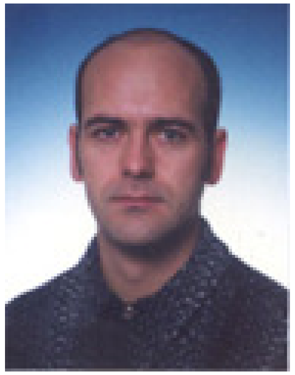
Acceptance Testing at MeVis of an C++ Application

Felix Petriconi

About Me



Study of electrical engineering in Bochum

Since 1994 working as a programmer

- at the university (Turbo Pascal, C++)
- education of high gifted children (PovRay, C++)
- 7 years as freelancer for Ericsson / Siemens-VDO, et al. (C/C++, Perl)
- Since 2003 employed by MeVis Medical Solutions AG (C++, x86, Ruby)

About the product

Reviewing workstation for mammography images

Manufactured for a single OEM customer

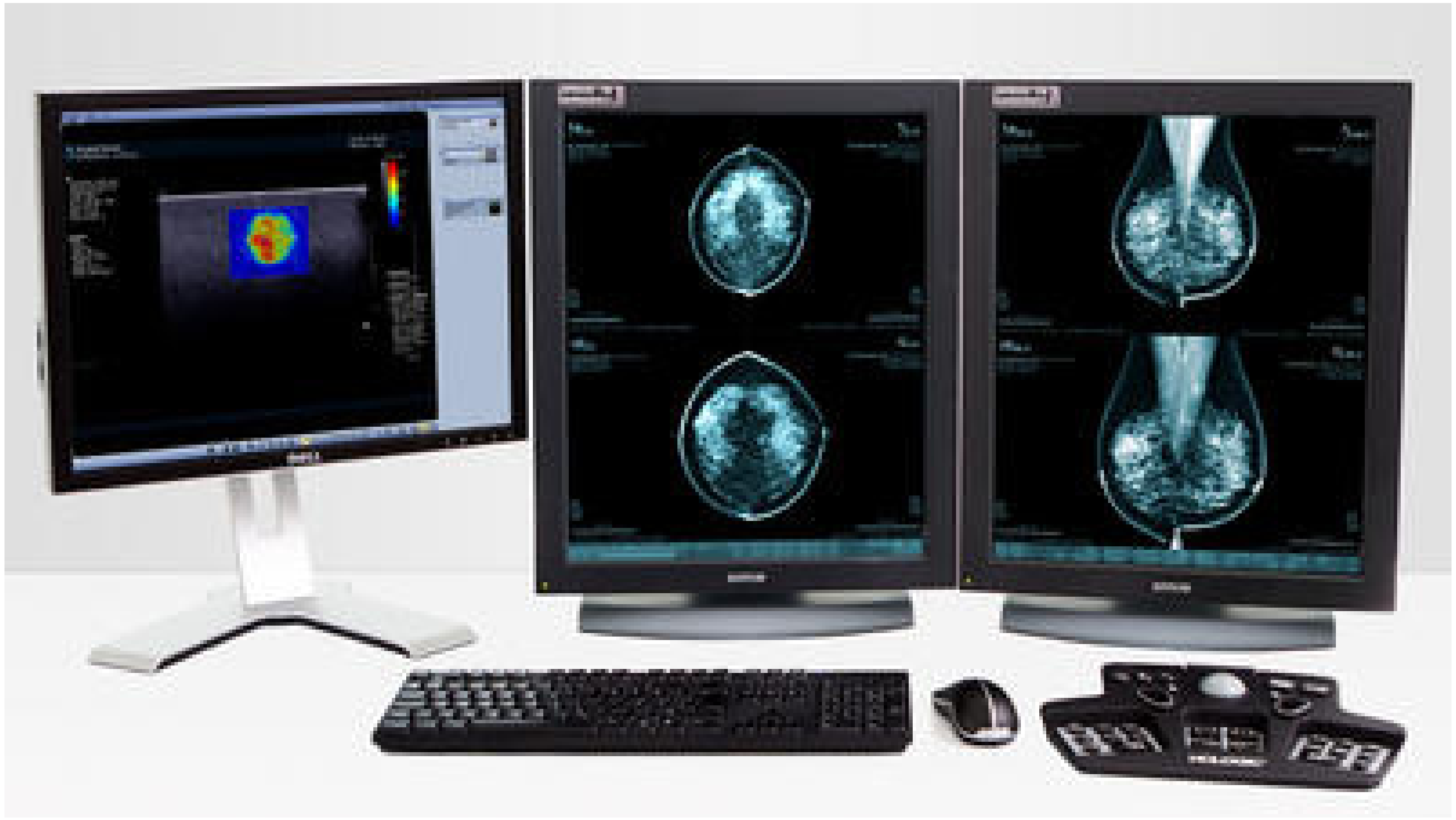
Medical product => Regulated Development Process

In the market since 2002

About 5000 installations world wide

About 50% market share in that segment

Our product



About the application

Deployed as standalone / client-server

OS: Windows 7 / Server 2008 R2

C++ / Qt application

2,5 million lines of code

About the technical challenges

Up to 2.5 GB uncompressed pixel data for a single patient

Up to 400 patients per day

8-16 bit grayscale images on 2 * 5MP 10 bit grayscale displays

Of the shelve workstations

No special HW possible

Each case-change, image change < 1s

Huge variety of hospital setups

About the development problems from the past

At 2011 about 10,000 requirements in a requirement management tool

All requirements had to be traced to a test case

Only paper scripts existed to test the application

Each release test phase took up to 8-12 weeks

Our way out

Let machines perform dumb work

Use people for intelligent work

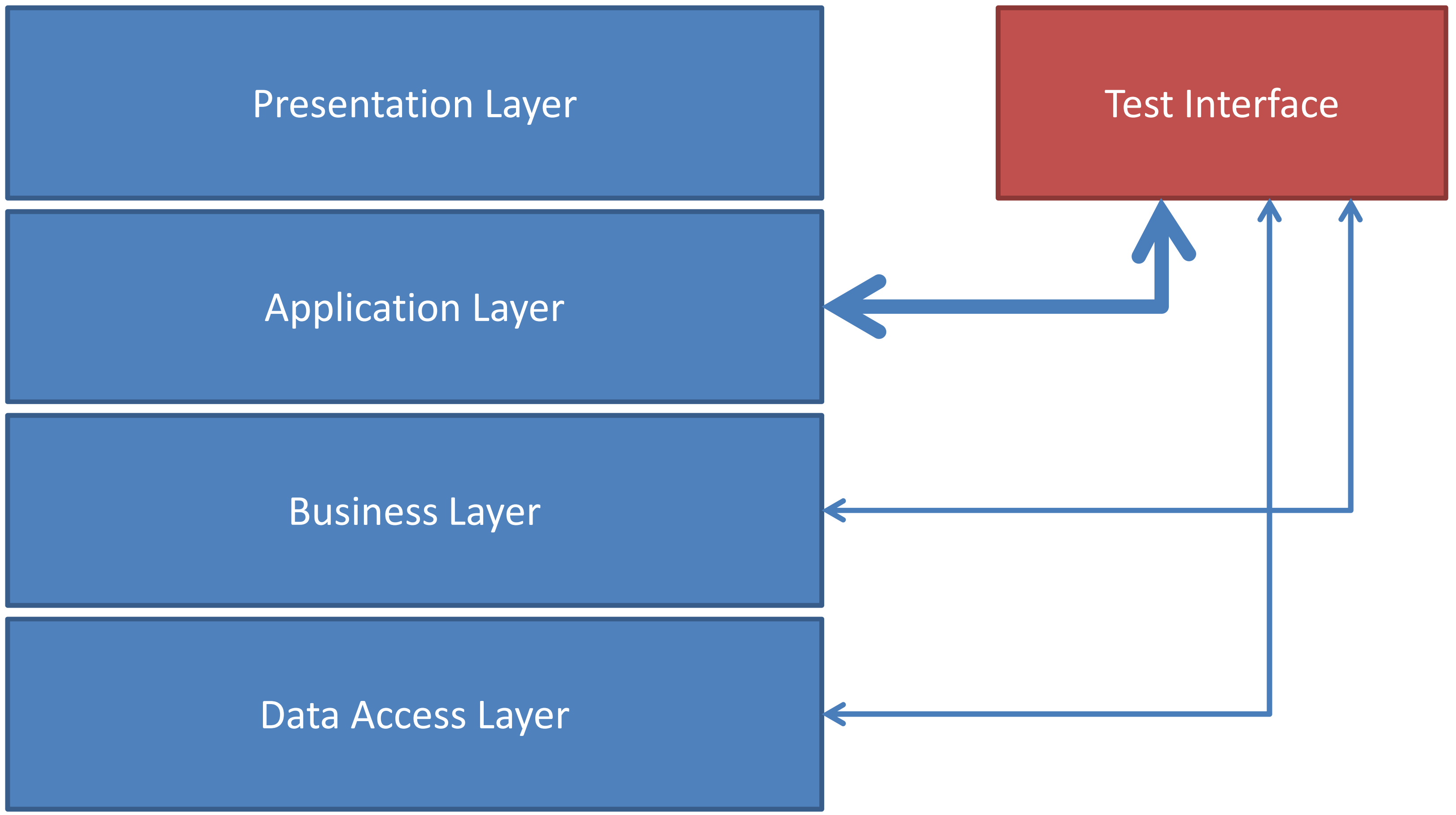
Automated testing

- UnitTests (GoogleTest)
- UI Tests (TestComplete)

And new

- Acceptance Tests

Where to inject Acceptance Test



Acceptance Tests

Specification by Example with Cucumber

```
Given the login dialog is visible
When a registered user provides
  username and password
Then the user is logged in
And the administration module is
  available
```

Which Cucumber binding?

Native C++ binding (cukebins) could not be used, because our application runs with multiple processes on multiple machines.

=> Cucumber with Ruby binding was the natural choice

Acceptance Tests with Cucumber

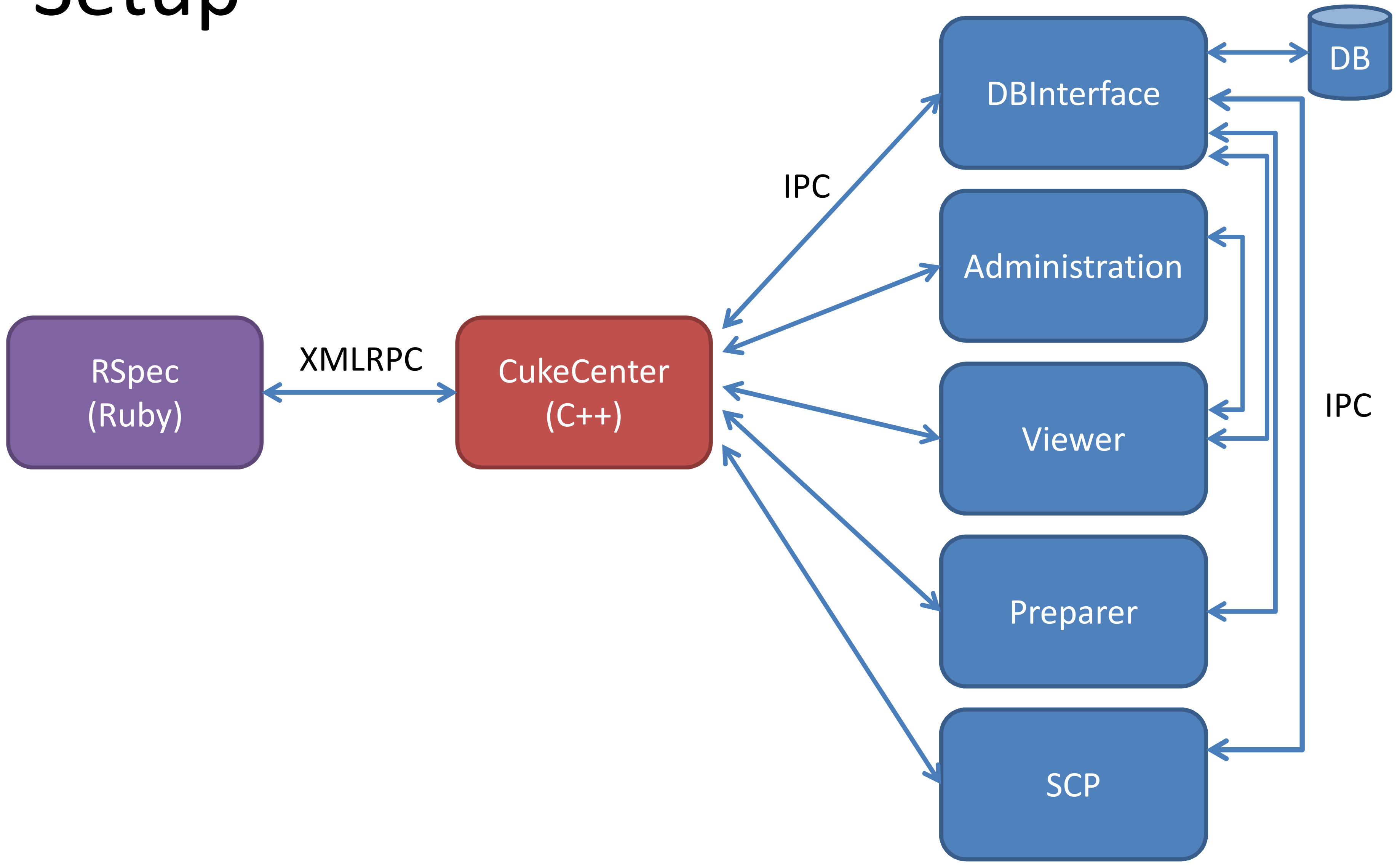
Started very promising

But the tool Cucumber is not capable of handling nested contexts inside a test

Required intensive collaboration with Product Owner

=> New approach with RSpec (Predecessor of Cucumber)

Setup


RSpec
(Ruby)
XMLRPC
CukeCenter
(C++)
IPC
DBInterface
DB
Administration
Viewer
Preparer
SCP
IPC

Let’s write a simple test

```
describe ‘Login mechanism’ do
  context ‘When the login dialog is available’ do
    before (:all) do
      administration.waitUntilLoginIsVisible()
    end

    context ‘And the user logs into the application’ do
      before (:all) do
        administration.login(“user1”, “password4user1”)
      end

      it ‘Then th[illegible] administr[illegible]ion module is a[illegible]ilable for the user’ do
        administr[illegible]ion.waitUn[illegible]AdministrationI[illegible]sible()
      end
    end
  end
end
```

Representative of Administration process

Test method in Administration process

Parameters of login method

Feaze the Ruby part …

For each process a representative Ruby object exists

Ruby’s `method_missing` feature is used to “generate” methods on the fly. So there is no need to specify all methods manually

XMLRPC protocol

```
<methodCall>
  <methodName>cukecommand</methodName>
  <params>
    <param><value><string>ADMISTRATION</string></value></param>
    <param><value><string>login</string></value></param>
    <param><value><i4>60</i4></value></param>
    <param><value>
      <array><data>
        <value><string>user1</string></value>
        <value><string>password4user1</string></value>
      </data></array>
    </value></param>
  </params>
</methodCall>
```

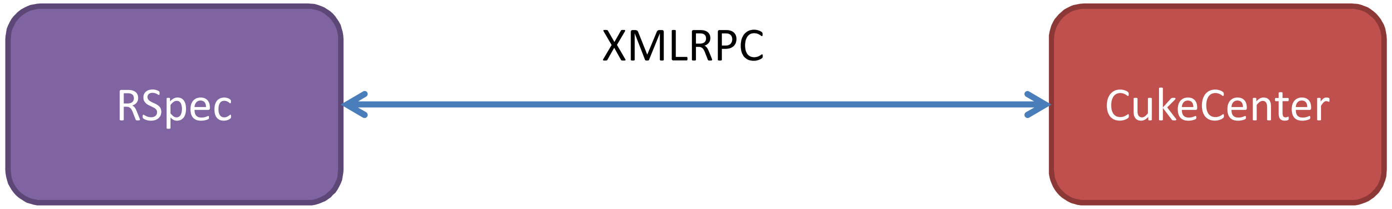
Process name

Method name

Command timeout (s)

Array with all method parameters

CukeCenter



```
<methodCall>
  <methodName>scrcukecommand</methodName>
  <params>
    <param><value><string>ADMISTRATION</string></value></param>
    <param><value><string>login</string></value></param>
    <param><value><i4>60</i4></value></param>
    <param><value>
      <array><data>
        <value><string>user1</string></value>
        <value><string>password4user1</string></value>
      </data></array>
    </value></param>
  </params>
</methodCall>
```

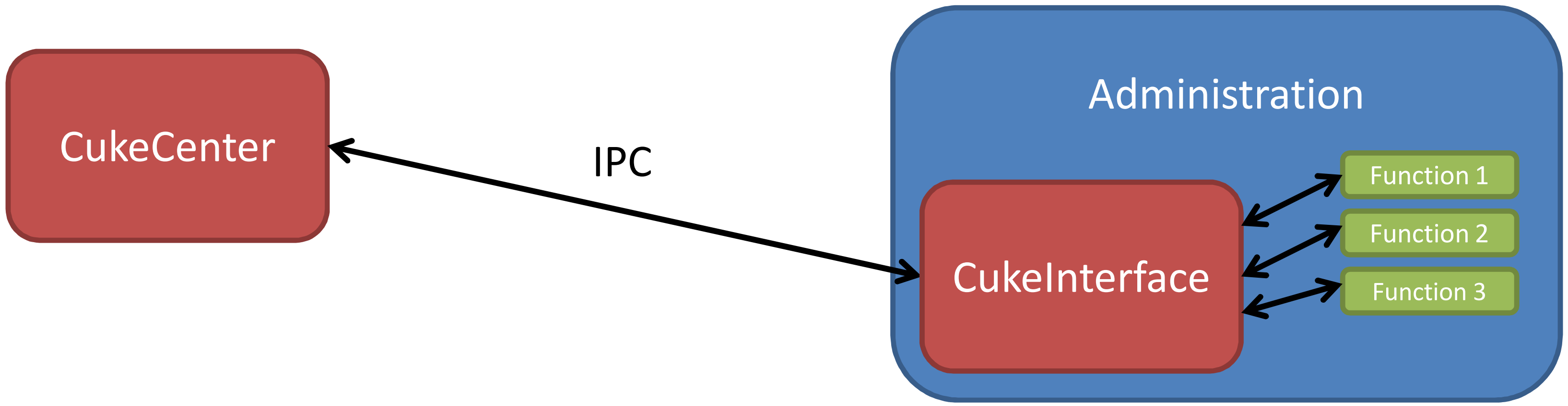
Process lookup

Convert XMLRPC to application specific binary IPC protocol

Limited list of supported types: string, int, double, bool, array, hash

Any nested combination is possible

CukeInterface



Each process has a CukeInterface instance

Special IPC callback

Starts to parse the binary stream and extracts method name

Lookup of registered test method

Calls method with remaining in-stream (Source) and returns new values in out-stream (Sink)

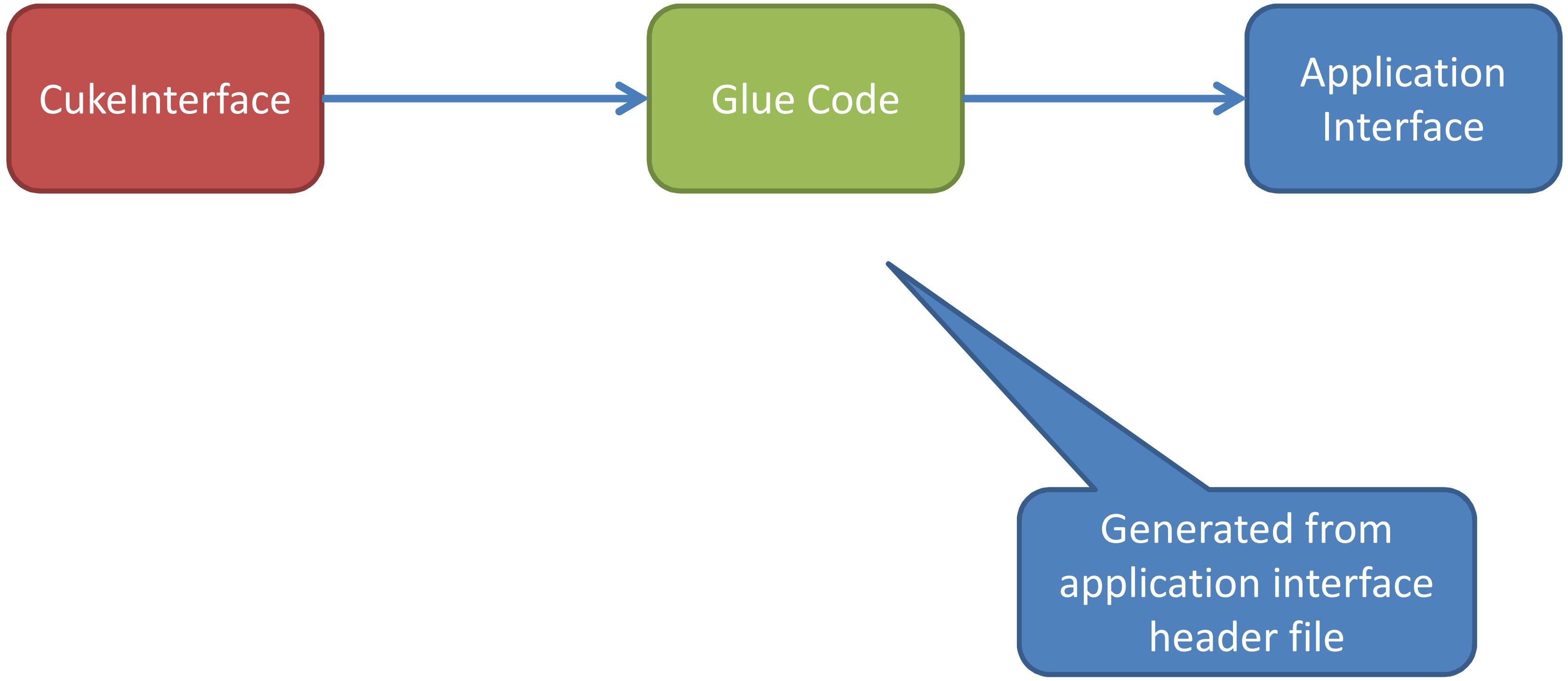
Application test interface

```
class AdminstrationInterface
{
public:
  void userLogin(const std::string& userName,
                 const std::string& password);
  void logout();

  CommandResult waitUntilAdministrationIsVisible();

  static AdministrationInterface s_interface;
};
```

Execution chain in application



Glue Code

All is generated by
Code Generator

```
// defining the test function
void login(const Source& source, Sink& sink);

// registering the function and its name with a registrar
CommandRegistrar(login, "login");


// implementation of the test function
void login(const Source& source, Sink& sink)
{
  auto userName = createFromSource<std::string>(source);
  auto password = createFromSource<std::string>(source);

  s_interface.userLogin(userName, password);
}
```

When to proceed?

Many things in the application happen asynchronously

- Add sleep call into the test script
- Callback from the application into the test could be an option, but would make the application depend on the test
- CukeInterface polls with short interval (100ms) until a certain condition is reached or the command timed out

Test Functions

```
void AdminstrationInterface::userLogin(const std::string& userName,
                                       const std::string& password);
```

Synchronous Call

Which is identified by return value of the test function

Asynchronous Call

```
CommandResult
   AdminstrationInterface::waitUntilAdministrationIsVisible();

enum class CommandResult
{
  Success, // when the condition is fulfilled
  Failed,  // when the condition cannot be fulfilled (anymore)
  Pending  // when the condition is not yet fulfilled
};
```

Asynchronous Test Function

```
CommandResult
AdminstrationInterface::waitUntilAdministrationIsVisible()
{
  if (administrationModule().isVisible())
  {
    return CommandResult::Success;
  }
  return CommandResult::Pending;
}
```

Current Test Status

UnitTests are integrated into the build process

Complete continuous test suites run takes 1h30

Release test cycle takes 2 weeks (main focus is now on exploratory tests)

	2011-03	2014-03	2015-09
UnitTests	603	4588	5413
RSpec Tests	0	1851	4052

Reference

- Continuous Delivery; Jez Humble & David Farley; Addison Wesley, 2010
- Continuous Integration; Stephen M. Matyas, Nicholas Schneider, Mark Voit & Paul Duvall; Addison Wesley, 2007
- Clean Coders – Screen casts by Robert C. Martin
- Effective Programming with Components - Screen casts by Alexander Stepanov
- Cucumber
- RSpec
- GoogleTest
- Why Most UnitTesting is Waste and Segue by James O. Coplien

Contact

Felix Petriconi

Web: http://petriconi.net

Mail: felix@petriconi.net

Twitter: @FelixPetriconi

GitHub: https://github.com/FelixPetriconi

Feedback is always welcome!