RIM's First Look at TTCN3

Sean Cavanagh

June 5th, 2009

TTCN-3 User Conference 2009

3 – 5 June 2009 – ETSI, Sophia Antipolis, France

Last Updated: August 5, 2009

Agenda

- Who Am I and How Did I Get Here?
- What I Do For a Living Now?
  - Interesting things about RIM and the BlackBerry
- What We're Hoping For From TTCN3
- How Things Are Going So Far
- What We Really Like
- What We Would Like To See Improved
  - Inflammatory Statements

Who Am I And How Did I Get Here?

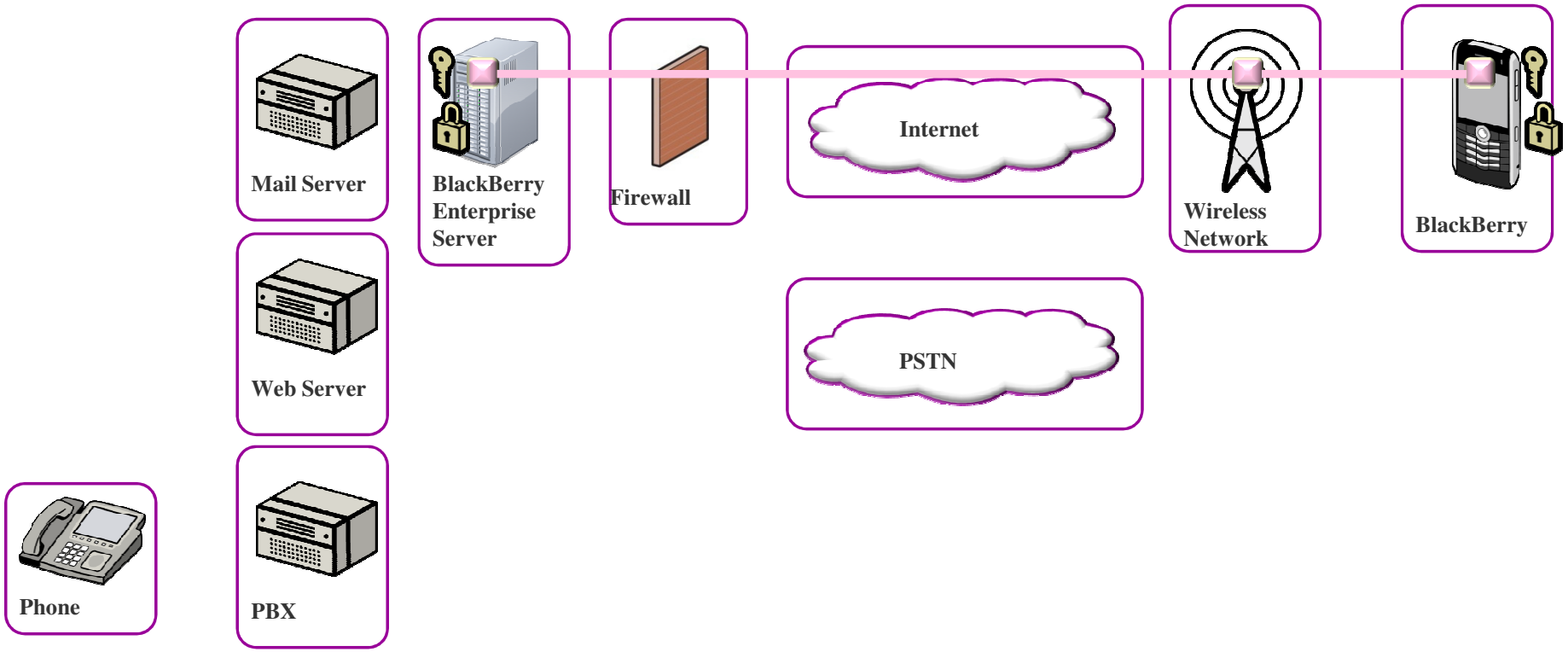
Tests

- Optical Testing
- Telephony Testing
- SIP Endpoint Testing

Languages

- Labview
- Python
- VB
- TCL
- TTCN3

What I Do For a Living Now?



Alphabet Soup:

- SIP to PBX
  - Plus all the stuff underneath SIP
  - SDP, RTP, etc.
- Proprietary Stack to BB
  - N layers deep
  - Encrypted
- SQL
- RMI
- GSM
- CDMA

WHAT WE'RE HOPING FOR FROM TTCN3

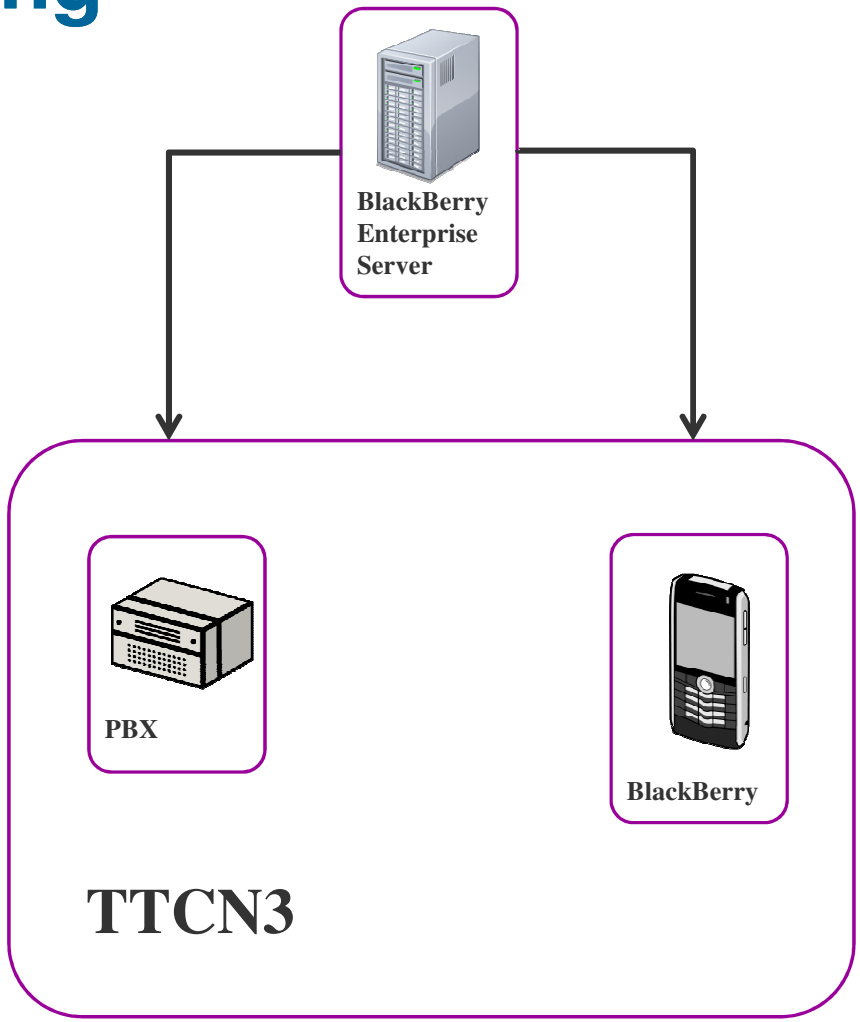
The Grand Dream

- Continuous Integration
  - You didn't screw up, as far as we can tell.
- Modeling of Multiple Environments
- Developer API in TTCN3
- Test Driven Development

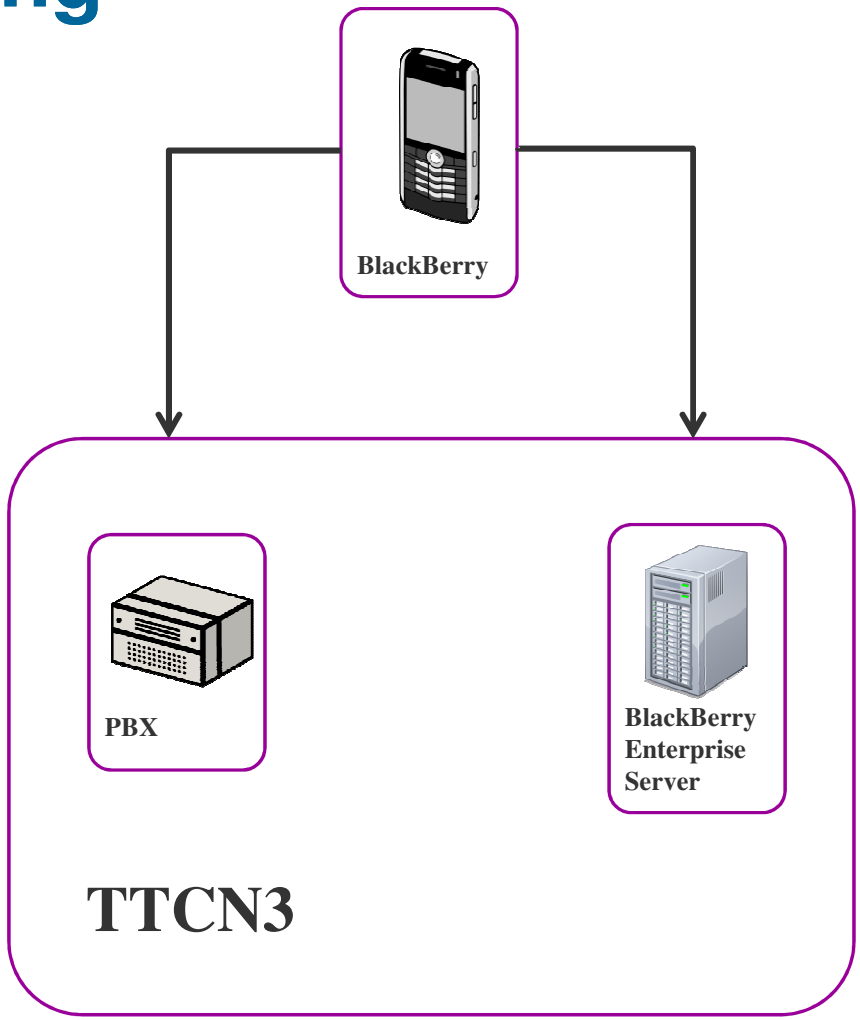
Modeling of Multiple Environments

- Interop is a multi-dimensional problem (probably 4 dimensions)
  - Vendor
  - Model
  - Version
  - Configuration
- My experience is that moving from one point to another in the cube can be ½ a day.

Clamp Testing

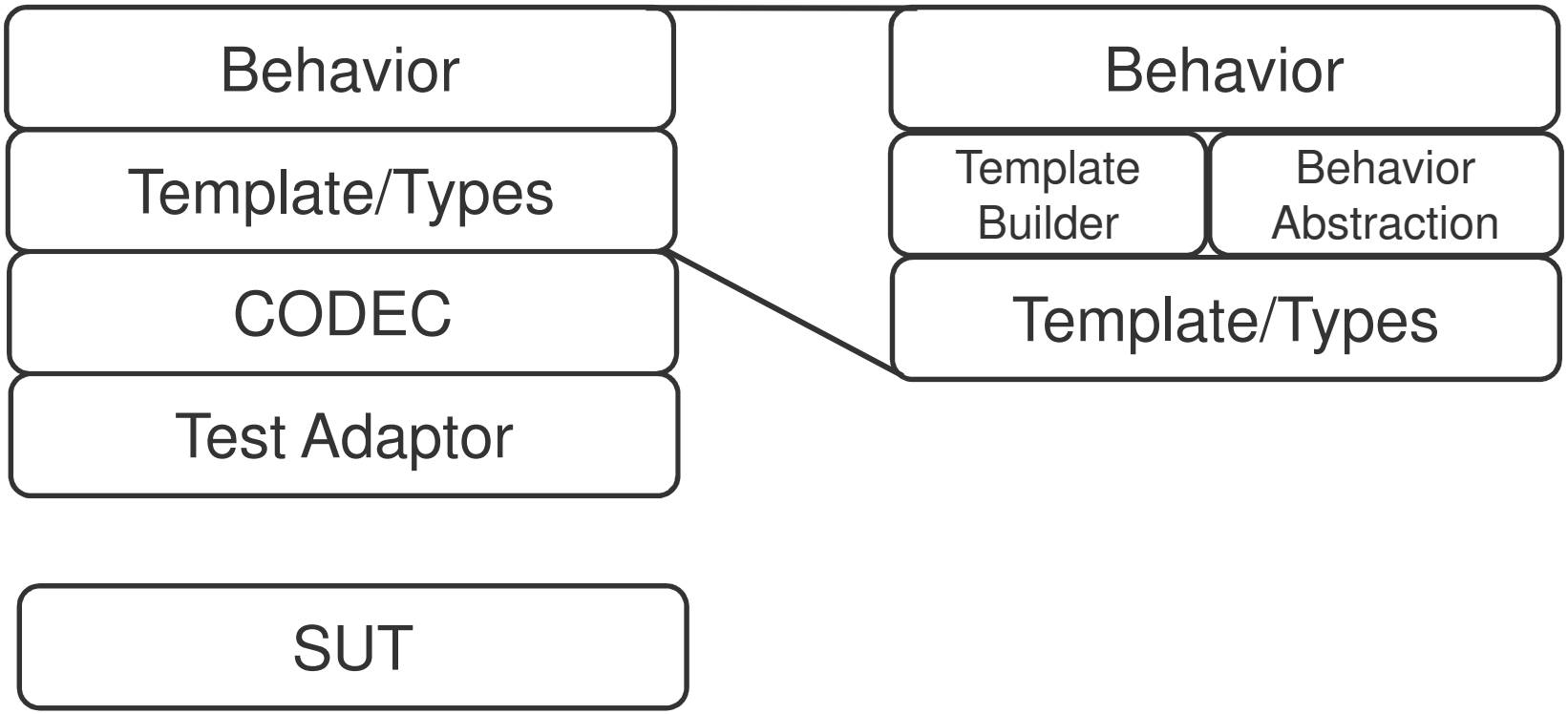


Clamp Testing



Developer API (or toolkit)

- **“Why don’t we just do it in Java?”**



Test Driven Development

1. Write a test
2. Watch it fail
3. Write some code
4. Watch the test pass
5. Profit!

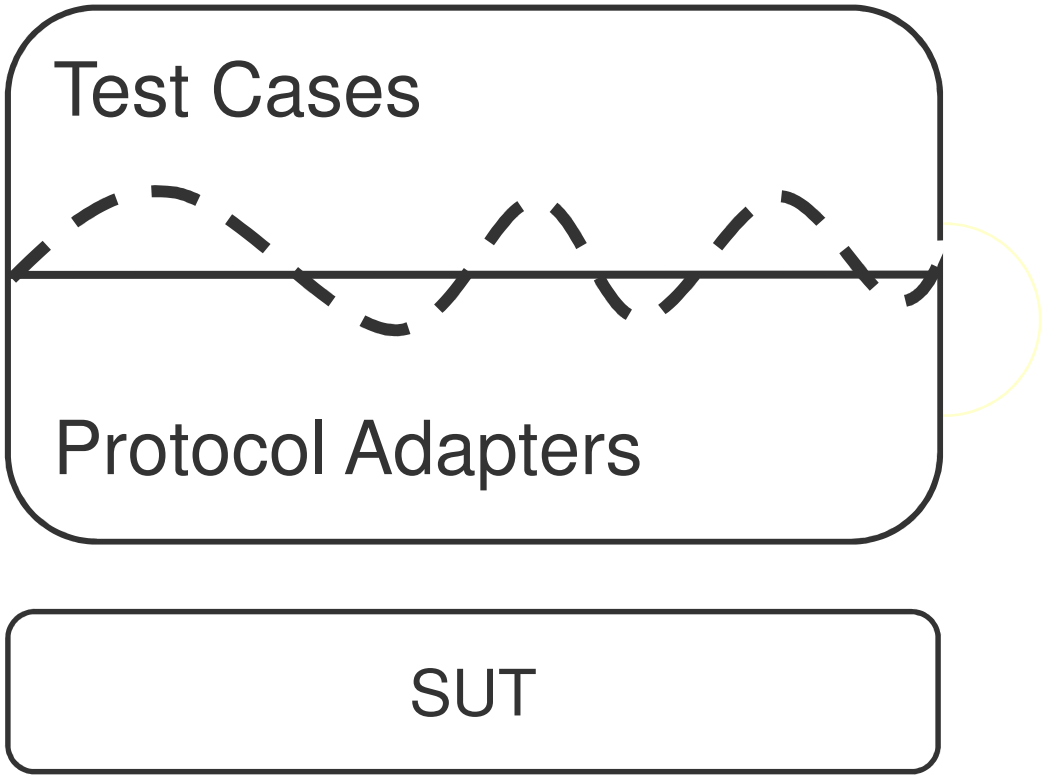
- Tests define behavior
- Behavior defines interfaces
- Developers can become fearless

WHAT WE REALLY LIKE

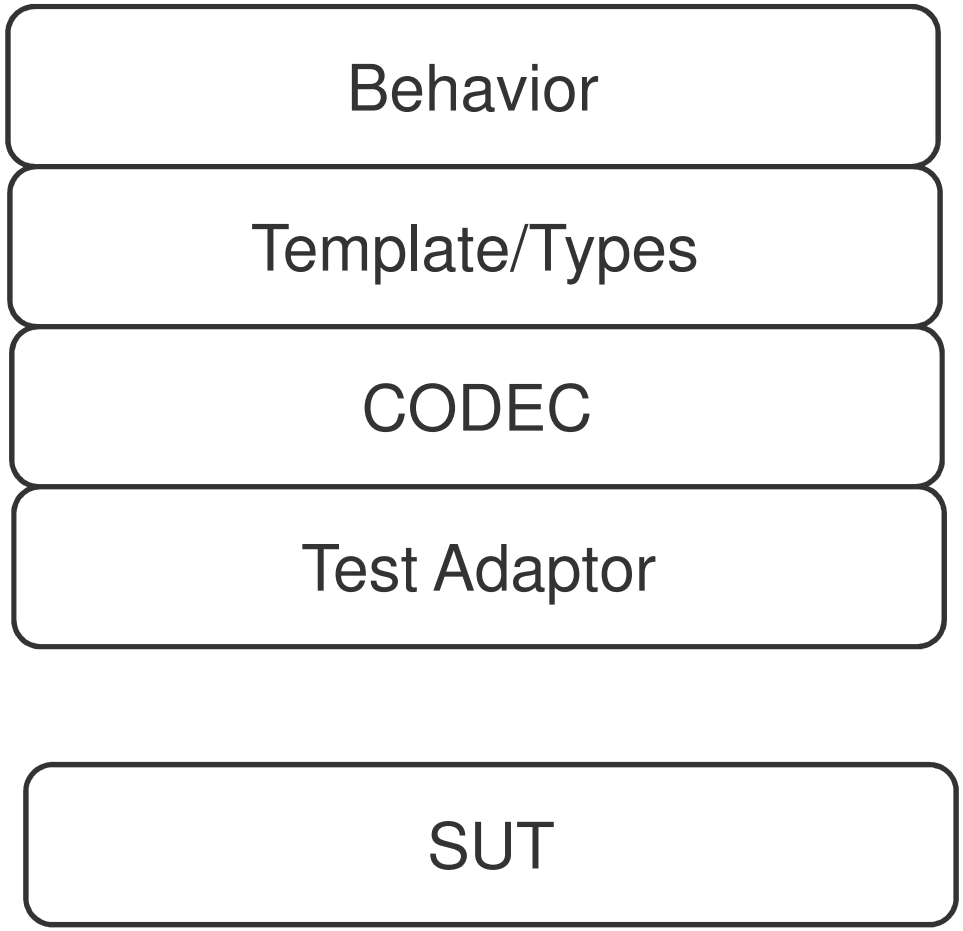
TTCN3 Saves you from Failure

- Architecture
- Type system
- Templates

Common Architecture



TTCN3 Architecture



Type System

- Well Done
- Flexible
- Reusable
- Simple
  - A single type for a single thing

Templates

- Very Strong Idea
- Insist on Full type declaration and parsing
- All incoming data gets semantic meaning
- Poor test oracles another common failure mode
  - Nothing but REGEXES
  - Nothing but Hard Coded
  - Nothing but Wild Cards

WHAT WE WOULD LIKE TO SEE IMPROVED

Getting Started

- It Costs Money
  - Which really means time
- Online Language Reference
  - Perl
  - TCL
  - Python
  - MSDN
  - TTCN-3
  - Labview

Moving along

- Downloadable Content
- CPAN?
- Sourceforge
  - Has the ttcn3gen compiler
  - No modules

Language Features I Think are Missing

Some data structures:

- List methods:
  - Find
  - Replace
  - Length
  - Push, pop, append
  - Sort
- Maps:
  - Len
  - Has_key
  - Items

First Class Functions

- 22 lines. Lots of duplication

```
function receiveInvite() runs on
SipComponentType {
    alt { [] _receiveInvite() {} }
}
altstep _receiveInvite() runs on
SipComponentType {
    var Request v_Request;

    [] SIPP.receive(sipT_1) -> value
v_Request {
     log("LOG 1");
     setInviteHeaders (v_Request);   }
}
```

```
function receiveInviteWithSDP() runs on
SipComponentType {
    alt { [] _receiveInvite() {} }
}
altstep _receiveInviteWithSDP() runs on
SipComponentType {
    var Request v_Request;

    [] SIPP.receive(sipT_2) -> value
v_Request {
     log("LOG 2");
     setInviteHeaders (v_Request);   }
}
```

First Class Functions

- **Refactored! 18 Lines!**

```
function receiveInviteWithSDP() runs on SipComponentType {
    alt { [] _receiveInvite(sipT_2,"LOG 2") {} }
}

function receiveInvite() runs on SipComponentType {
    alt { [] _receiveInvite(sipT_1,"LOG 1") {} }
}


altstep _receiveInvite(template sipTemplate, charstring logMsg) runs on SipComponentType {
    var Request v_Request;

    [] SIPP.receive(sipTemplate) -> value v_Request {
      log(logMsg);
      setInviteHeaders(v_Request);
    }
}
```

First Class Functions

- But wait! How do I refactor this?

```
function receiveInvite() runs on SipComponentType
{
    alt { [] _receiveInvite() {} }
}
altstep _receiveInvite() runs on SipComponentType
{
    var Request v_T_1;
    [] SIPP.receive( sipT_1 ) -> value v_T_1{
      log("LOG 1");
      setInviteHeaders( v_T_1 );    }
}
```

```
function receiveNotify() runs on SipComponentType
{
    alt { [] _receiveNotify() {} }
}
altstep _receiveNotify() runs on SipComponentType
{
    var Request v_T_1;
    [] SIPP.receive( sipT_3 ) -> value v_T_1 {
      log("LOG 2");
      setNotifyHeaders( v_T_1 );     }
}
```

First Class Functions

- 21 Lines! 12 lines savings!

```
function receiveInviteWithSDP() runs on SipComponentType {
    alt { [] _receiveInvite(sipT_1,"LOG 1",setInviteHeaders) {} }
}

function receiveInvite() runs on SipComponentType {
    alt { [] _receiveInvite(sipT_2,"LOG 2",setInviteHeaders) {} }
}

function receiveNotify() runs on SipComponentType {
    alt { [] _receiveInvite(sipT_3,"LOG 3",setNotifyHeaders) {} }
}

altstep _receiveInvite(template Request, charstring logMsg, function setHeaders)
    runs on SipComponentType {
    var Request v_T_1;

    [] SIPP.receive(Request) -> value v_Request sender sent_label {
     log(logMsg);
     setHeaders( v_T_1);
    }
}
```

First Class Functions

- OK, closures are probably pushing it…

```
function generateReceive(template sipTemplate, charstring logMsg, function setHeaders) runs on
SipComponentType{
    var output := function () runs on SipComponentType {
         alt { [] _receiveInvite(sipTemplate,logMsg,setHeaders) {} }
    }
    return output;
}

altstep _receiveInvite(template sipTemplate, charstring logMsg, function setHeaders)
    runs on SipComponentType {
    var Request v_T_1;

    [] SIPP.receive(sipTemplate) -> value recvdTemplate sender sent_label {
      log(logMsg);
      setHeaders(v_T_1);
    }
}

receiveInviteWithSDP := generateReceive(sipT_1, "INVITE+SDP not received", setHeadersOnReceiptOfInvit
receiveInvite := generateReceive(sipT_2, "INVITE not received", setHeadersOnReceiptOfInvite);
receiveNotify := generateReceive(sipT_3,"NOTIFY not received", setHeadersOnReceiptOfNotify);
```

CONCLUSIONS

Early days yet

- 6 months of development and we have:
- Completed our codecs
- Deeply characterized our SUT
- Designed our API

What if we had “Just Used Java”

- Not much Different
  - Completed codecs
  - Characterized SUT
  - Developed API
    - Probably deployed
- But missing
  - Strong IDE
  - Interactive MSC’s
  - Fully developed template system

Questions?