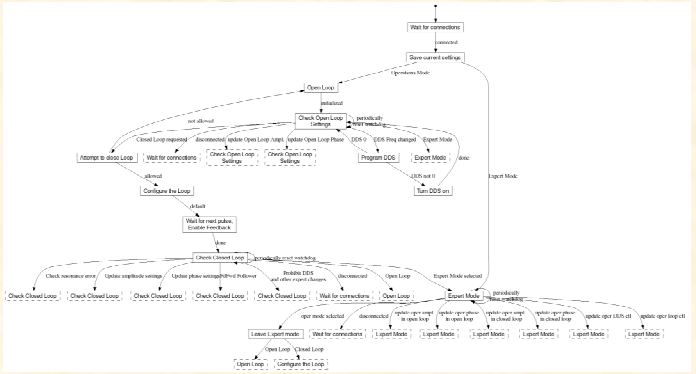
Compare: Traditional state diagram

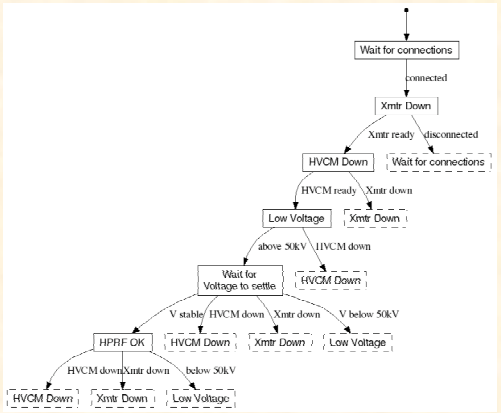


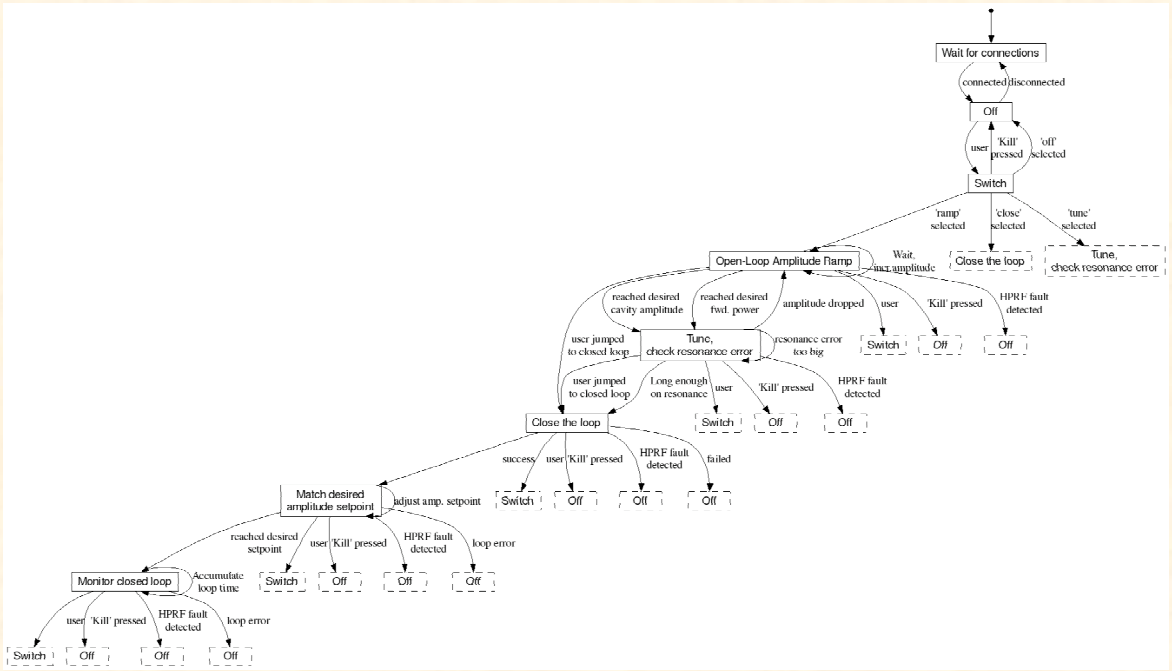
.. and Mixed Representation

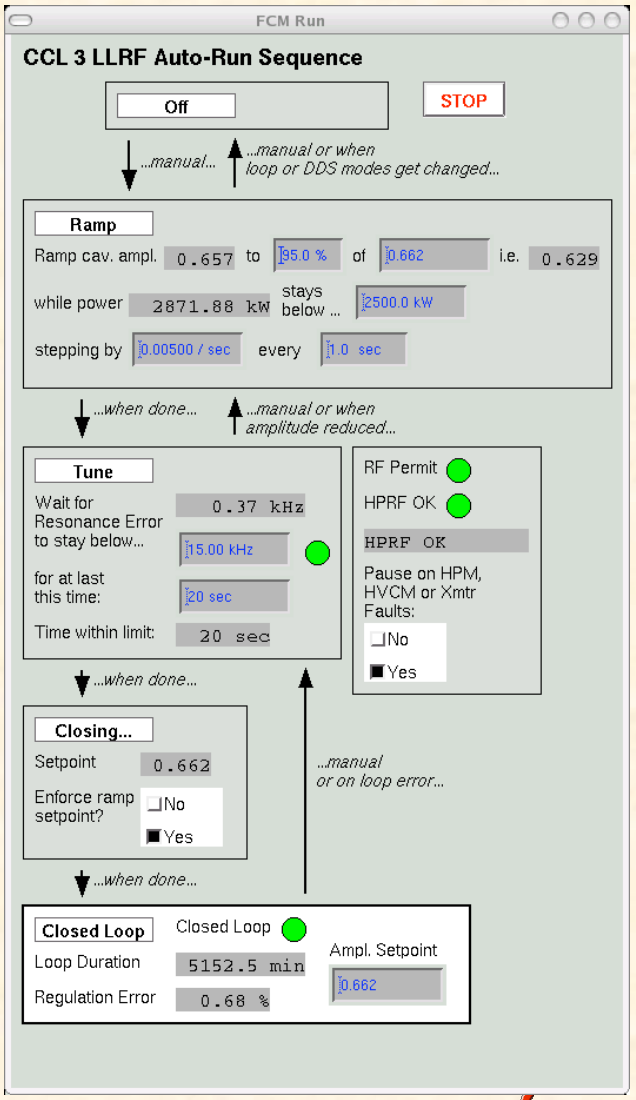


State Diagrams still different from OPI







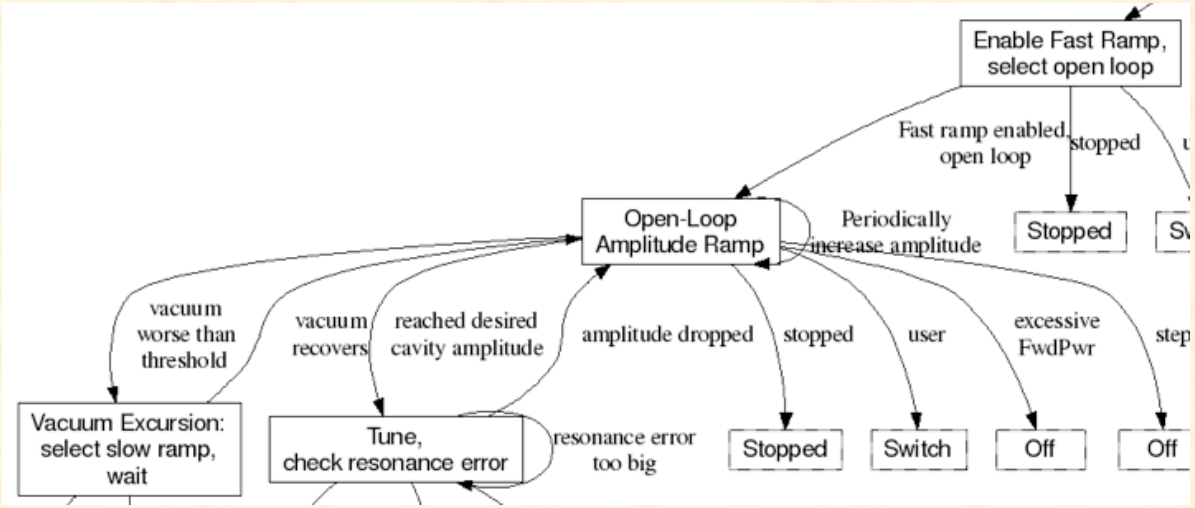


Sequences, Testing

- **Motivation:**
  One piece of the LLRF sequence stopped working
  - **One SS correctly computes an OK/Faulted PV**
  - **Displays fine on OPI**
  - **But second SS which is supposed to react stopped reacting at some point in time (R3.14.7?, seq-2.0.10?)**
- **Debatable:**
  - **Is this a sequencer error?**
    - **Worked when SSs were in separate SNL files, but not when in same.**
  - **Result of fuzzy definition of what wakes state sets and causes them to check values?**
- **The Point:**
  - **Nobody noticed, because it worked once.**
  - **Subsequently added features were tested, but lacking unit tests, a broken feature went unnoticed.**

Usage for LLRF: Implement Tests

- Split SNL code into several files which were easier to 'diagram'.
- Used Matlab (MATUnit) to create tests, using diagrams as guideline what to test.
- Currently ~1000 lines test code, ~10 minute runtime.

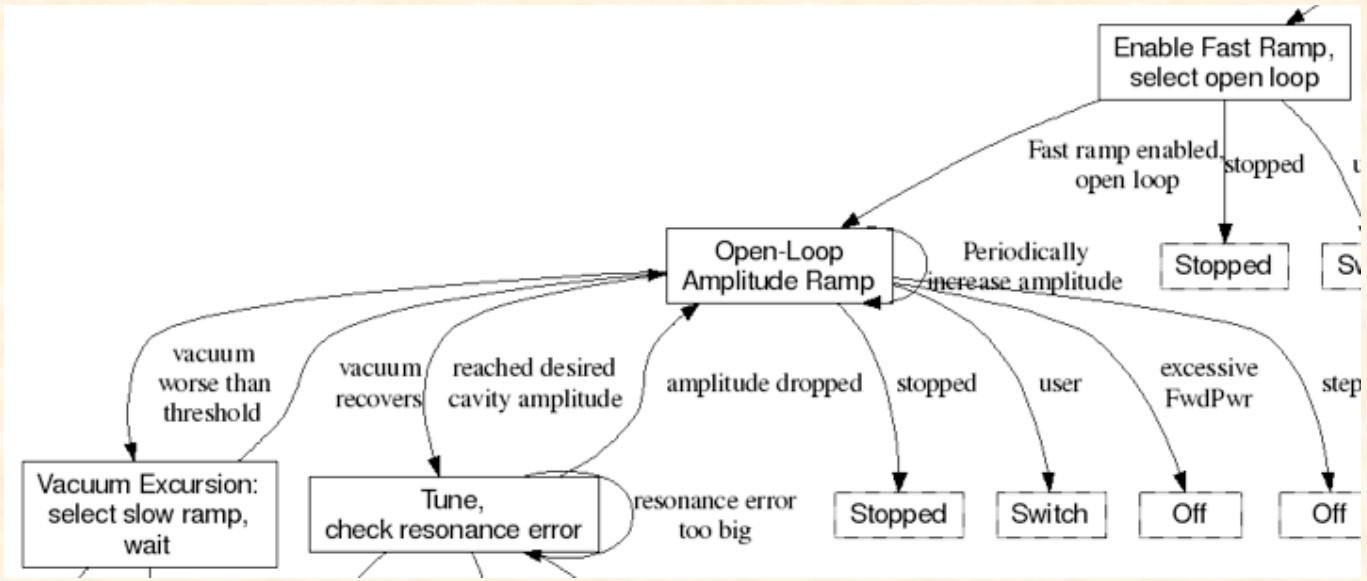



```
function testRampWithVacExcursion
global S N vaclim
caput([ S '_LLRF:FCM' N ':RunState' ], 1);
assertReachValue('Fast Mode', [ S '_LLRF:FCM' N ':FastRamp' ], 1, 5.0);
caput([ S '_LLRF:FCM' N ':Vacuum' ], 2.5*vaclim);
% Expect vac-wait state, fast-mode disabled
assertReachValue('Vac. Excursion state', [ S '_LLRF:FCM' N ':RunState' ], 3, 5.0);
amp = caget([ S '_LLRF:FCM' N ':CtlAmpSet' ]);
assertTrue('Non-zero amplitiude', amp > 0.0);
fast = caget([ S '_LLRF:FCM' N ':FastRamp' ]);
assertFalse('Fast mode', fast);
```

Usage for LLRF: Discussion & Design

- **Turn requests into SNL skeleton,**
- **Review, tweak,**
- **Then add the "body" code.**

1. Verify that RF is ready to operate. (This is the same as it is now)
2. Read the current vacuum level.
3. Ramp LLRF drive up at a user settable step size quickly. (Quickly means several steps a second, maybe 3-5 steps?)
4. After each RF increment, measure the Vacuum and see that it remains below 9.9e-10.
5. If the vacuum level degrades past 9.9e-10, the ramp should be halted until the vacuum recovers below this point.
6. After the vacuum has exceeded the 9.9e-10 limit and the vacuum has been



Messages to take home

- **With Junit/Eclipse type support, unit tests are no pain-in-the neck afterthought. Instead:**
  - **easy to add, easy to run**
  - **facilitating test-driven development**
  - **Good Stuff!**
- **Nothing equivalent for other languages, but one can get by with Matlab & perl**
- **It's not too hard to create state diagrams that are 'in sync' with the SNL code.**