Growing OO C++ Software
Guided by Tests

Alan Griffiths

alan@octopull.co.uk

#alanatoctopull

Octopull Limited

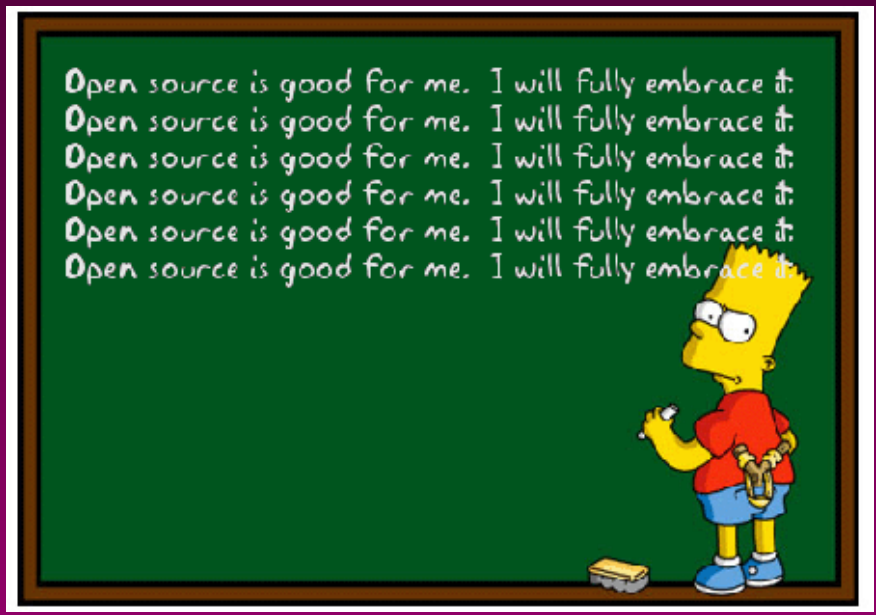
www.octopull.co.uk

Who am I?

- Developed software through many fashions in development processes, technologies and programming languages
- Delivered working software (and development processes)
- A regular at the ACCU conference
- Current ACCU Chair
- Written for magazines and books
- Spoken at a number of conferences
- Made many friends in developer communities

A project I can talk about

- Usually clients want confidentiality
  - Canonical doesn't
- This is an open-source (LGPL/GPL) project
- Documentation and code is online
  - I've borrowed shamelessly
  - Links at the end



Canonical and me

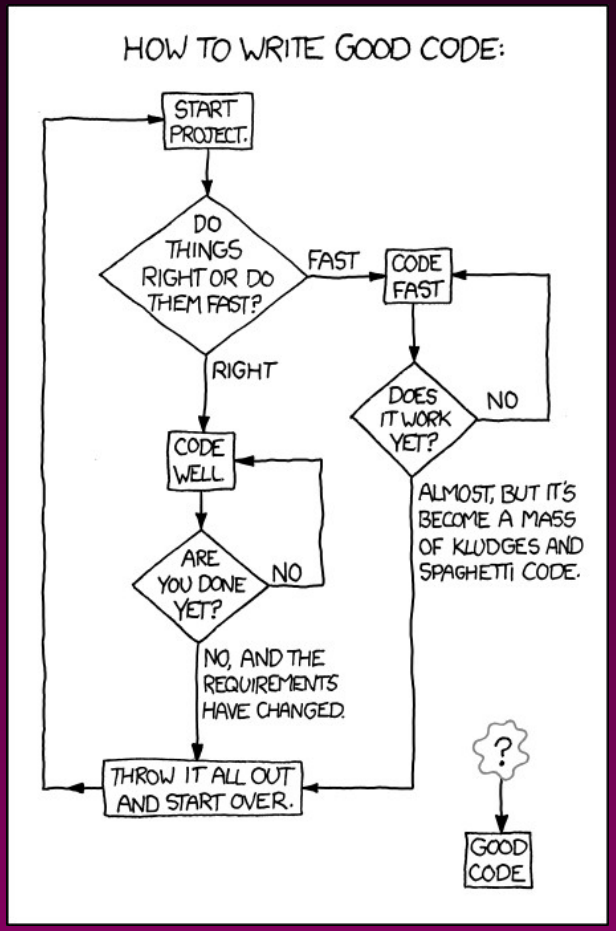
- Canonical make Ubuntu Linux
- Their “Unity” user interface
  - Is a Compiz plugin
  - Designed to cover PC, tablet, phone and TV
- Contracted me to
  - Improve quality of Compiz codebase
  - Review Compiz and alternative codebases
- Then to work on “something new”

Rewriting Software

- The organisation hasn't changed
- Existing software incorporates many lessons
- Often a moving target

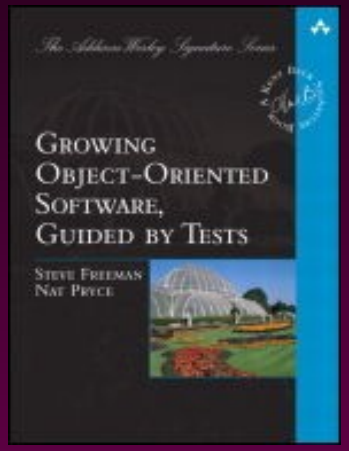
- Unless *significant*
  - Changes in requirements
  - New technology

Writing good code



http://xkcd.com/844/

Growing Object-Oriented Software, Guided by Tests



By: Steve Freeman & Nat Pryce

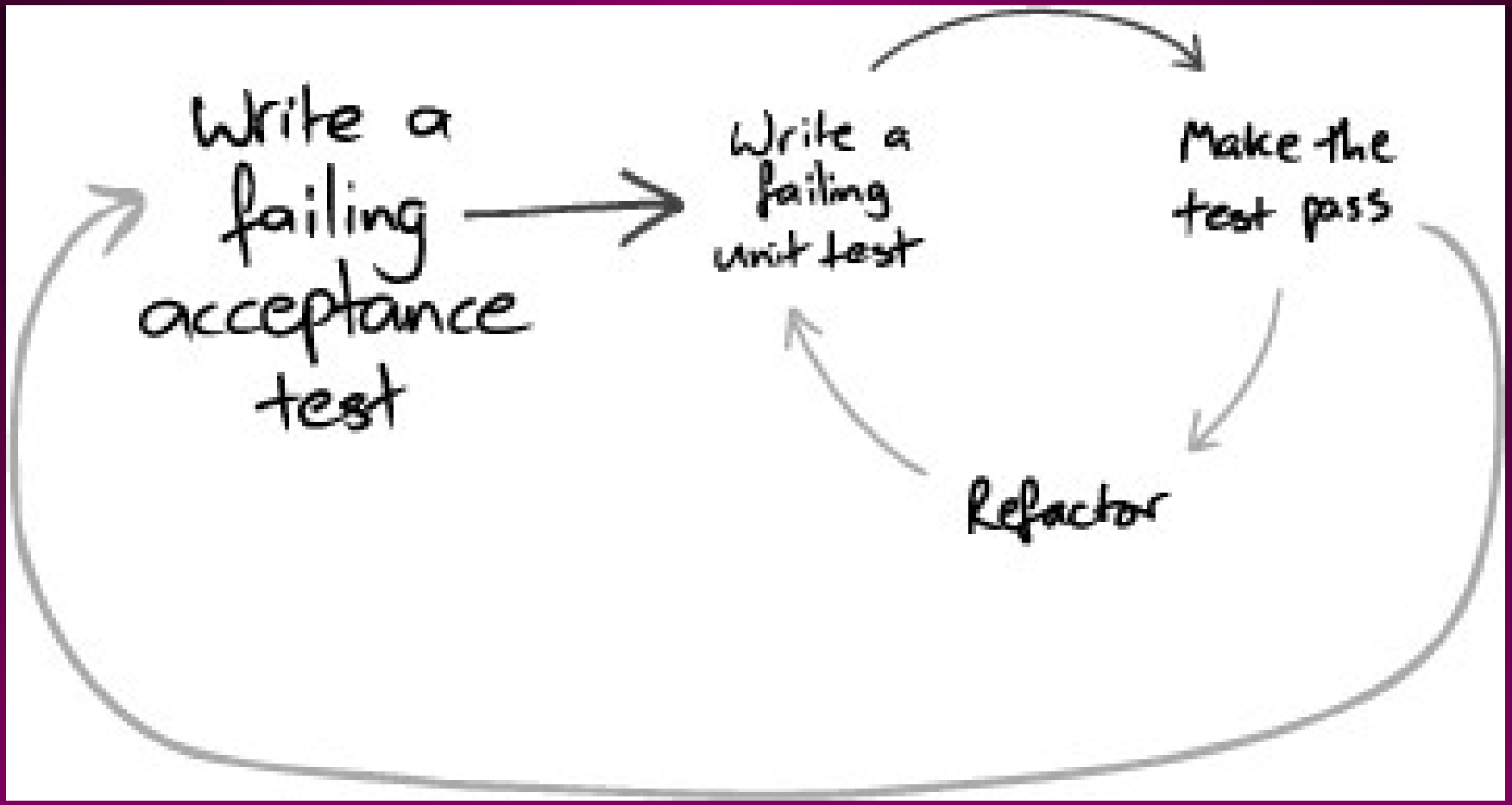
http://www.growing-object-oriented-software.com/

- **Required reading for team**
- **Inspiration for development approach**
- **A good read**

GOOS - Chapter 1
What is the Point of Test-Driven Development?

- Software Development as a Learning Process
- Feedback Is the Fundamental Tool
- Practices That Support Change
- Test-Driven Development in a Nutshell
- The Bigger Picture
- Testing End-to-End
- Levels of Testing
- External and Internal Quality

GOOS - Chapter 1
Inner and outer feedback loops in TDD



GOOS - Chapter 1
Feedback Is the Fundamental Tool

"In a project organized as a set of nested feedback loops, development is incremental and iterative.

"Incremental development builds a system feature by feature, instead of building all the layers and components and integrating them at the end. Each feature is implemented as an end-to-end "slice" through all the relevant parts of the system. The system is always integrated and ready for deployment.

"Iterative development progressively refines the implementation of features in response to feedback until they are good enough."

“You Can't do TDD in C++”

- Chris Matts “challenge” at XtC
  - His clients made this assertion
- News to me – I've been doing it for years
  - TDD is possible in C++
    - (although *all* tooling in C++ is hard)
  - TDD isn't the answer to all problems
    - (but is a big improvement on the usual hack & fix)

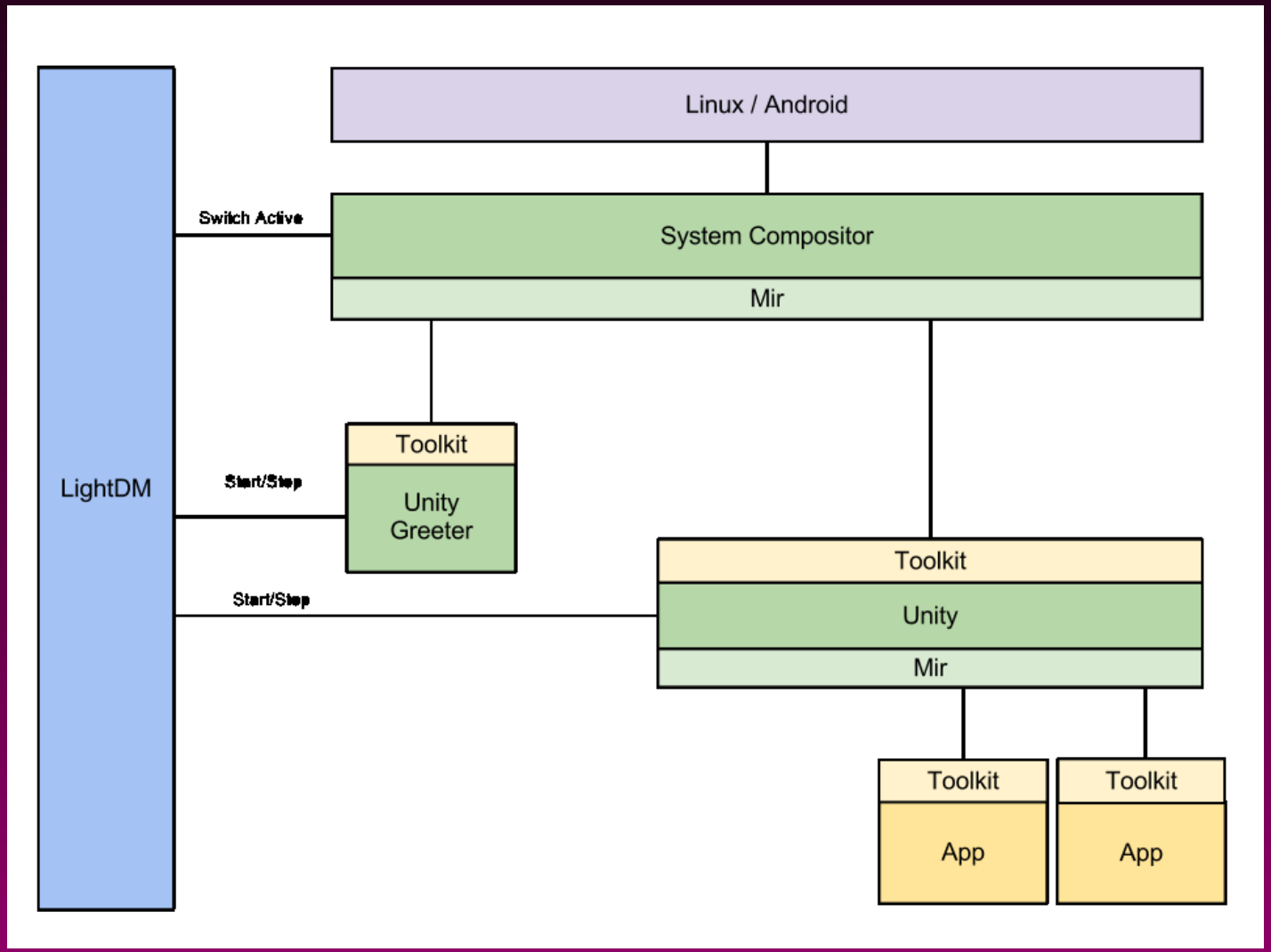
Project mir - the goals

- Support the Unity shell...
- Across multiple form factors
  - Phone, tablet, TV, PC, ...
- Provide driver independence
  - Mesa, Nvidia, Android, …
- Exploit GPU acceleration
- System and Session compositing
- Support for Qt and, possibly, other toolkits
- Support for X applications

The Project: Mir
a new Window Manager

- X-Windows+Compiz/kwin/... is successful but:
  - Designed with distributed application and server
  - Pre-dates sharing GPUs
  - Lots of legacy baggage
- Wayland is great but...
  - Is a protocol, not an implementation
  - Covers functions Canonical didn't need
- Weston (reference implementation of Wayland)
  - Not optimised for Canonical's needs
  - In development (with lots of other stakeholders)

Canonical: Graphics Stack



Multiple platforms

- Ubuntu 12.10
  - g++ 4.6

- Ubuntu 13.x
  - g++ 4.7

- Android NDK
  - g++ 4.4

- Android/libhybris
  - g++ 4.7

Multiple Locations

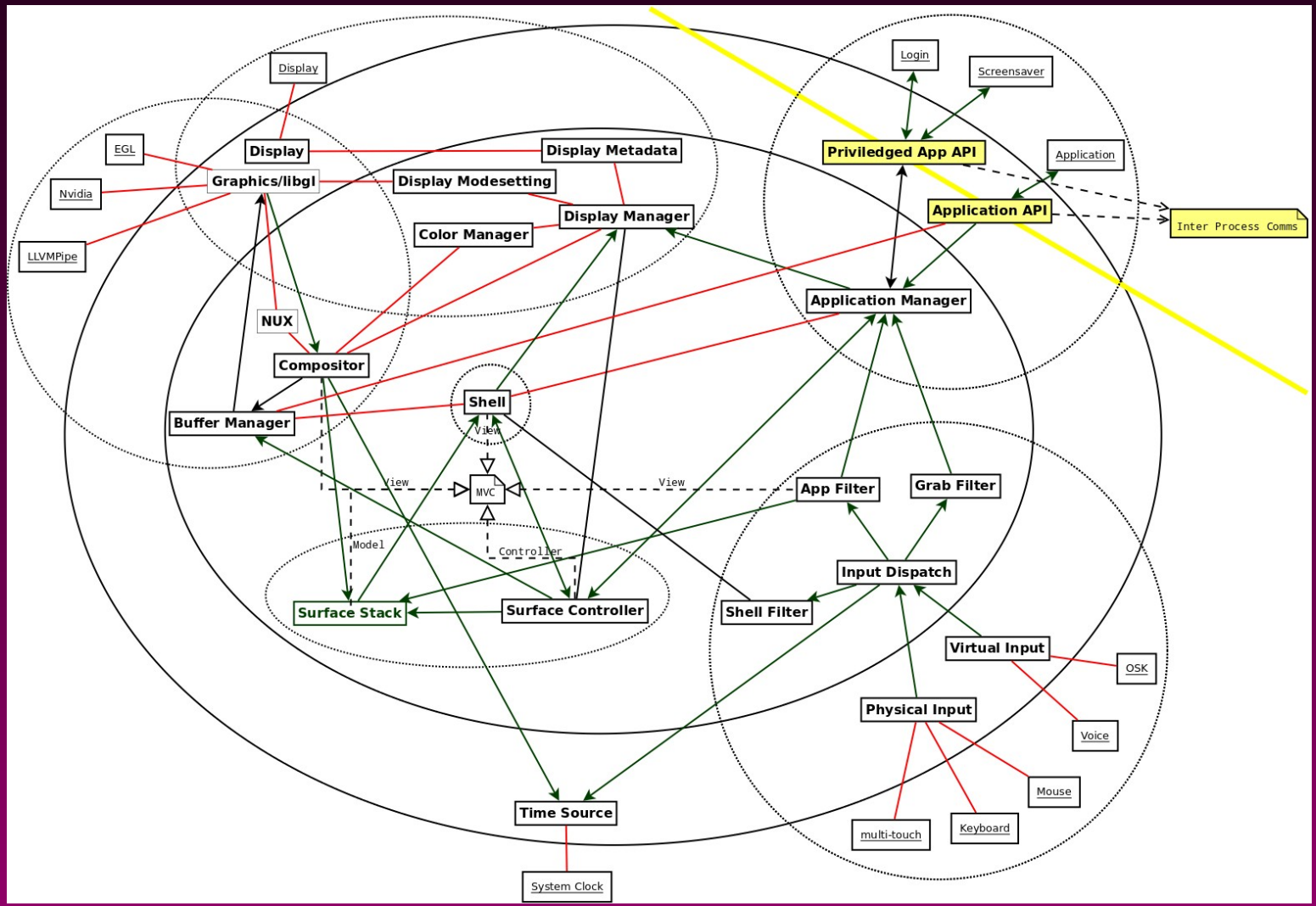
- Nottingham, UK
- Athens, Greece
- Bochum, Germany
- San Diego, CA, USA
- Carlisle, PA, USA
- Perth, Australia
- ...

"The most efficient and effective method of conveying information to and within a development team is face-to-face conversation" Agile Manifesto

Forming The Team

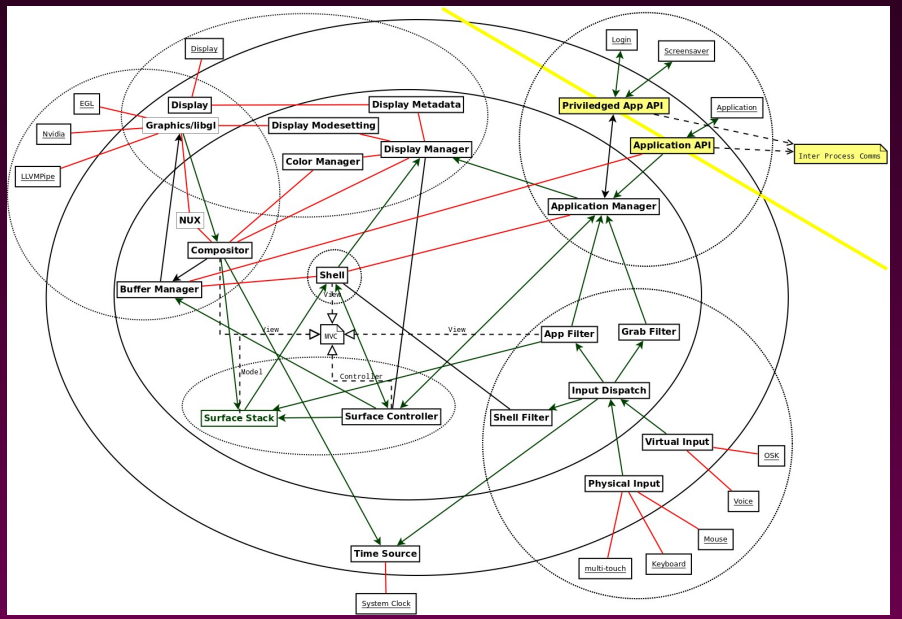
- A “Sprint” [Canonical's term]
  - Getting the developers together for, typically, a week
  - Didn't quite work that way – half the team were in Boston USA and half in London UK
  - Agreed development approach – GOOS, C++, CMake, GoogleTest/Mock, Jenkins, bzr
  - High level design map

The High Level Design



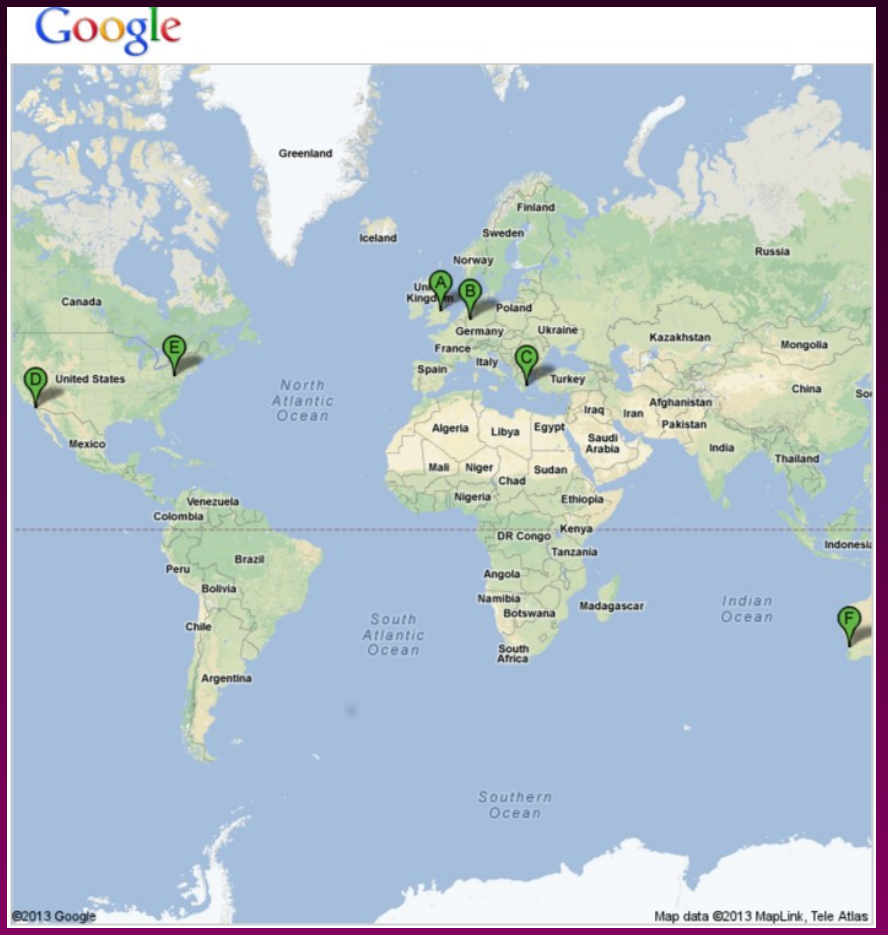
Development approach

- Initially worked on the core loop – with “acceptance tests” interfacing inside the outer boundary
- Automating tests for code dealing with real drivers (graphics or input) limited



Dealing with Multiple Locations

- Standups & meetings
- Pairing vs code reviews
- Documentation



Distributed standups & other meetings

- Early in the project
  - Google Hangouts for standup and discussions
  - Skype for one-one
  - Email and IRC used, but mostly to arrange above
- Later in the project
  - Most discussions on IRC and email
  - Some Google Hangouts for other discussion

Remote pairing

- Pair programming at a distance
  - TeamViewer
  - Different timezones
  - Different editors
  - Bandwidth

Code reviews

- Code reviews of merge proposals
  - Less “bandwidth”
  - More “latency”
  - Some frustration
  - But easier to schedule across timezones
- “Merge Proposals”
  - managed by Launchpad website
    - Shows diff
    - Shows comments and review status
  - Monitored by Jenkins
    - Builds supported configurations
    - Merges “Approved” changes

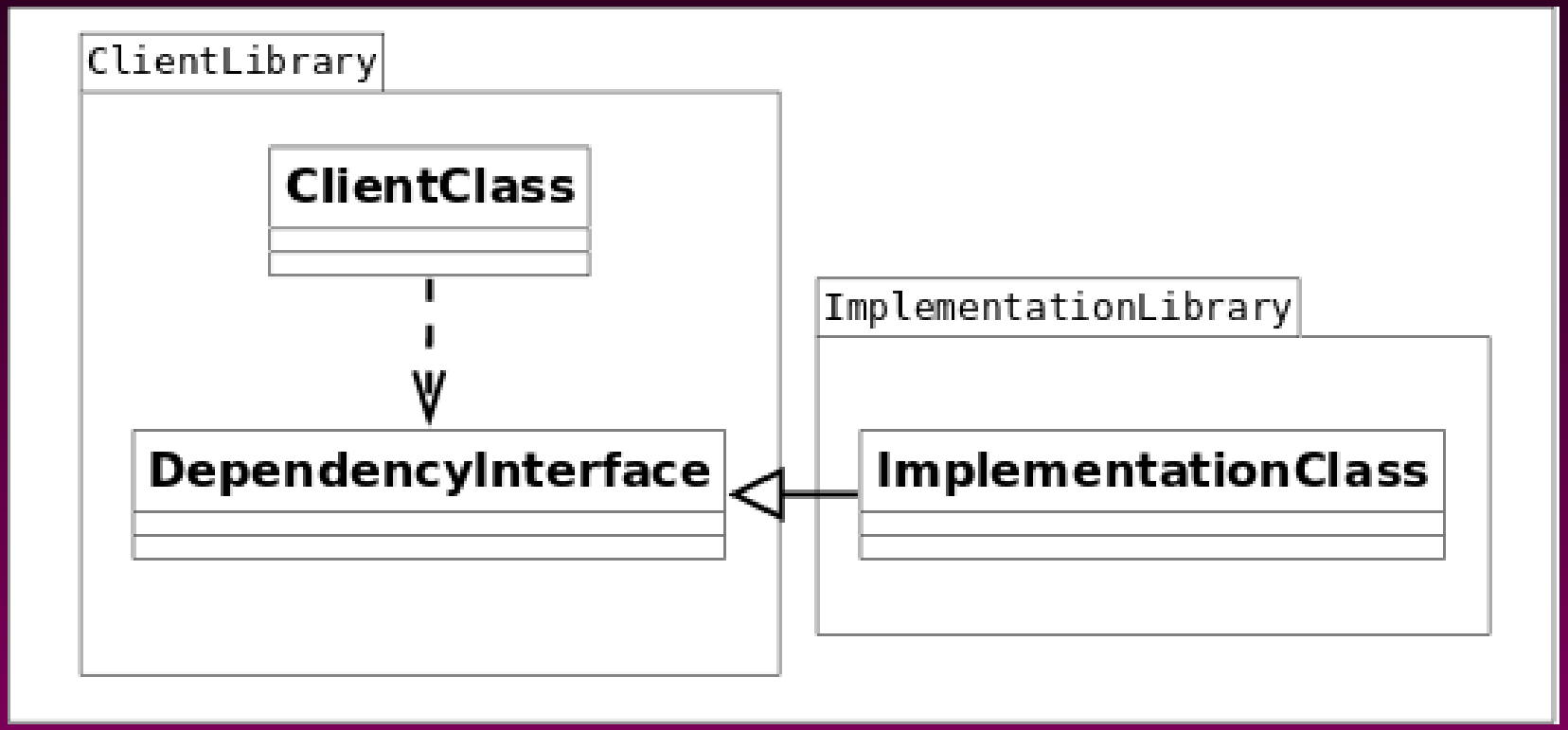
Documentation

- Launchpad
  - Source control
  - Merge Proposals
  - Bugs
  - Blueprints
- Source code
  - Doxygen (code + markdown)
  - Design images
- Google docs
  - Discussion documents
  - Working notes
  - Not deliverables/public
- Email
- Wiki

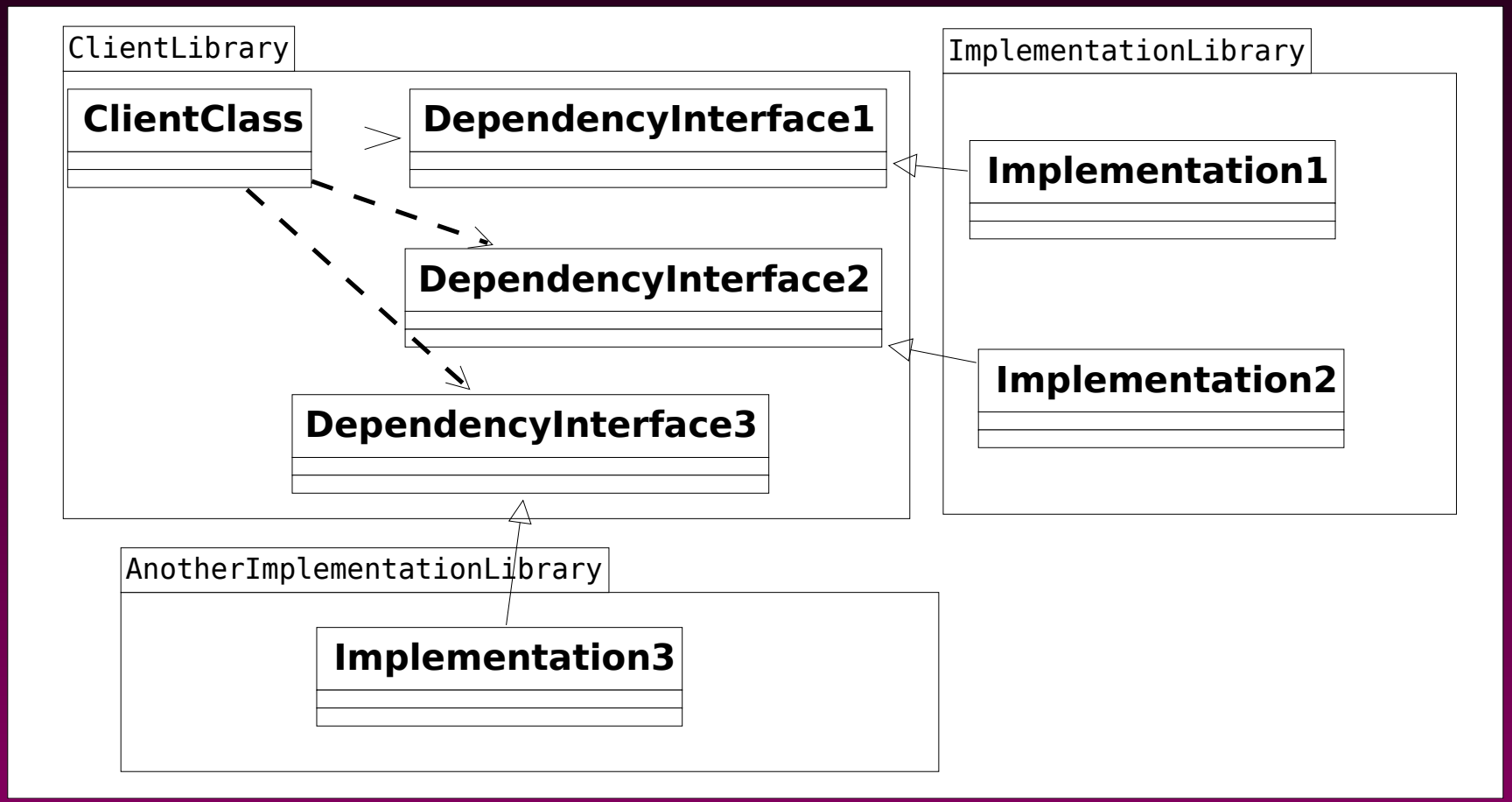
Documentation – in the code

```
// TODO comparing strings in an if-else chain isn't efficient.
// It is probably possible to generate a Trie at compile time.
if ("connect" == invocation.method_name())
{
    invoke(&protobuf::DisplayServer::connect, invocation);
}
else if ("create_surface" == invocation.method_name())
...
```

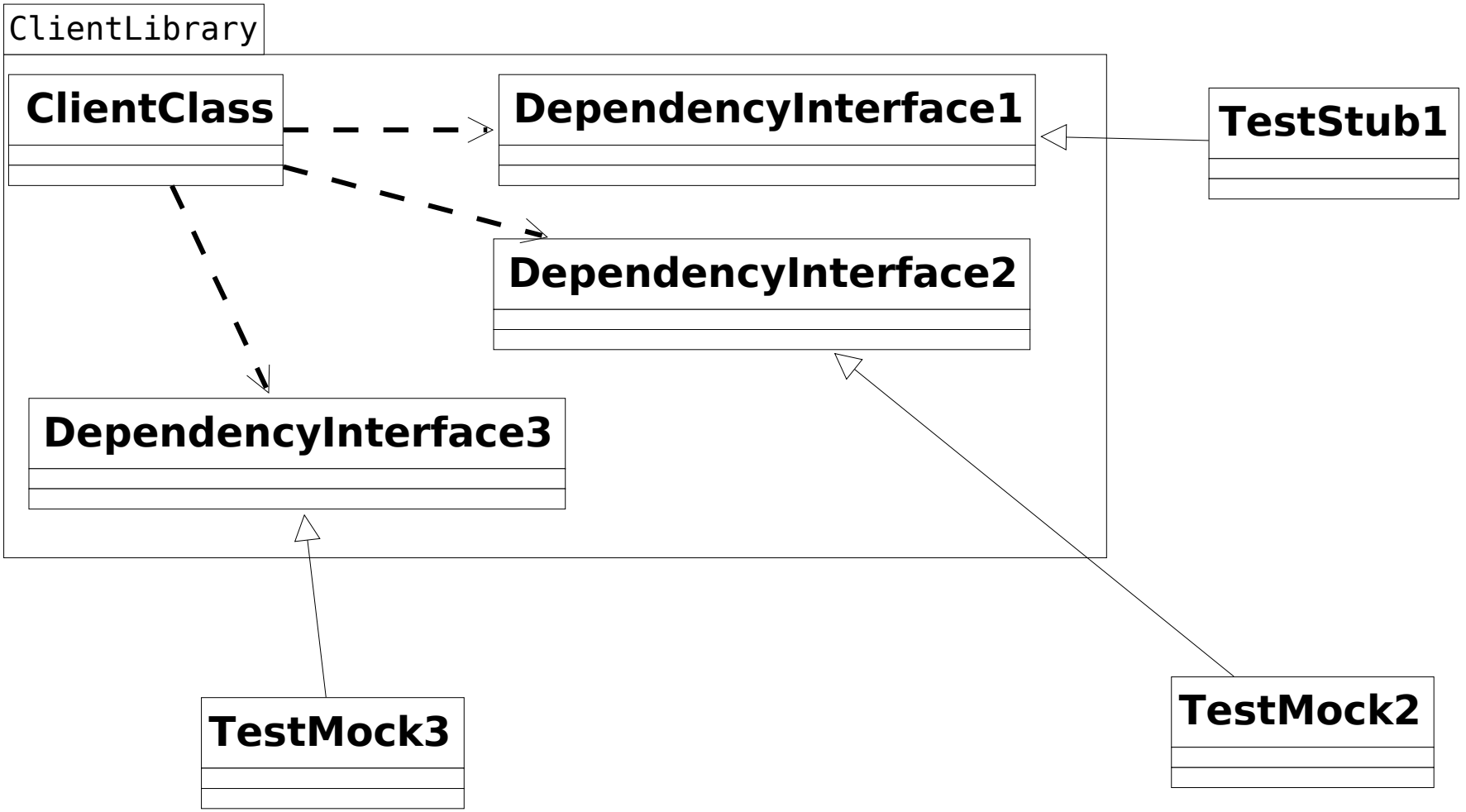
TDD : use interfaces



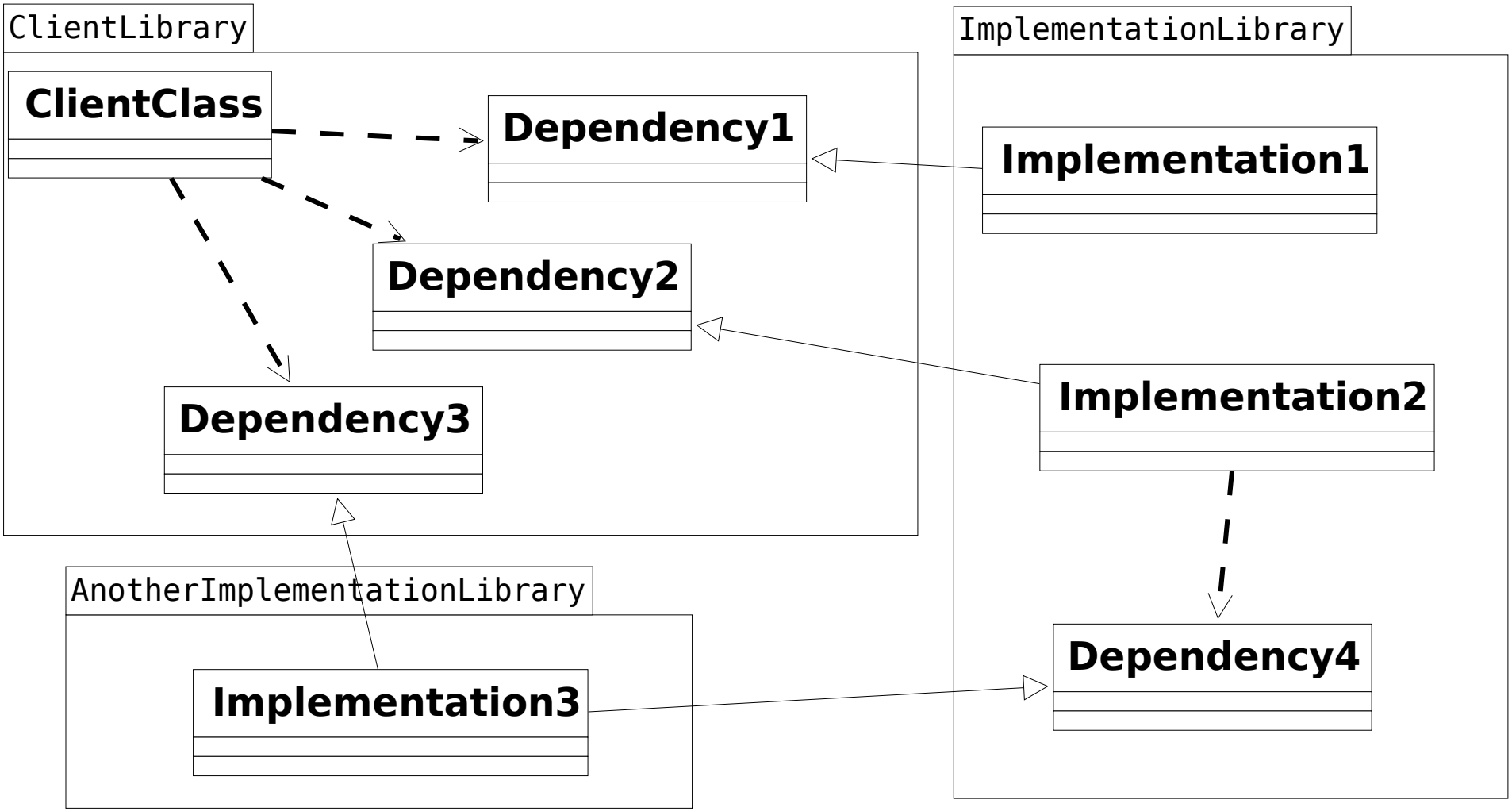
Design using Interfaces



Unit Testing



Design using Interfaces



A unit test

```
struct ShellSurface : testing::Test
{
    std::shared_ptr<mi::InputChannel> const null_input_channel;
    MockSurfaceBuilder surface_builder;
};

TEST_F(ShellSurface, creation_and_destruction)
{
    using namespace testing;

    mf::SurfaceCreationParameters const params;

    InSequence sequence;
    EXPECT_CALL(surface_builder, create_surface(params)).Times(1);
    EXPECT_CALL(surface_builder, destroy_surface(_)).Times(1);

    msh::Surface test(
        mt::fake_shared(surface_builder),
        params,
        null_input_channel);
}
```

A mock object

```
class MockSurfaceBuilder : public msh::SurfaceBuilder
{
public:
    MockSurfaceBuilder()
    {
        using namespace testing;
        ON_CALL(*this, create_surface(_)).
            WillByDefault(Invoke(&self, &StubSurfaceBuilder::create_surface));

        ON_CALL(*this, destroy_surface(_)).
            WillByDefault(Invoke(&self, &StubSurfaceBuilder::destroy_surface));
    }

    MOCK_METHOD1(create_surface, std::weak_ptr<ms::Surface> (const mf::SurfaceCreationParameters&));

    MOCK_METHOD1(destroy_surface, void (std::weak_ptr<ms::Surface> const&));

private:
    StubSurfaceBuilder self;
};
```

A stub object

```
class StubSurfaceBuilder : public msh::SurfaceBuilder
{
public:
    StubSurfaceBuilder() :
        buffer_bundle(new mtd::NullBufferBundle()),  dummy_surface()
    {
    }

    std::weak_ptr<ms::Surface> create_surface(mf::SurfaceCreationParameters const&)
    {
        dummy_surface = std::make_shared<ms::Surface>(mf::a_surface().name, buffer_bundle);
        return dummy_surface;
    }

    void destroy_surface(std::weak_ptr<ms::Surface> const&)
    {
        dummy_surface.reset();
    }

private:
    std::shared_ptr<ms::BufferBundle> const buffer_bundle;
    std::shared_ptr<ms::Surface> dummy_surface;
};
```

A modified test

```
TEST_F(ShellSurface, create_throws_means_no_destroy)
{
    using namespace testing;

    mf::SurfaceCreationParameters const params;

    InSequence sequence;
    EXPECT_CALL(surface_builder, create_surface(params)).Times(1)
        .WillOnce(Throw(std::runtime_error(__PRETTY_FUNCTION__)));
    EXPECT_CALL(surface_builder, destroy_surface(_)).Times(Exactly(0));

    EXPECT_THROW({
        msh::Surface test(
            mt::fake_shared(surface_builder),
            params,
            null_input_channel);
    }, std::runtime_error);
}
```

Configuring dependencies

- Dependencies are supplied to constructors
  - Objects are known by multiple interfaces; but,
  - The initial system state has “one of each”
- A DefaultServerConfiguration class
  - Caches the created objects
  - Knows what to supply for each interface
- A TestingServerConfiguration class overrides
  - For integration & acceptance tests

ServerConfiguration

```
class ServerConfiguration
{
public:
    virtual std::shared_ptr<frontend::Communicator>   the_communicator() = 0;
    virtual std::shared_ptr<shell::SessionStore>      the_session_store() = 0;
    virtual std::shared_ptr<graphics::Display>        the_display() = 0;
    virtual std::shared_ptr<compositor::Drawer>       the_drawer() = 0;
    virtual std::shared_ptr<input::InputManager>      the_input_manager() = 0;

protected:
    ServerConfiguration() = default;
    virtual ~ServerConfiguration() = default;

    ServerConfiguration(ServerConfiguration const&) = delete;
    ServerConfiguration& operator=(ServerConfiguration const&) = delete;
};
```

main()

```
int main(int argc, char const* argv[])
try
{
    mir::DefaultServerConfiguration config(argc, argv);

    run_mir(config);
    return 0;
}
catch (mir::AbnormalExit const& error)
{
    std::cerr << error.what() << std::endl;
    return 1;
}
catch (std::exception const& error)
{
    std::cerr << "ERROR: " << boost::diagnostic_information(error) << std::endl;
    return 1;
}
```

run_mir()

```
void run_mir(mir::ServerConfiguration& config)
{
    signal(SIGINT, signal_terminate);
    signal(SIGTERM, signal_terminate);
    signal(SIGPIPE, SIG_IGN);

    mir::DisplayServer server(config);

    signal_display_server.store(&server);

    server.run();
}
```

```
std::atomic<mir::DisplayServer*> signal_display_server;

extern "C" void signal_terminate(int)
{
    while (!signal_display_server.load())
        std::this_thread::yield();

    signal_display_server.load()->stop();
}
```

Configuration magic

```
CachedPtr<graphics::DisplayReport>      display_report;
```

```
std::shared_ptr<mg::DisplayReport>
mir::DefaultServerConfiguration::the_display_report()
{
    return display_report(
        [this] -> std::shared_ptr<graphics::DisplayReport>
        {
            if (the_options()->get(log_display, false))
            {
                return std::make_shared<ml::DisplayReport>(the_logger());
            }
            else
            {
                return std::make_shared<mg::NullDisplayReport>();
            }
        });
}
```

Project Testing Framework

- Test process
  - Start server process
  - Start client process(es)
  - Wait for client(s) to complete
  - Signal server to exit
  - Wait for the server to exit
  - Validate expectations & assertions
- Facilitates
  - Injecting test doubles into server
  - Injecting test code into server
  - Injecting test code into client(s)

Configuring Server for Tests

```
class TestingServerConfiguration : public DefaultServerConfiguration
{
public:
    ...

    // Code to run in server process
    virtual void exec(DisplayServer* display_server);

    // Code to run in server process after server exits
    virtual void on_exit(DisplayServer* display_server);
    ...

    // We override the_input_manager in the default server configuration
    // to avoid starting and stopping the full android input stack for tests
    // which do not leverage input.
    virtual std::shared_ptr<input::InputManager> the_input_manager();
    ...
};
```

Starting a server with a mock

```
TEST_F(ApplicationMediatorReport, application_connect_called)
{
    struct Server : TestingServerConfiguration
    {
        std::shared_ptr<mf::ApplicationMediatorReport>
        the_application_mediator_report()
        {
            auto result = std::make_shared<MockApplicationMediatorReport>();

            EXPECT_CALL(*result, application_connect_called(testing::_)).
                Times(1);

            return result;
        }
    } server_processing;

    launch_server_process(server_processing);

    ...
```

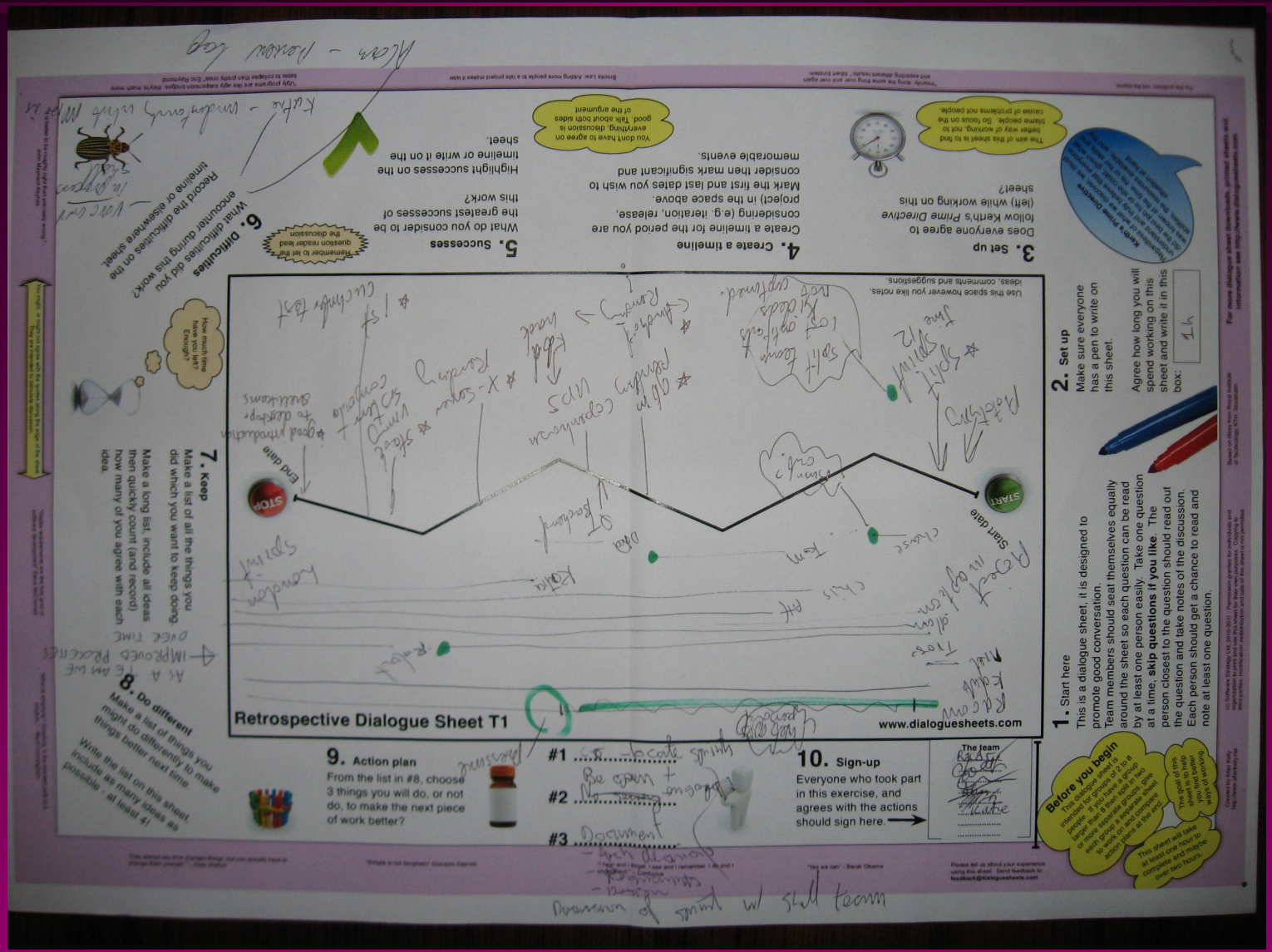
Starting a test client process

```
...
struct Client: TestingClientConfiguration
{
    void exec()
    {
        mt::TestProtobufClient client(mtf::test_socket_file(), rpc_timeout_ms);

        client.connect_parameters.set_application_name(__PRETTY_FUNCTION__);
        EXPECT_CALL(client, connect_done()).Times(testing::AtLeast(0));

        client.display_server.connect(
            0,
            &client.connect_parameters,
            &client.connection,
            google::protobuf::NewCallback(&client, &mt::TestProtobufClient::connect_done));

        client.wait_for_connect_done();
    }
} client_process;

launch_client_process(client_process);
}
```

Retrospective

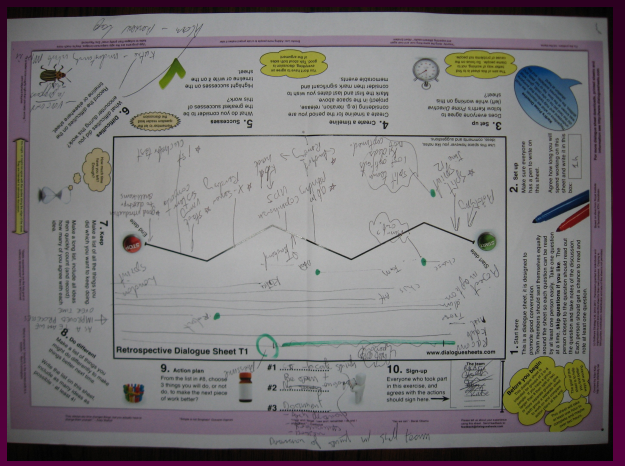
"At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly." - Agile Manifesto

Retrospective



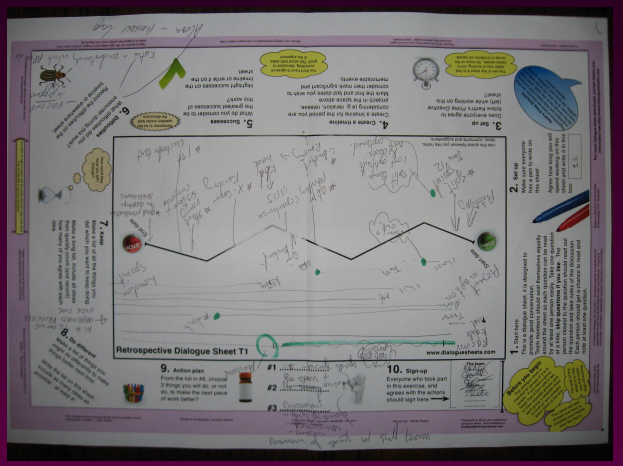
Retrospective

- Timeline
  - First (split) “sprint”
  - First GBM rendering
  - QT backend
  - Copenhagen “sprint”
    - Android rendering
  - X-server rendering
  - Running as system compositor
  - 1st Cucumber test
  - London “sprint”



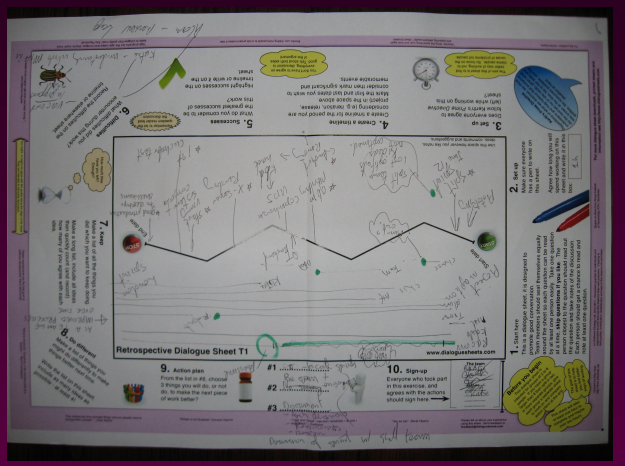
Retrospective

- Difficulties
  - feedback lag in code reviews
  - Understanding what mir is
  - Having an “in process” shell



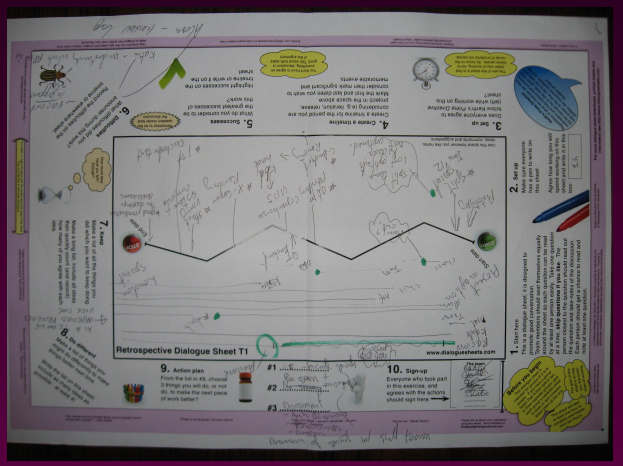
Retrospective

- Keep
  - Tests
  - Code reviews
  - Power to veto
  - Maturity of discussion & receptiveness to criticism
  - Hangouts
  - “stream of consciousness” on IRC
  - Conservative
  - High quality design



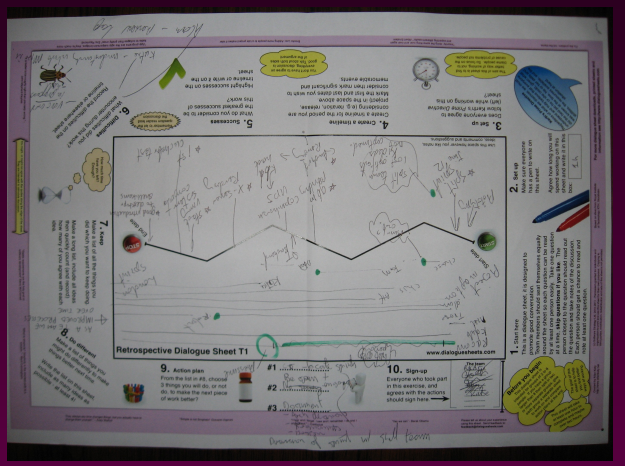
Retrospective

- Do different
  - Co-locate first sprint
  - Reduce turnover
    - Better support for employee development
    - Better support to prevent “burnout”
  - Secrecy
  - Language experimentation
  - Start earlier

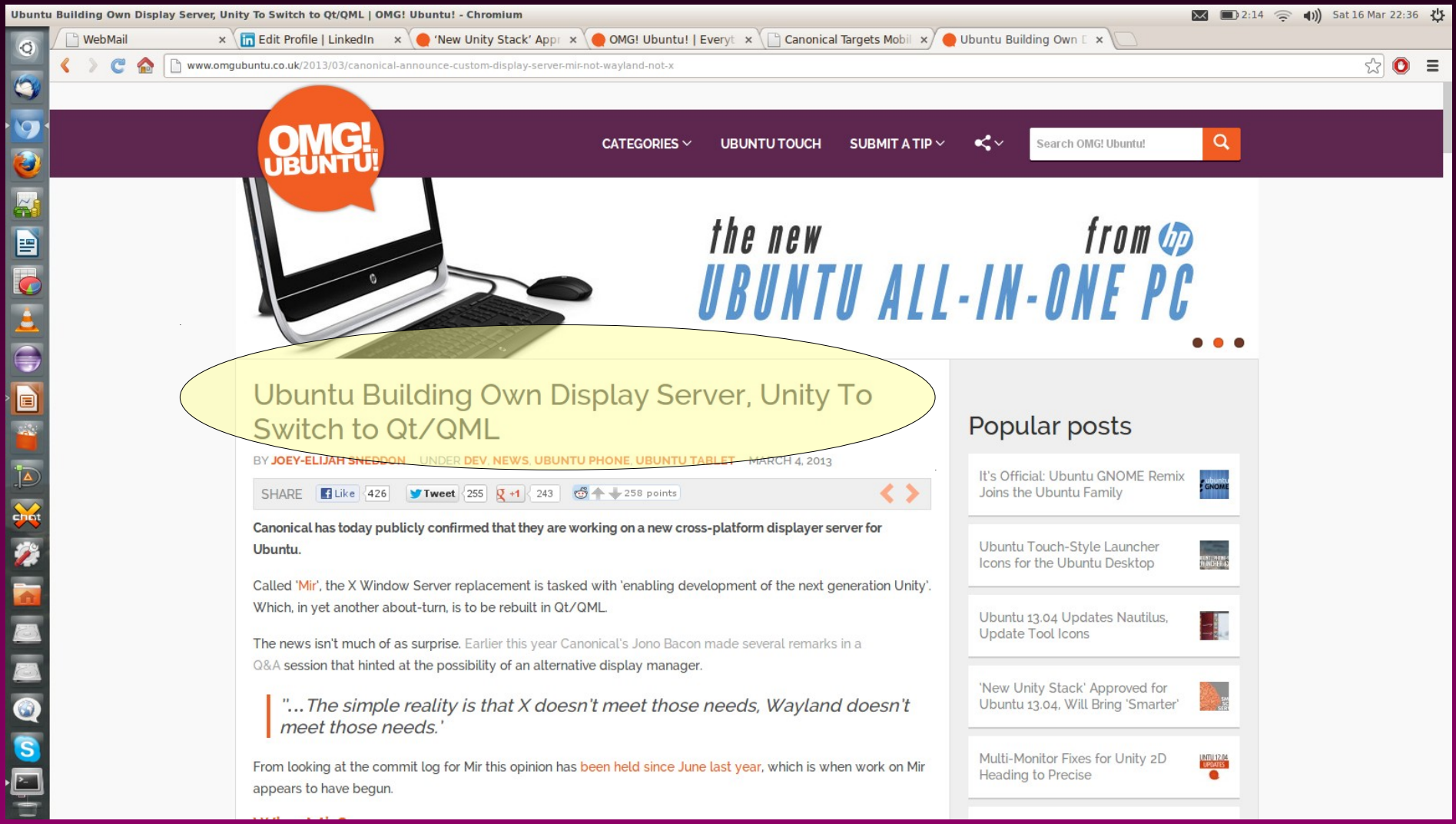


Retrospective

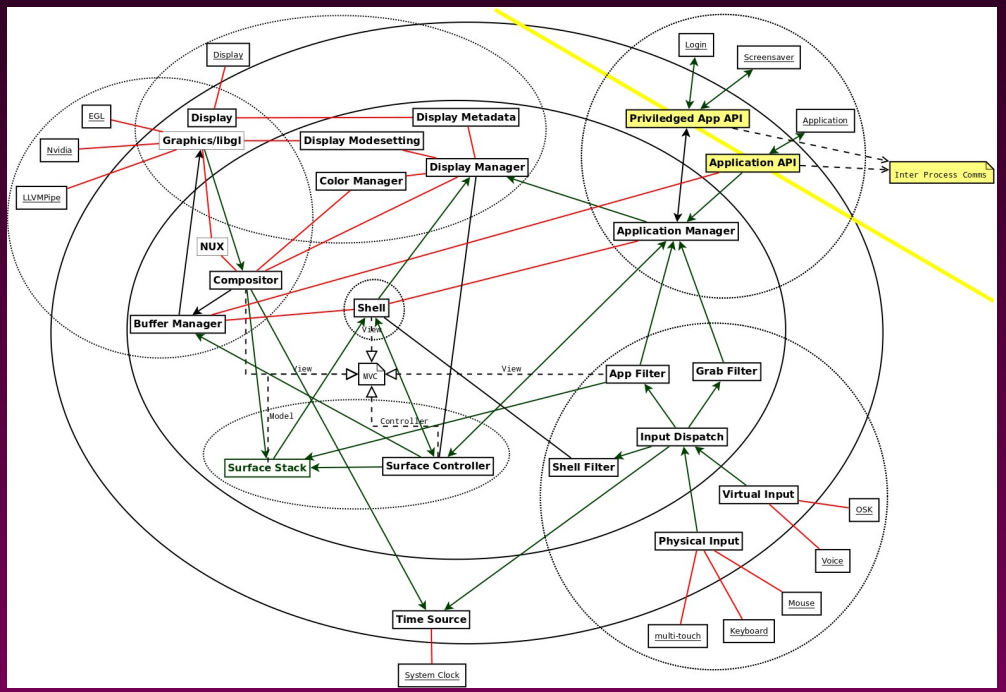
- Action plan
  - Co-locate sprints
  - Be open and engage with stakeholders
  - Document
    - Architecture decisions
    - Requirements
    - Design

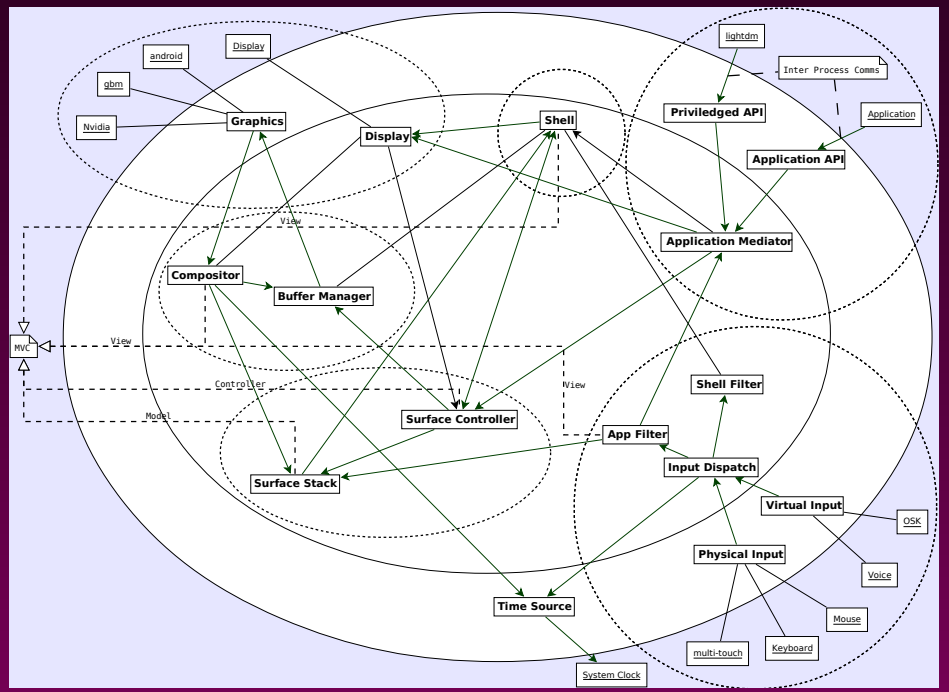


Public release

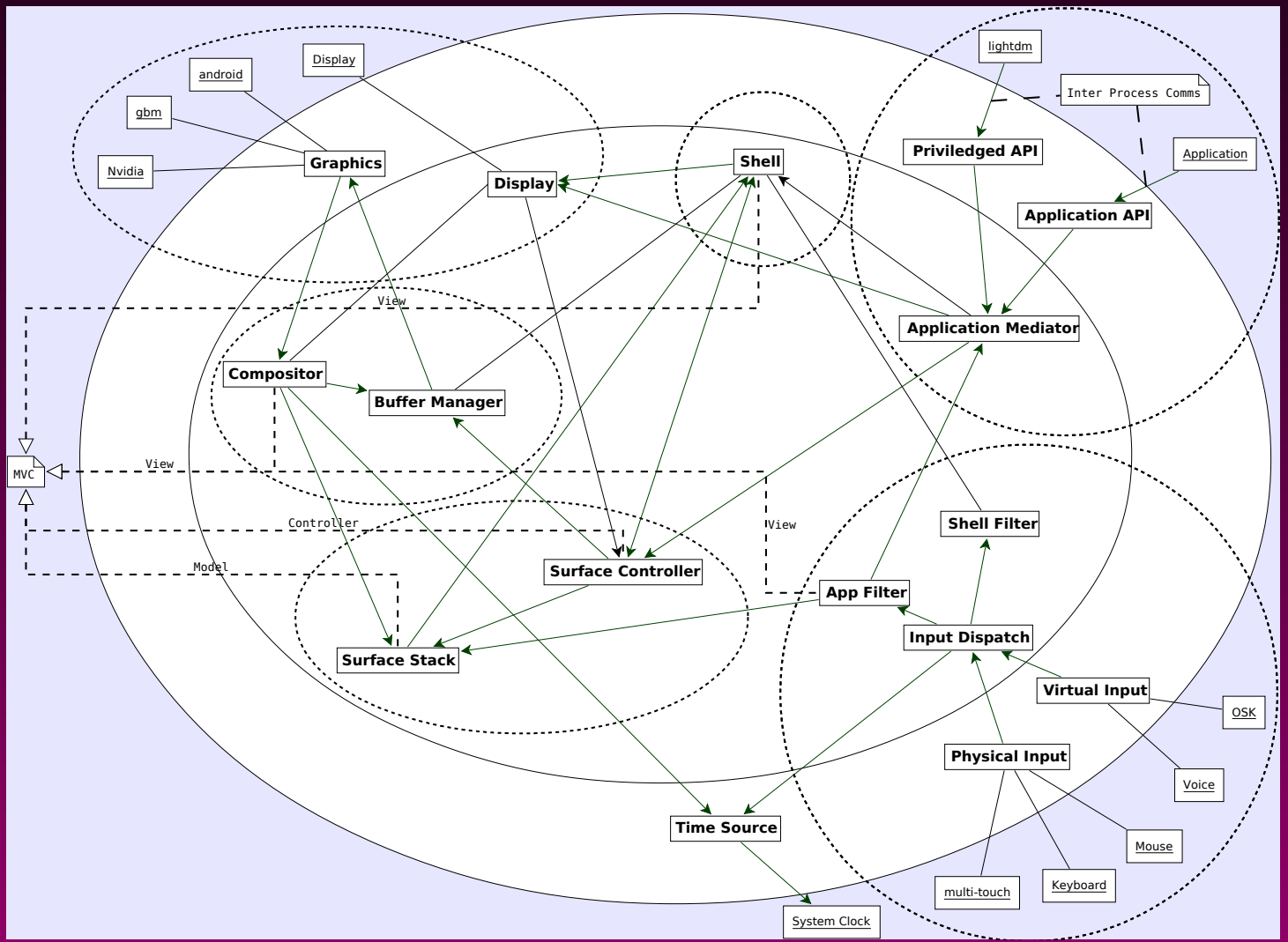


Design update





Design update



Feedback

Can I just say (from poking around the codebase) that it's fantastic to see Canonical producing lovely C++ code like this!

Particularly I like:
* Use of modern C++ features like scoped locks, smart pointers and auto types
* Good use of namespaces and matching directory paths
* Using protobuf for wire formats
* Good quantity of tests and use of google mock
* Nice clean CMake scripts
* Minimal debian packaging rules

I have no constructive input to add! I just wanted to highlight these things.

Cheers
Pete

Links and References

- mir
  - https://launchpad.net/mir
  - http://unity.ubuntu.com/mir/
  - https://wiki.ubuntu.com/MirSpec
- GOOS
  - http://www.growing-object-oriented-software.com/
- TeamViewer
  - http://www.teamviewer.com
- Canonical
  - http://www.canonical.com/
- g++
  - http://gcc.gnu.org/
- cmake
  - http://www.cmake.org/
- Jenkins
  - http://jenkins-ci.org/
- Skype
  - http://www.skype.com/
- Android
  - http://www.android.com/
- Dialogue sheets
  - http://www.softwarestrategy.co.uk/dlgsheets/
- Movie
  - http://www.youtube.com/watch?v=7grEFrTBzus