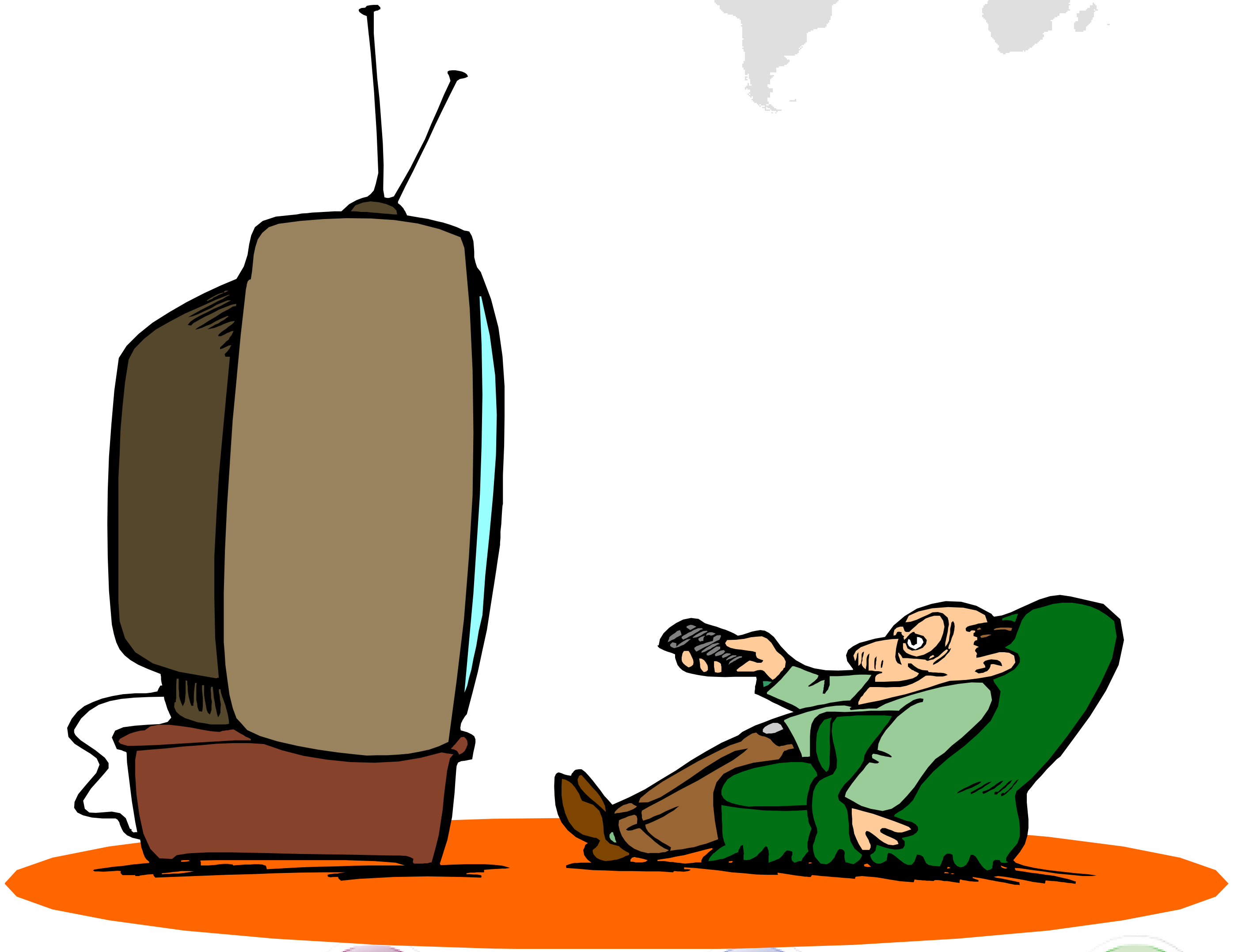
Practical C++ Test-Driven Development with Boost.Test and Bmock

Asher Sterkin, NDS Technologies, Israel

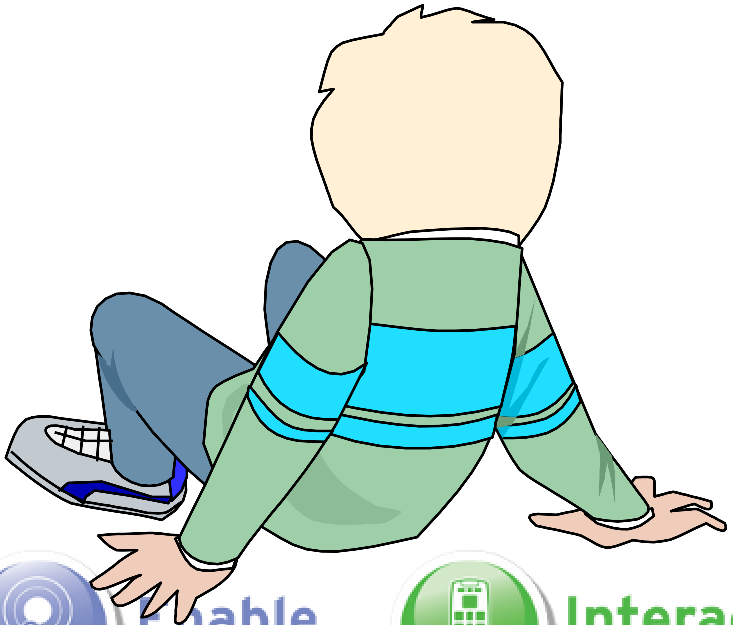
Secure Enable Interact

Let’s Watch TV

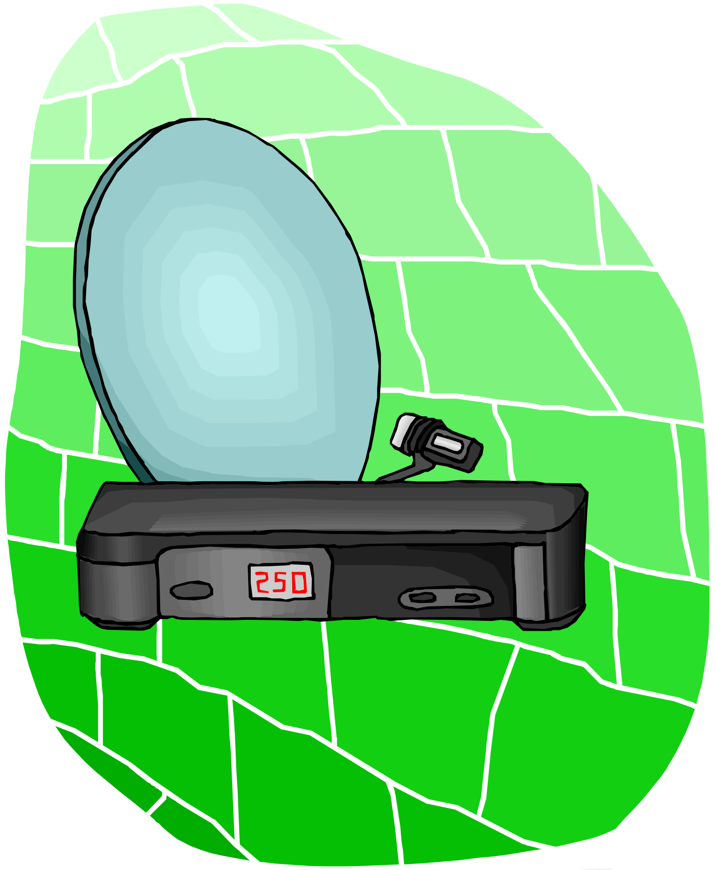


Secure Enable Interact

For Some People TV is Like This



In Reality It's More Like This



Secure Enable Interact

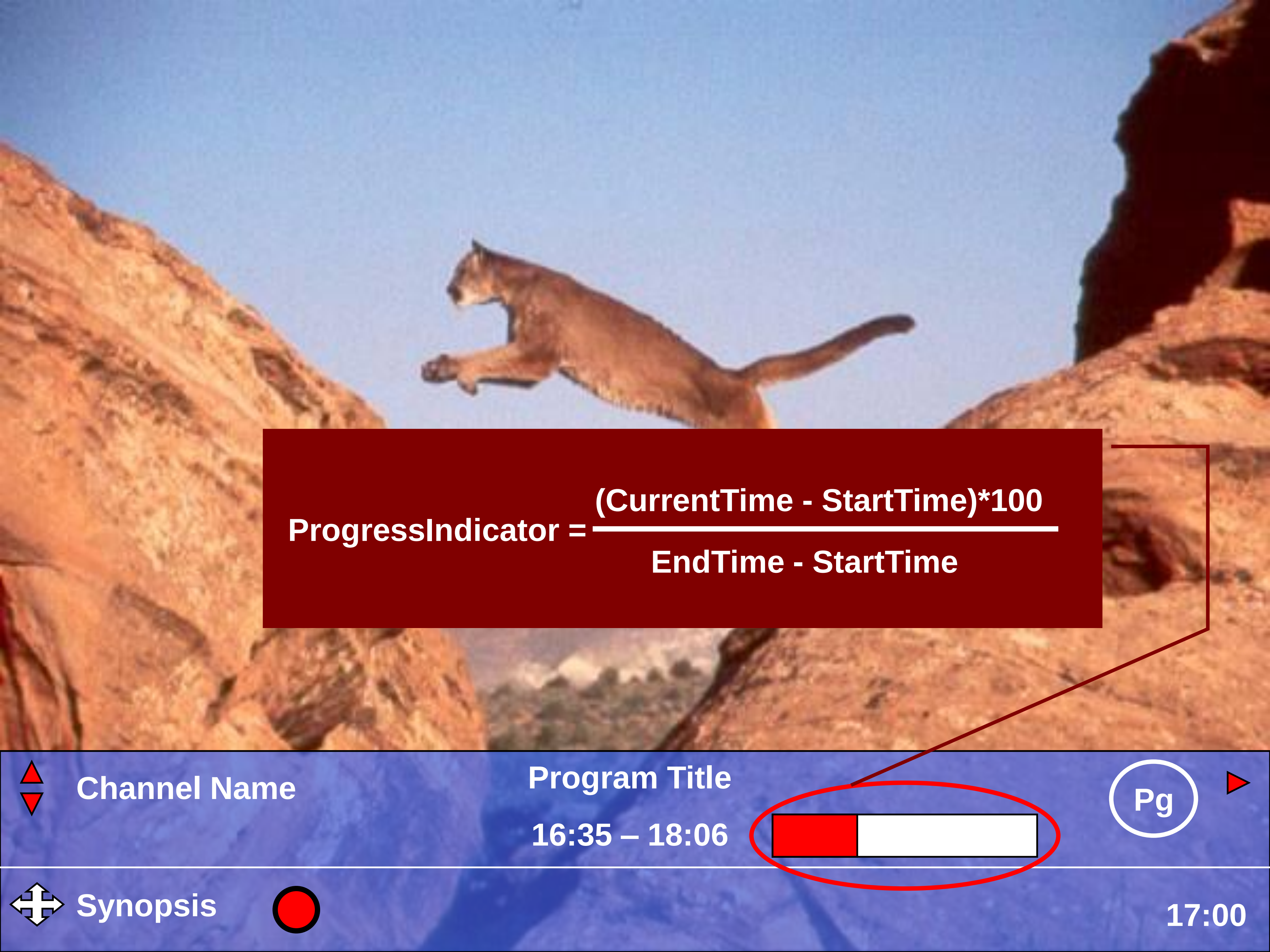
How can we test it?

Agenda

- Testing Strategies
- TDD with Bmock
- Bmock under the hood
- Comparison with Gmock and MockItNow

Secure Enable Interact

Testing Strategies



$$\text{ProgressIndicator} = \frac{(\text{CurrentTime} - \text{StartTime})*100}{\text{EndTime} - \text{StartTime}}$$

How Can We Test It?

Any Ideas?

Testing Strategies

1. Automated acceptance test on PC
2. Automated unit test on PC
3. Manual exploratory test on target platform

Secure Enable Interact

Testing Challenges

Testing Technique	Challenges
Manu	liable,
GUI R	s, low
PC Simulators	Slow, never “exact”, low coverage, uncontrollable environment (clock, communication, …)

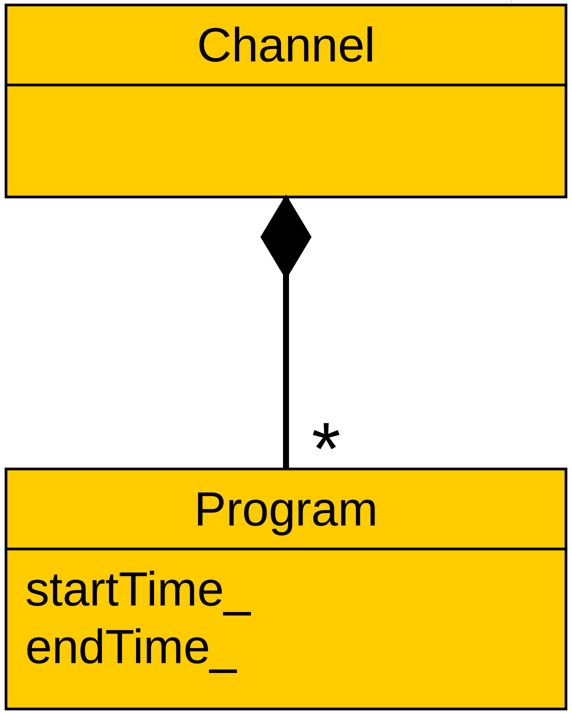
Want to have fully controllable test automation environment with high (close to 100%) coverage

Testing Strategy

Type of Test	Objectives	Automation
Unit ⬅	Modularity, coverage, edge cases at platform level	Fully automated
Acceptance	Definition of done for each feature, edge cases at external interfaces level	Fully automated
Exploratory	To find unanticipated defects and common “sense faults”	MANUAL
System end-to-end	To put multiple components together	Manual at first stage, gradually automated
Stress, endurance	Non-functional requirements	Automated with m.b. manual analysis

Secure Enable Interact

What To Test?



Fixture Class

```
#include <stdafx.h>
#include "Program.h"
using namespace boost::posix_time;

namespace ProgramGuide
{
    static const ptime START_TIME = time_from_string("2009-05-03 19:00");
    static const ptime END_TIME     = time_from_string("2009-05-03 20:00");

    struct program_progress_tester : public Program
    {
        program_progress_tester()
        {
            setStartTime(START_TIME);
            setEndTime(END_TIME);
        }
    };
```

Test Case

```
 static const ptime CURRENT1   = START_TIME;
 static const ptime CURRENT2   = START_TIME + minutes(30);
 static const ptime CURRENT3   = START_TIME + minutes(20);

 BOOST_FIXTURE_TEST_CASE(test_progress, program_progress_tester)
 {
     BOOST_CHECK_EQUAL(0, getProgress(CURRENT1));
     BOOST_CHECK_EQUAL(50, getProgress(CURRENT2));
     BOOST_CHECK_EQUAL(33, getProgress(CURRENT3));
 }
}
```

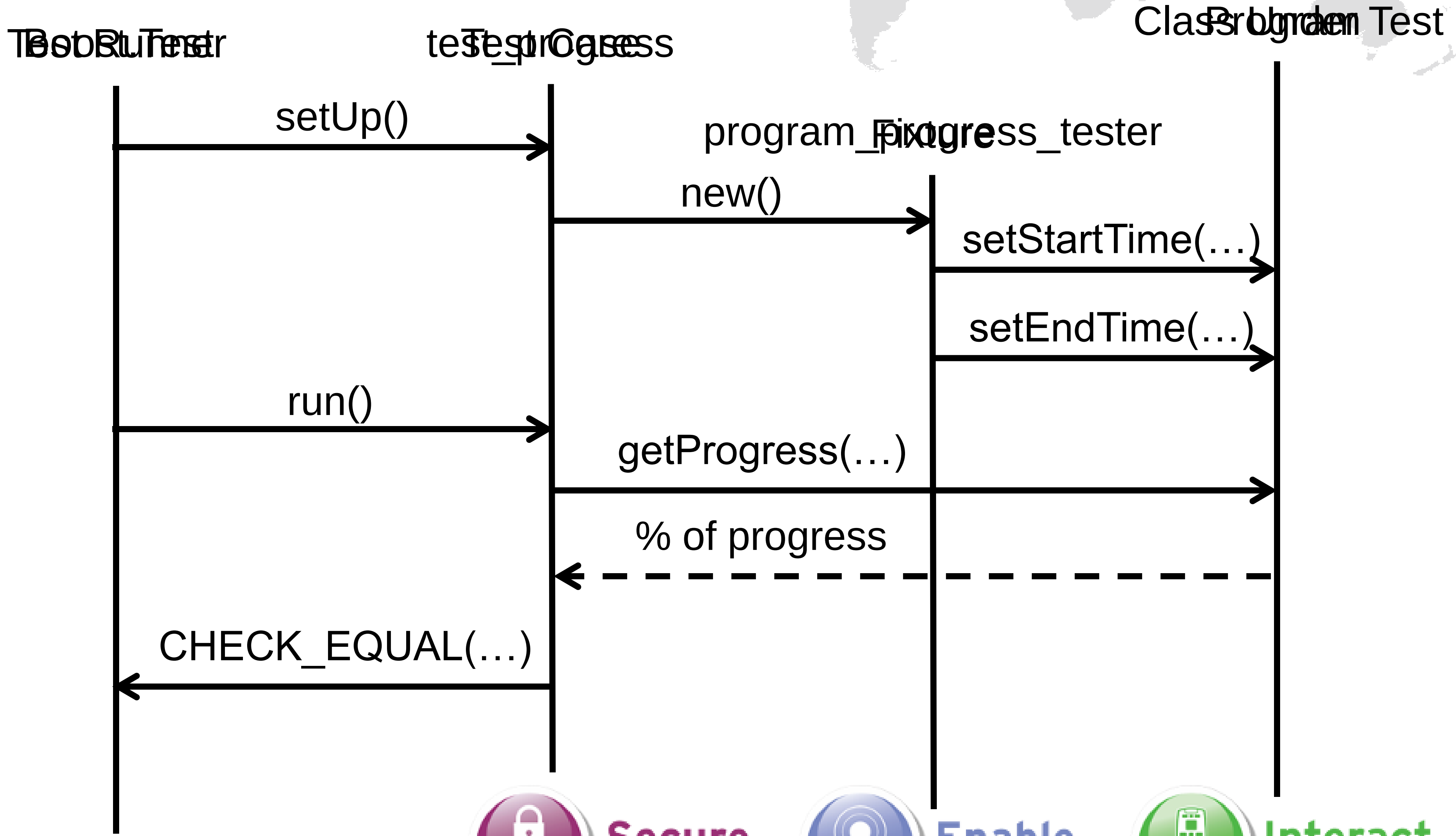
Implementation

```
#pragma once
#include <boost/date_time/posix_time/posix_time.hpp>
namespace ProgramGuide
{
    struct Program
    {
        void setStartTime(const boost::posix_time::ptime &t) { startTime_ = t; }
        void setEndTime(const boost::posix_time::ptime &t)  { endTime_ = t; }
        long getDuration() const
        {
            return (endTime_ - startTime_).total_seconds();
        }
        long getProgress(const boost::posix_time::ptime &t) const
        {
            const long playTime = (t - startTime_).total_seconds();
            const long progress = 100*playTime / getDuration();
            return progress;
        }
        protected:
            boost::posix_time::ptime startTime_;
            boost::posix_time::ptime endTime_;
    };
}
```

Jamroot

```
import testing ;
using msvc ;

BOOST_HOME = C:/Boost ;
BOOST_LIB  = $(BOOST_HOME)/lib ;
BOOST_INC  = $(BOOST_HOME)/include/boost-1_38 ;

project
    : requirements
                <include>$(BOOST_INC)
                <include>inc
                <link>static
                <library-path>$(BOOST_LIB)
                <define>BOOST_TEST_MODULE=epg
              ;

cpp-pch stdafx_test
              : inc/stdafx.h
              ;

unit-test bmock_tutorial
              : [ glob src/*.cpp ]
                [ glob test/*.cpp ]
              ;
```

Unit Testing Machinery



TDD Mantra

- For every branch of every function:
  - Create a simple test, make it so it fails (**red**)

TDD is about bug prevention, rather than bug detection

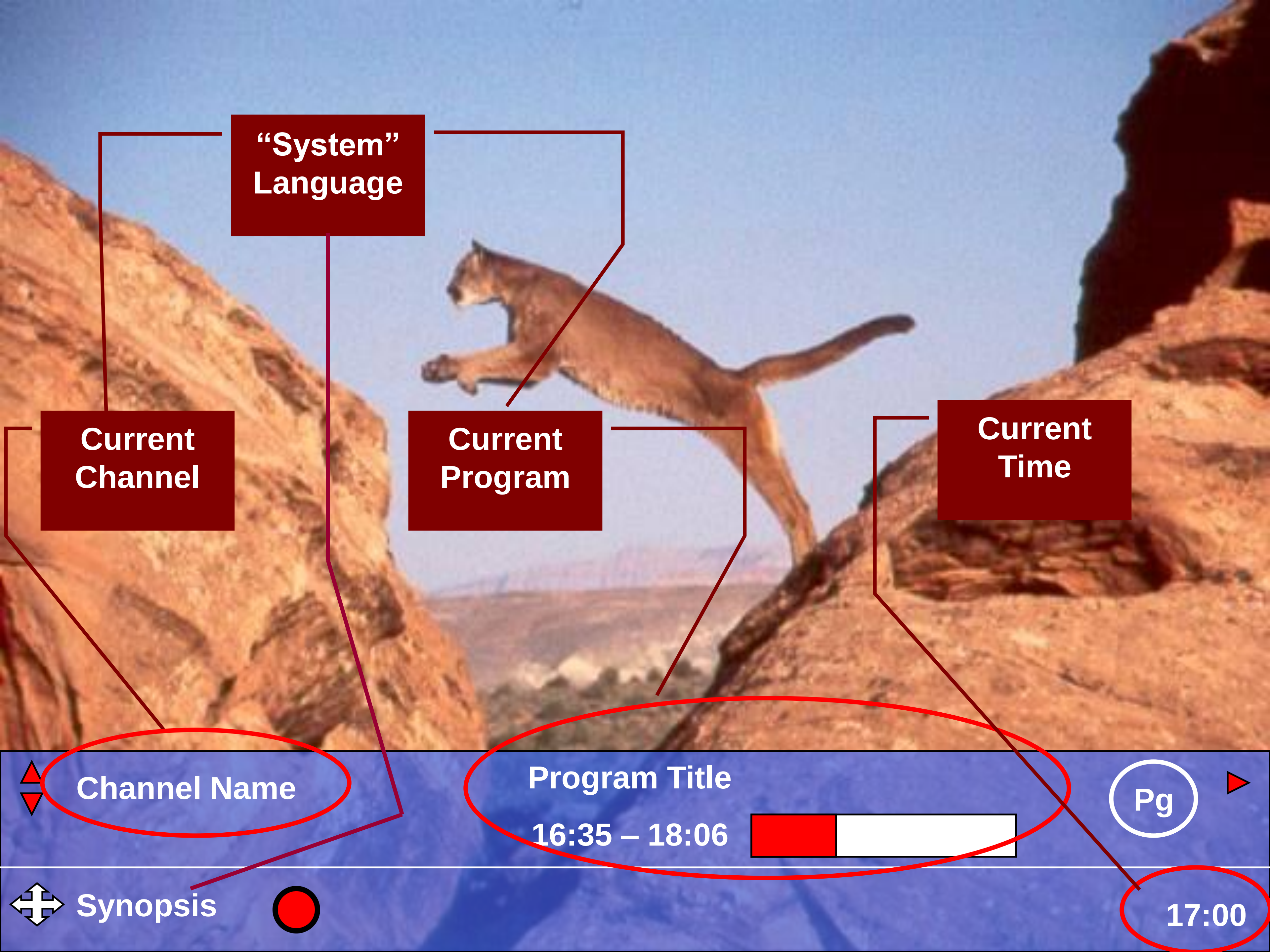
keep all tests running (**re-factor**)

  - When a bug is reported do not fix it until you build a test case, which fails under the same conditions (**maintain test suites**)

Secure Enable Interact

TDD With Bmock

Secure Enable Interact


"System" Language
Current Channel
Current Program
Current Time
Channel Name
Program Title
16:35 – 18:06
Pg
Synopsis
17:00

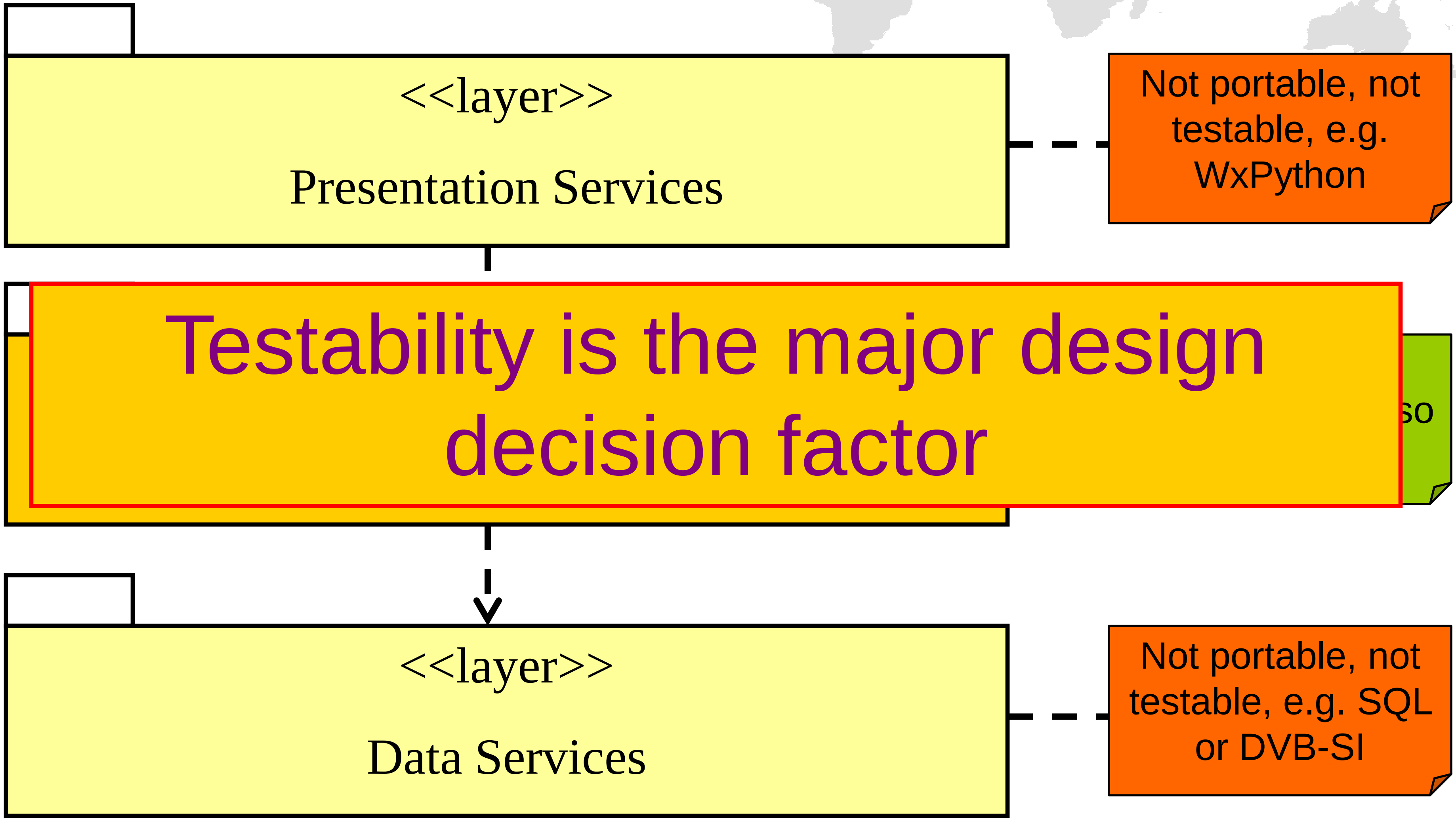
How Can We Test It?

Any Ideas?

Some Design Patterns Would Help

Secure Enable Interact

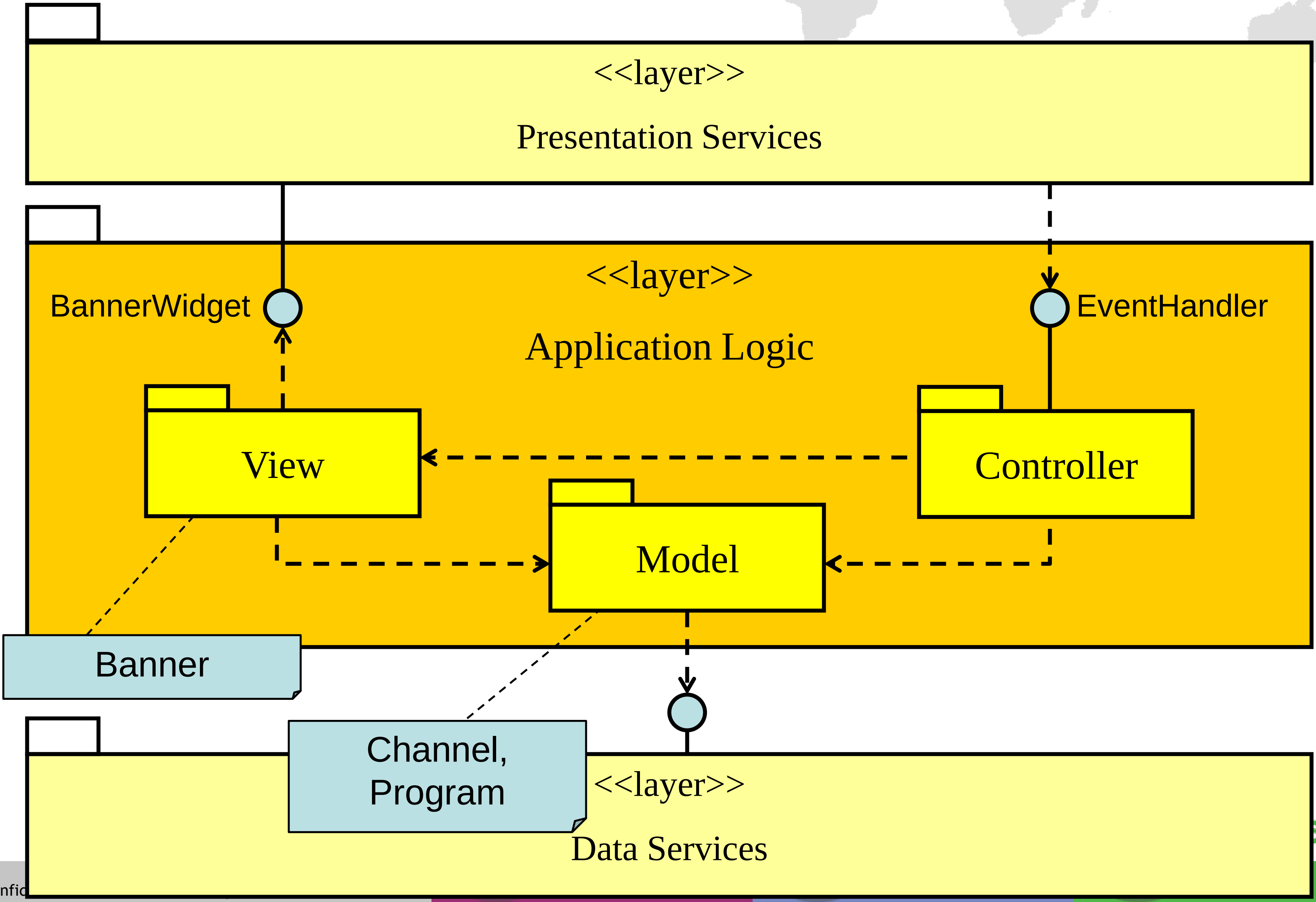
Layers



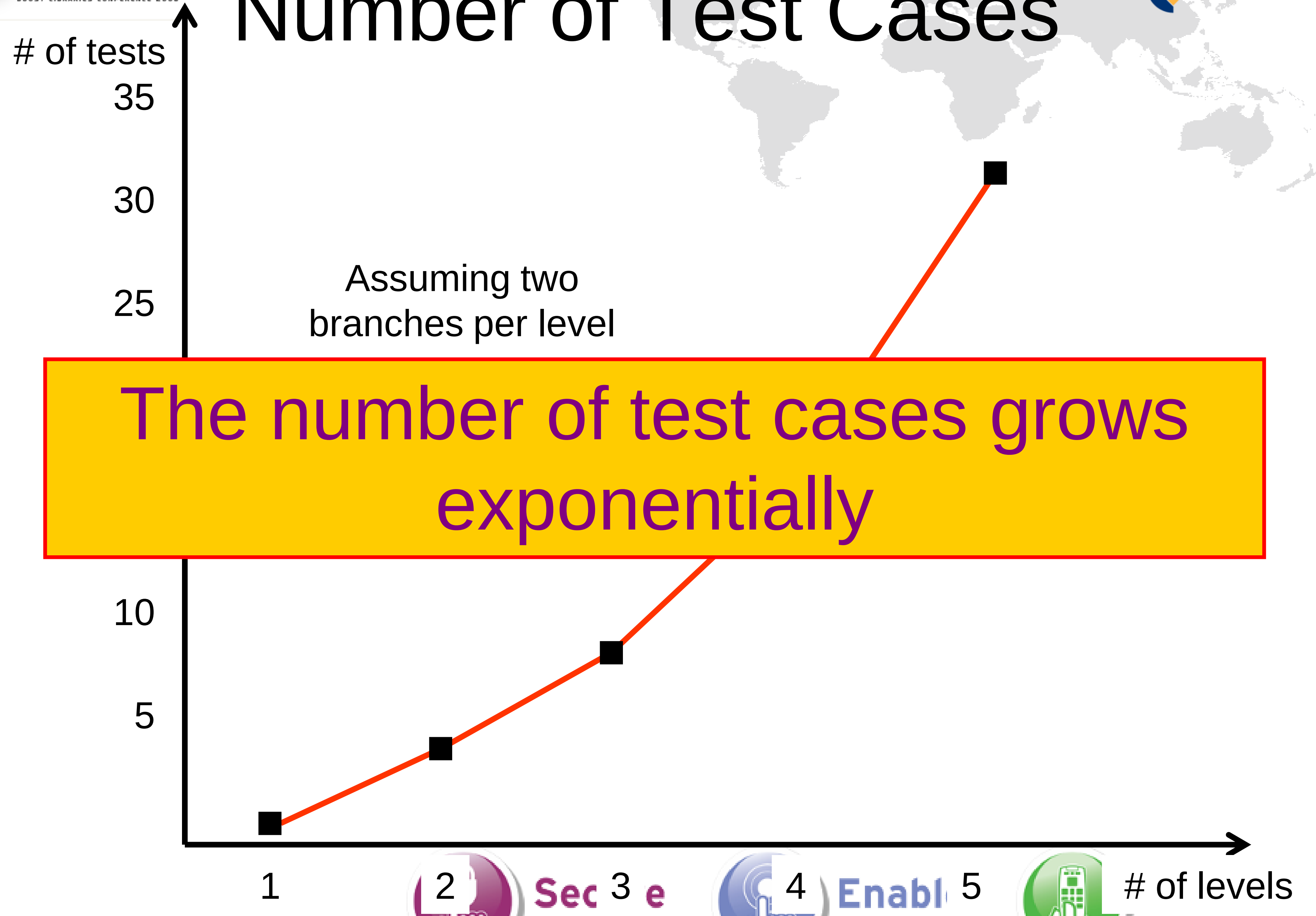
<<layer>>
Presentation Services
Not portable, not testable, e.g. WxPython
so
<<layer>>
Data Services
Not portable, not testable, e.g. SQL or DVB-SI

Testability is the major design decision factor

Model-View-Controller

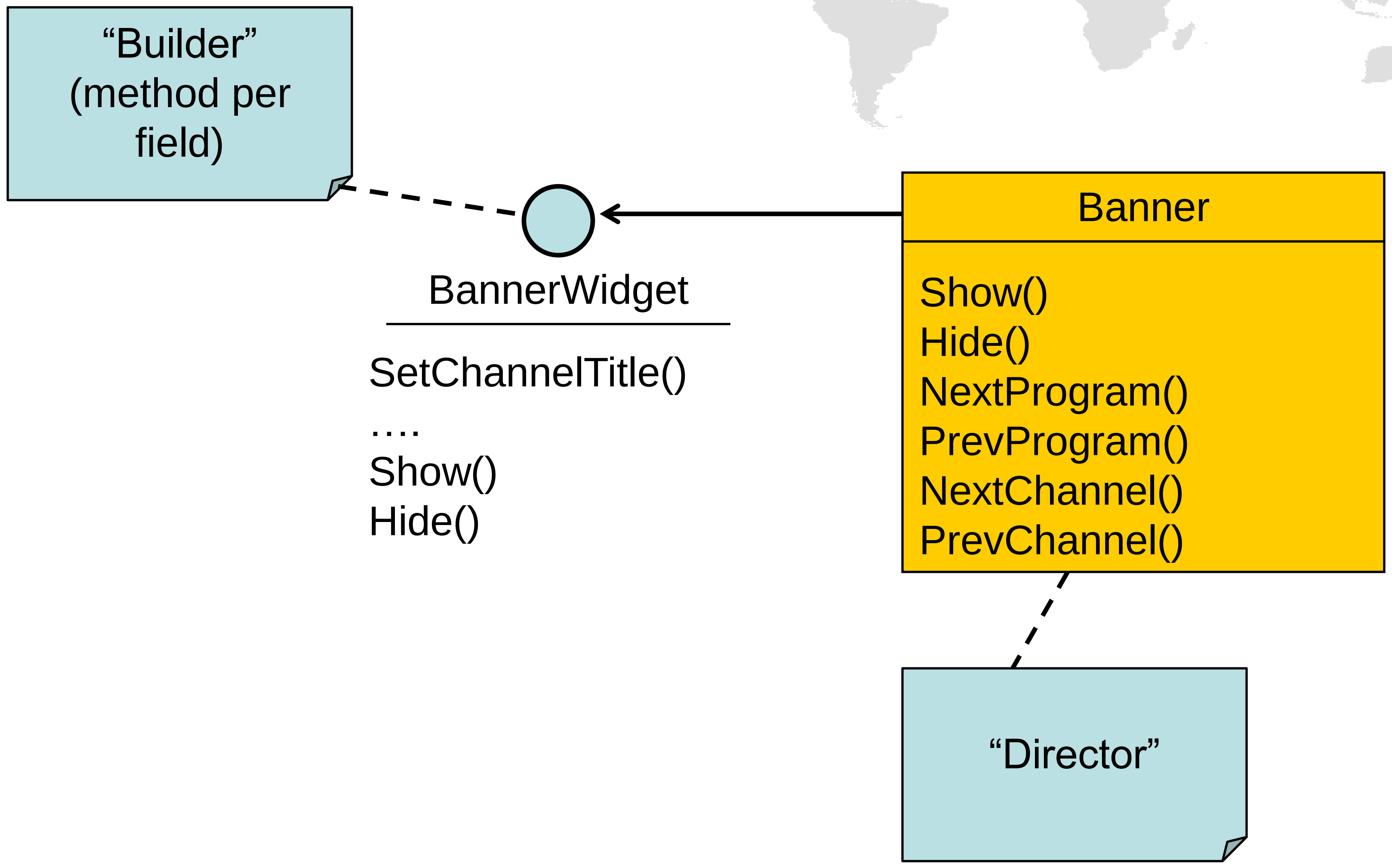


Number of Test Cases



The number of test cases grows exponentially

Builder



“Builder” (method per field)
BannerWidget
SetChannelTitle()
….
Show()
Hide()
Banner
Show()
Hide()
NextProgram()
PrevProgram()
NextChannel()
PrevChannel()
“Director”

Unit Test

```
struct banner_test
{
    banner_test()
      :pWidget_((BannerWidget *)0x01234567)
      ,pChannel_((const Channel *)0x89ABCDEF)
      ,banner_(pWidget_, pChannel_)
    {
        BMOCK_CREATE_METHOD_MOCK(Channel);
        BMOCK_CREATE_METHOD_MOCK(BannerWidget);
    }

    BannerWidget    *pWidget_;
    const Channel   *pChannel_;
    Banner           banner_;
};

BMOCK_TEST(banner_test, test_show)
{
    const char *CHANNEL_TITLE = "Channel 1";

    BMOCK_EXPECT_RETURN(CHANNEL_TITLE, pChannel_ -> GetTitle());
    BMOCK_EXPECT(pWidget_ -> SetChannelTitle(CHANNEL_TITLE));
    BMOCK_EXPECT(pWidget_ -> Show());
    BMOCK_REPLAY;
    banner_.Show();
    BMOCK_VERIFY;
}
```

Mock all methods of the Channel class

From now all mock calls will be treated as expectations

Expect particular mock to be called and specify return value

From now all mock calls will be validated against expectations

Check all expectations have been fulfilled

BannerWidget

```
 struct BannerWidget
 {
    void SetChannelTitle(const char *ch);
    void Show();
 };

 BMOCK_VOID_METHOD(BannerWidget, SetChannelTitle, 1, (IN(const char *,ch)))
 {
    //GUI platform-specific implementation will come here
 }
BMOCK_END

BMOCK_VOID_METHOD(BannerWidget, Show, 0, ())
{
   //GUI platform-specific implementation will come here
}
BMOCK_END
```

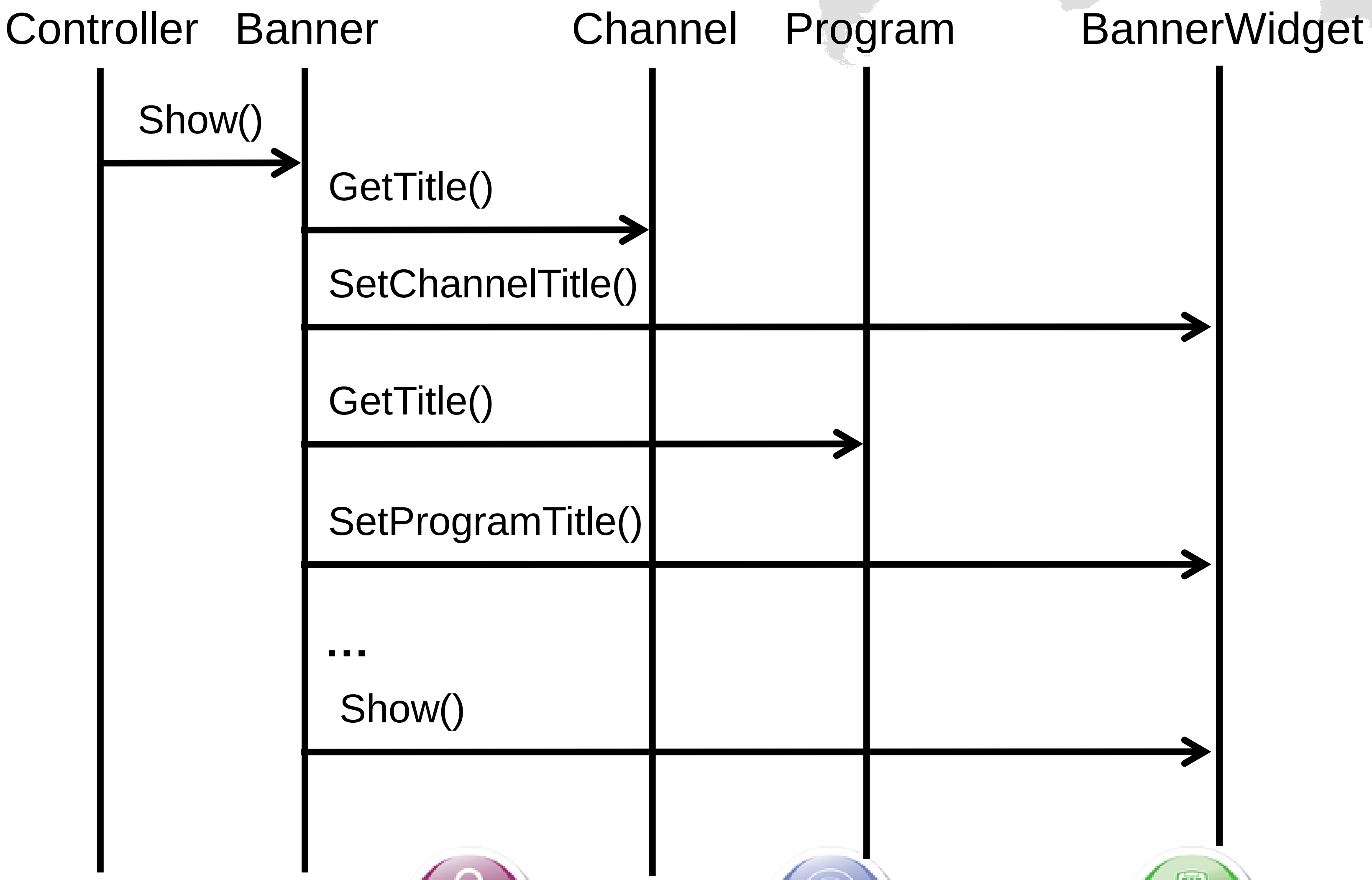
Define a mock

List of arguments

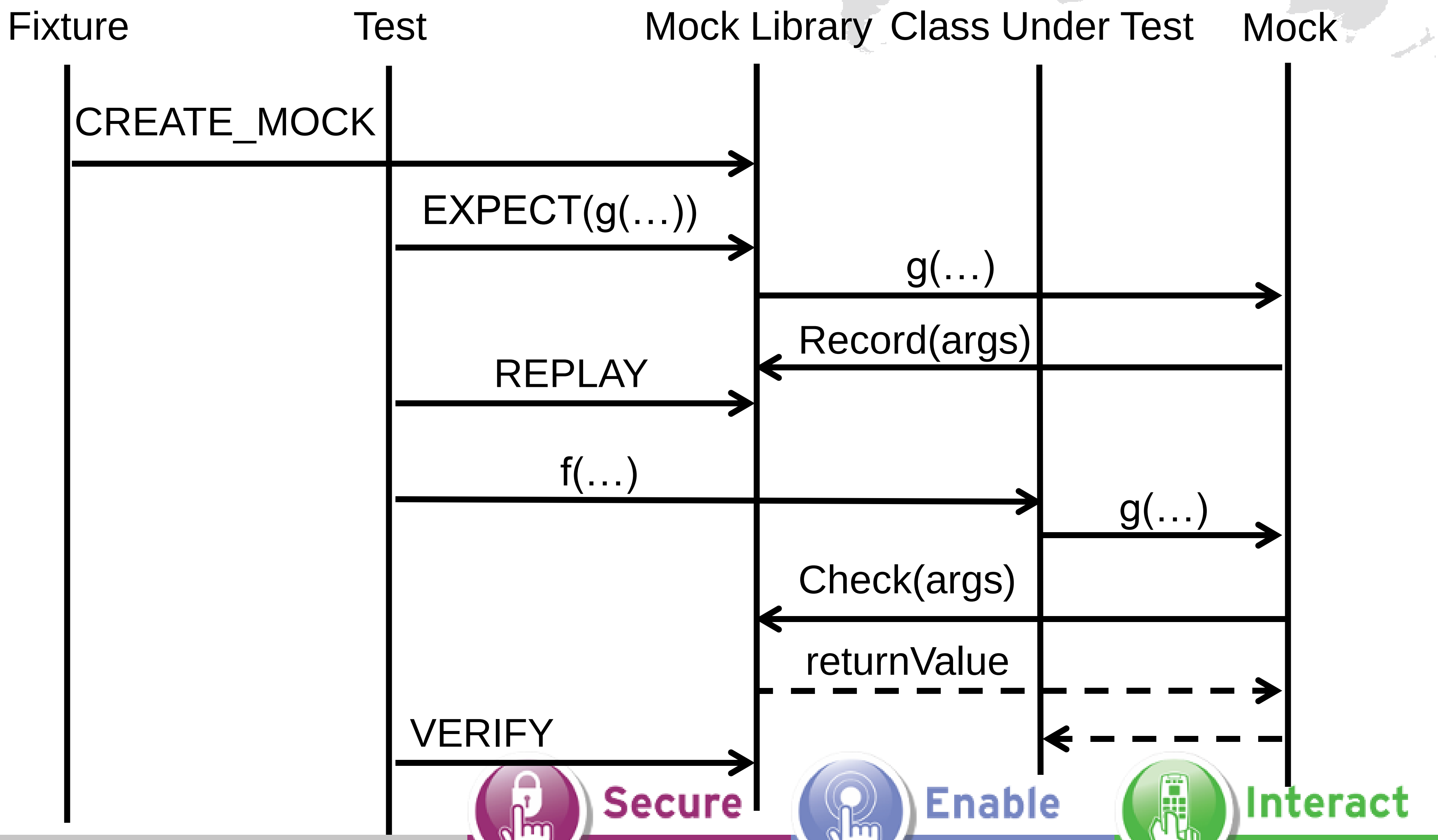
Banner

```
struct Banner
{
     Banner(BannerWidget *w, const Channel *ch)
         :pWidget_(w)
         ,pChannel_(ch)
     {}

     void Show()
     {
          pWidget_ -> SetChannelTitle( pChannel_ -> GetTitle() );
          pWidget_ -> Show();
     }

    BannerWidget  *pWidget_;
    const Channel  *pChannel_;
};
```

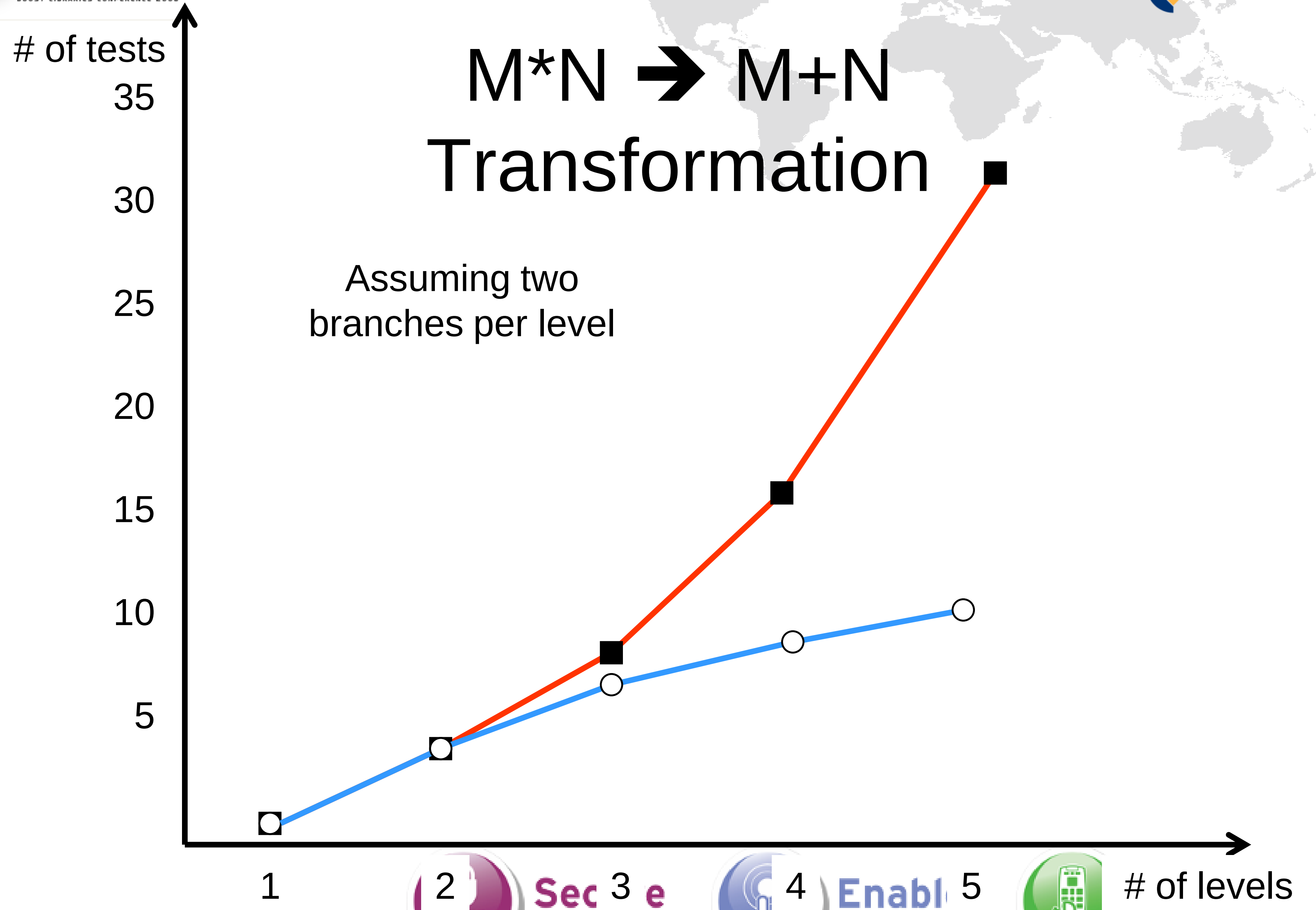
Builder Dynamics



Testing With Mocks



M*N ➔ M+N Transformation



Still Missing Something

- Need acceptance test for definition of done
- Need exploratory test to see how it works

Acceptance Testing with PyFit

Banner Details

fitLib.DoFixture	
start	at.ProgramGuideTester

LocalSettings
DefaultChannel
2

TimeAndDateTable
UTC Time
2009-05-03 19:15

ServiceTable	
Id	Title
1	Movies 1
2	Movies 2
3	Movies 3

EventTable			
Service	StartTime	EndTime	Title
1	2009-05-03 18:15	2009-05-03 20:00	Shut Em Up
1	2009-05-03 20:00	2009-05-03 21:30	Kill Bill
2	2009-05-03 19:00	2009-05-03 22:30	Godfather I
2	2009-05-03 22:30	2009-05-04 00:30	Godfather II
3	2009-05-03 18:45	2009-05-03 20:00	Once Upon a time ...
3	2009-05-03 20:00	2009-05-03 22:30	Matrix

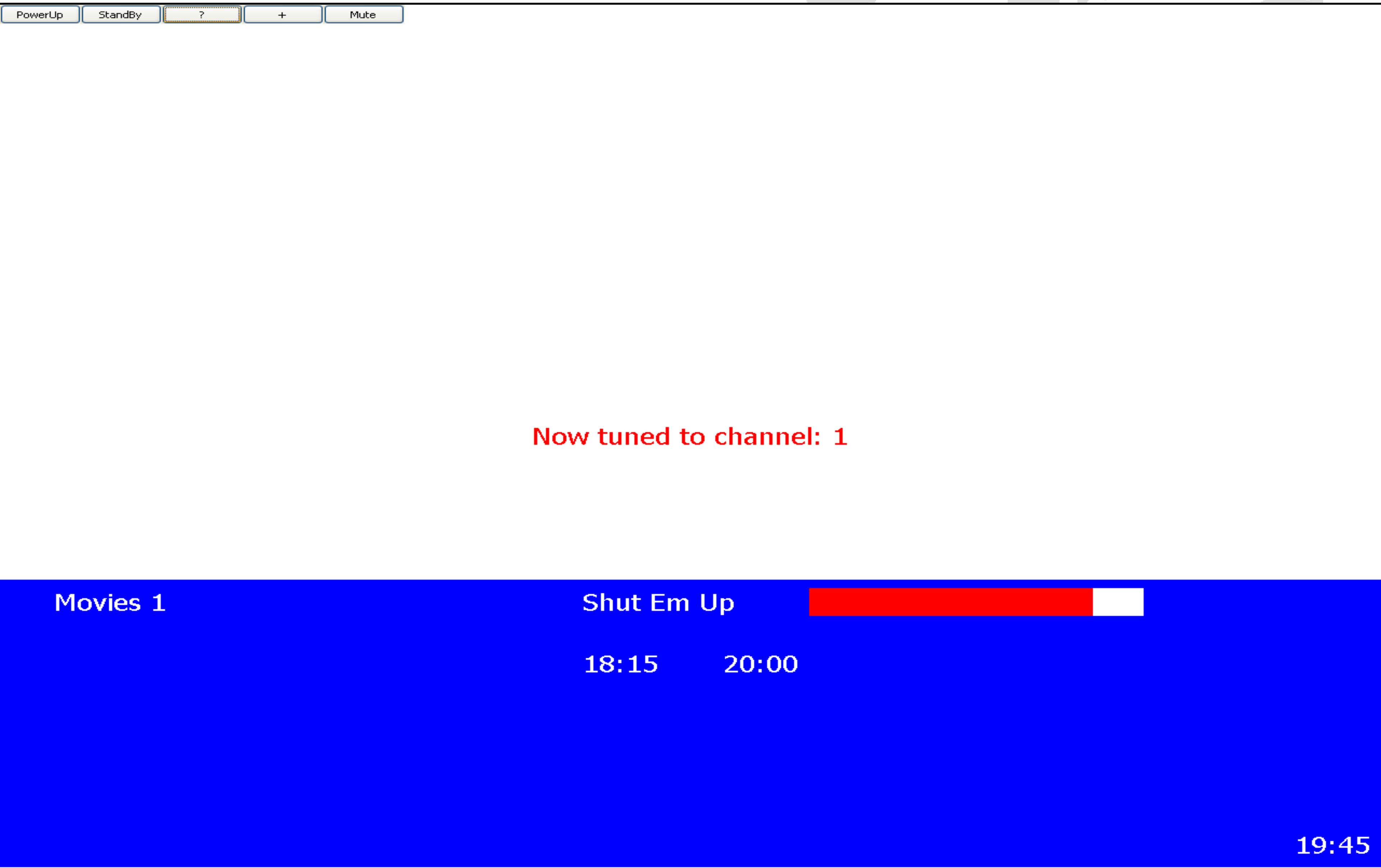
BannerDetails					
Channel	CurrentTime	Program	StartTime	EndTime	Progress
Movies 2	19:15	Godfather I	19:00	22:30	7

power up	
press button	?

BannerDetails					
Channel	CurrentTime	Program	StartTime	EndTime	Progress
Movies 2	19:15	Godfather I	19:00	22:30	7

e Enable Interact

PC Simulator



Now tuned to channel: 1

Movies 1

Shut Em Up

18:15 20:00

19:45

Acceptance Test Driven Development Mantra

- Define Acceptance Test for each user story
- Define interfaces for Controller, View and Data Access

User Story scope (batch size) is the critical success factor

  - View (mock Model and Presentation)
  - Model (mock Data Access)
- Implement mocked interfaces in Fit Fixtures
- Implement mocked interfaces in PC simulator

Secure Enable Interact

Electronic Program Guide User Stories

Any suggestions?

Secure Enable Interact

EPG User Stories

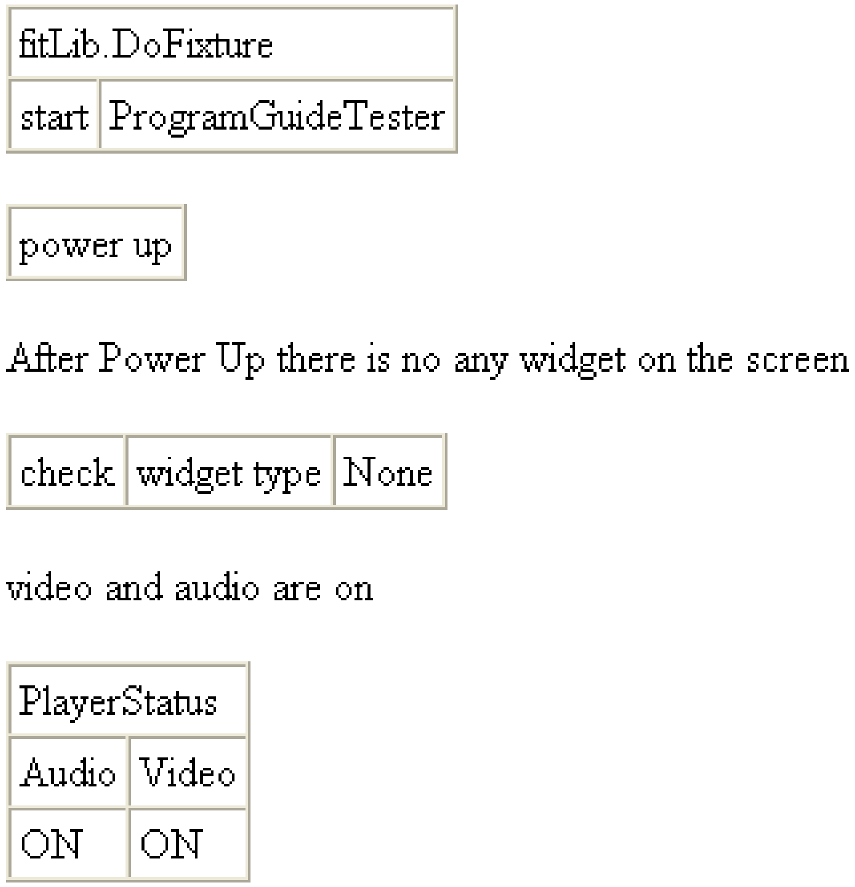
- Power Up:
  - Show Video
  - Tune to Default Channel
- Stand By
- Banner:
  - State Machine
  - Details
- Synopsis
- Mute
- …

Secure Enable Interact

Power Up: Show Video

How to specify acceptance test?

Power Up/Video Acceptance Test



fitLib.DoFixture	
start	ProgramGuideTester

power up

After Power Up there is no any widget on the screen

check	widget type	None

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

PyFIT DoFixture

```
import ProgramGuide as pg
from MediaPlayerStub import MediaPlayerStub

class NoWidget(object):
                def GetType(self):
                    return "None"

class ProgramGuideTester(object):
    _typeDict = {}

    _typeDict["powerUp.types"] = [None]
    def powerUp(self):
        self.currentWidget = NoWidget()
        self.player        = MediaPlayerStub()
        self.guide         = pg.EventHandler(self.player)
        self.guide.StartEvt()

    _typeDict["widgetType.types"] = ["String"]
    def widgetType(self):
        return self.currentWidget.GetType()

    _typeDict["PlayerStatus.types"] = ["$Row"]
    def PlayerStatus(self):
        return ([self.player],  { "Video" : "String",  "Audio" : "String" } )
```

Meta-data for PyFIT

Defines Controller interface

MediaPlayerStub

```
import ProgramGuide as pg

class MediaPlayerStub(pg.MediaPlayer):
        def __init__(self):
           pg.MediaPlayer.__init__(self, self)
           self.Video = "OFF"
           self.Audio = "OFF"

        def SwitchOn(self):
           self.Video = "ON"
           self.Audio = "ON"
```

Defines View interface

Controller

Secure Enable Interact

Unit Test

```
#include <stdafx.h>
#include "Controller/EventHandler.h"

namespace ProgramGuide
{
	namespace Controller
	{
		struct event_handler_tester
		{
			event_handler_tester()
				:player_(0)
				,handler_(&player_)
			{
				BMOCK_CREATE_METHOD_MOCK(View::MediaPlayer);
			}
			MediaPlayer  player_;
			EventHandler handler_;
		};
		BMOCK_TEST(event_handler_tester, test_start)
		{
			BMOCK_EXPECT(player_.SwitchOn());
			BMOCK_REPLAY;
			handler_.StartEvt();
			BMOCK_VERIFY;
		}
	}
}
```

Event Handler

```
#include <stdafx.h>
#include "EventHandler.h"

namespace ProgramGuide
{
	namespace Controller
	{
		EventHandler::EventHandler(MediaPlayer *p)
			:pPlayer_(p)
		{}

		void EventHandler::StartEvt()
		{
			pPlayer_->SwitchOn();
		}
	}
}
```

View

Media Player API

```
#pragma once

namespace ProgramGuide
{
	namespace View
	{
		struct MediaPlayer
		{
			MediaPlayer(HANDLE h)
				:self_(h)
			{}
			void SwitchOn();
		private:
			HANDLE self_;
		};
	}
}
```

Presentation

Boost.Python Wrapper

```
#include <stdafx.h>
#include "Controller/EventHandler.h"
#include "View/MediaPlayer.h"

using namespace ProgramGuide::View;
using namespace ProgramGuide::Controller;
using namespace boost::python;

BOOST_PYTHON_MODULE(ProgramGuide)
{
  class_<EventHandler>("EventHandler", init<MediaPlayer *>())
                .def("StartEvt", &EventHandler::StartEvt)
                ;

  class_<MediaPlayer>("MediaPlayer", init<HANDLE>())
                ;
}
```

MediaPlayer

```
#include <stdafx.h>
#include "View/MediaPlayer.h"

namespace ProgramGuide
{
	//
	//Bmock will generate Python adapters automatically
	//
	BMOCK_VOID_METHOD(View::MediaPlayer,SwitchOn,0,());
}
```

stdafx.h

```
#pragma once
#define WIN32
#define _CONSOLE
#define WIN32_LEAN_AND_MEAN
#define _CRT_SECURE_NO_DEPRECATE
#define _SCL_SECURE_NO_DEPRECATE
#include <vld.h>
#include <boost/test/auto_unit_test.hpp>
//
// If required put other windows header files (e.g. WinSock2.h) here
//
#undef IN
#undef OUT
#define BMOCK_USE_MOCKS //controls how mock code is generated
#define BMOCK_GENERATE_CODE
#include <bmock/bmock.hpp>
#include <boost/python.hpp>
#undef HANDLE //ensures presentation could be changed
#define HANDLE PyObject *
```

Jamroot

```
using msvc ;

BOOST_HOME   = C:/Boost ;
BOOST_LIB    = $(BOOST_HOME)/lib ;
BOOST_INC    = $(BOOST_HOME)/include/boost-1_38 ;

PYTHON_HOME = C:/Python26 ;
PYTHON_LIB   = $(PYTHON_HOME)/libs ;
PYTHON_INC   = $(PYTHON_HOME)/include ;

project
    : requirements
            <include>$(BOOST_INC)
            <library-path>$(BOOST_LIB)
            <include>$(PYTHON_INC)
            <library-path>$(PYTHON_LIB)
          ;

build-project test/unit ;
build-project test/acceptance ;
```

src/Jamfile

```
project
    : requirements
          <include>.
          <include>Presentation
        ;

EPG_MODULES = Presentation Model View Controller ;

import python ;
using python : 2.6 : C:/Python26 ;

cpp-pch stdafx_py
        : Presentation/stdafx.h
        ;

python-extension ProgramGuide
        : [ glob $(EPG_MODULES)/*.cpp ]
          stdafx_py
        ;

install install-pg
        : ProgramGuide
        : <location>../scripts
        ;
```

test/unit/Jamfile

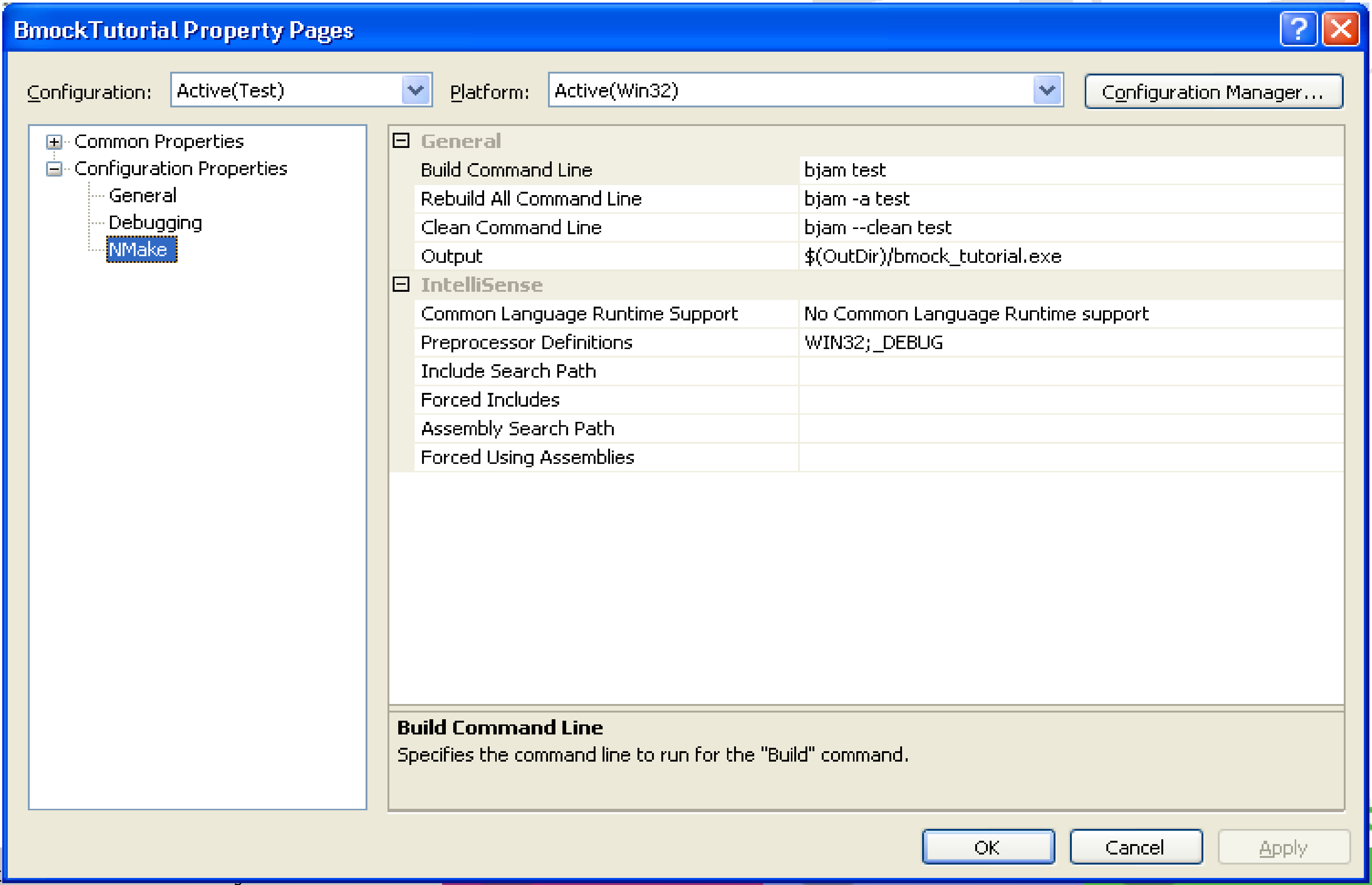
```
project
    : requirements
                <define>BOOST_TEST_MODULE=epg
                <include>.
                <include>../../src
              ;

EPG_MODULES = Model View Controller Presentation ;

import testing ;

cpp-pch stdafx_test
              : stdafx.h
              ;

unit-test bmock_tutorial
              : [ glob ../../src/$(EPG_MODULES)/*.cpp ]
                [ glob $(EPG_MODULES)/*.cpp ]
                stdafx_test
              ;
```

Gui Simulator

```
import wx
import ProgramGuide as pg
import wx.lib.buttons  as  buttons

class MediaPlayerStub(pg.MediaPlayer):
        def __init__(self, panel):
                pg.MediaPlayer.__init__(self, self)
                self.screen = panel

        def SwitchOn(self):
                self.screen.SetBackgroundColour(wx.WHITE)
                self.screen.Refresh()

class ProgramGuidePanel(wx.Panel):
        def __init__(self, parent):
                wx.Panel.__init__(self, parent, -1)
                self.SetBackgroundColour(wx.BLUE)
                self.SetAutoLayout(True)
                b1 = wx.Button(self, -1, "PowerUp", (5,5))
                self.Bind(wx.EVT_BUTTON, self.OnPowerUp, b1)

        def OnPowerUp(self, btn):
                self.player = MediaPlayerStub(self)
                self.guide  = pg.EventHandler(self.player)
                self.guide.StartEvt()

….
```

Controller interface

Visual Studio Configurations



Configuration: Active(Test)   Platform: Active(Win32)

General

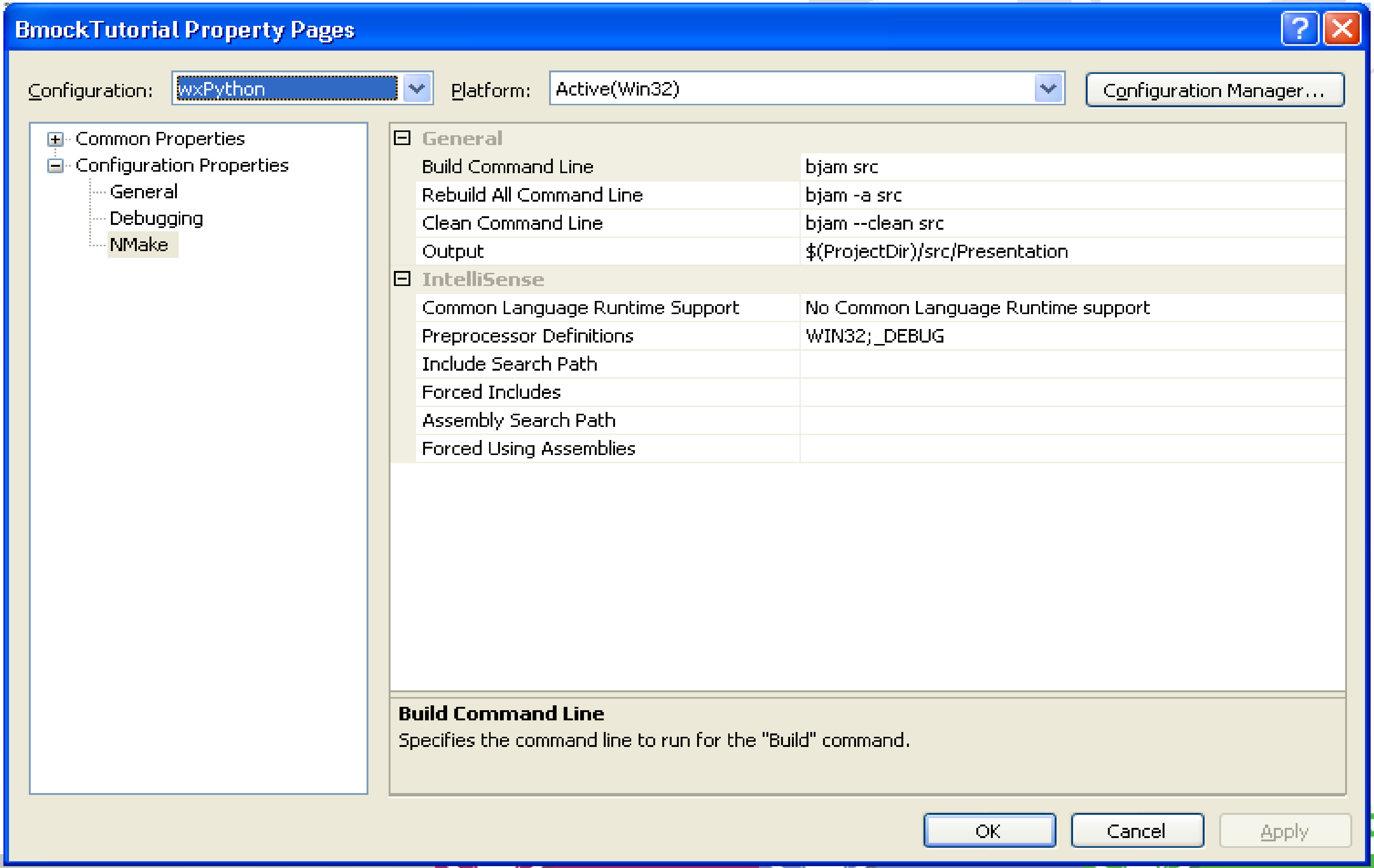
Build Command Line	bjam test
Rebuild All Command Line	bjam -a test
Clean Command Line	bjam --clean test
Output	$(OutDir)/bmock_tutorial.exe

IntelliSense

Common Language Runtime Support	No Common Language Runtime support
Preprocessor Definitions	WIN32;_DEBUG
Include Search Path	
Forced Includes	
Assembly Search Path	
Forced Using Assemblies	

Build Command Line
Specifies the command line to run for the "Build" command.

Visual Studio Configurations



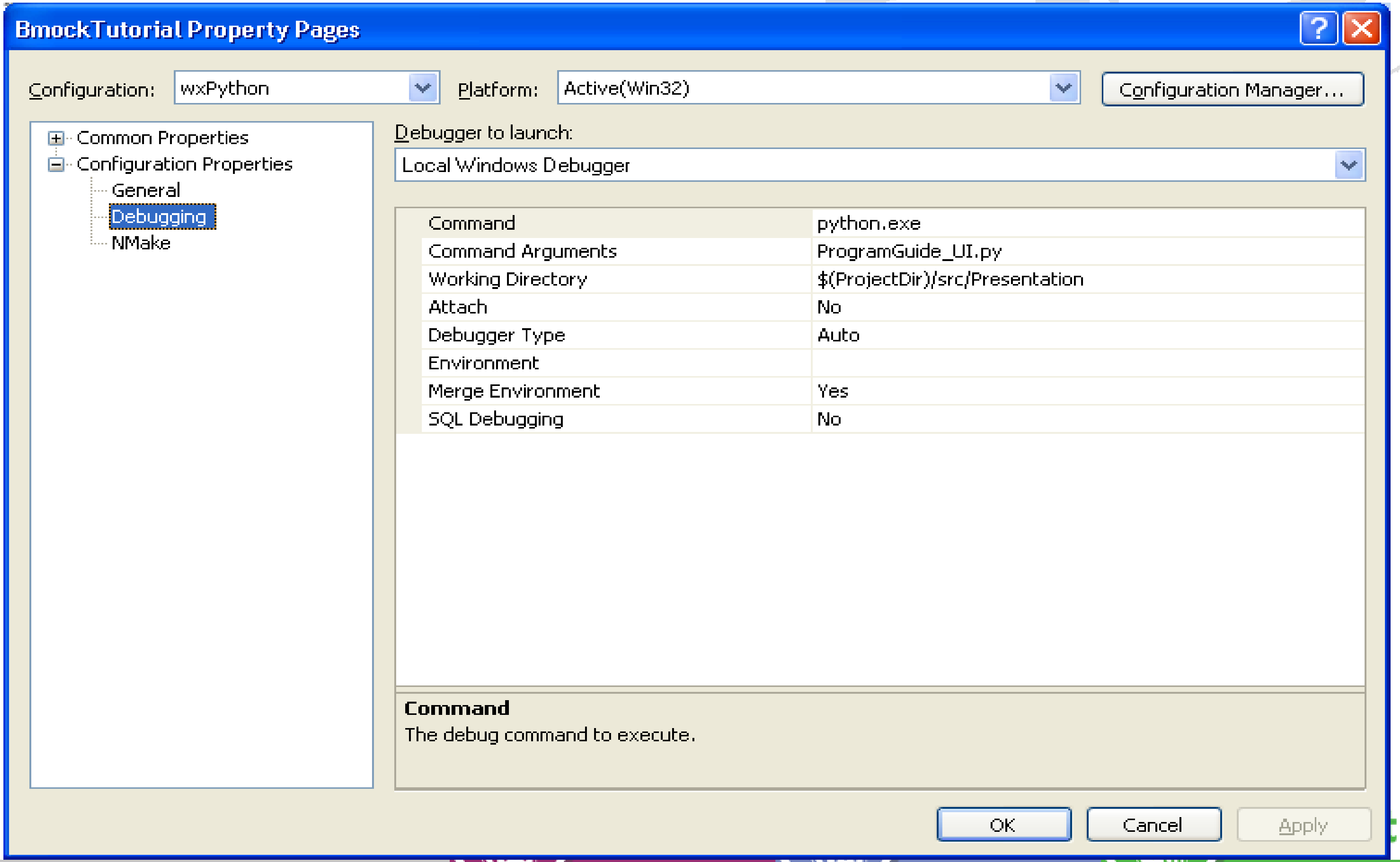
BmockTutorial Property Pages

Configuration: wxPython   Platform: Active(Win32)   Configuration Manager...

General	
Build Command Line	bjam src
Rebuild All Command Line	bjam -a src
Clean Command Line	bjam --clean src
Output	$(ProjectDir)/src/Presentation
IntelliSense	
Common Language Runtime Support	No Common Language Runtime support
Preprocessor Definitions	WIN32;_DEBUG
Include Search Path	
Forced Includes	
Assembly Search Path	
Forced Using Assemblies	

Build Command Line
Specifies the command line to run for the "Build" command.

Visual Studio Configurations



Configuration: wxPython

Platform: Active(Win32)

Debugger to launch: Local Windows Debugger

Command	python.exe
Command Arguments	ProgramGuide_UI.py
Working Directory	$(ProjectDir)/src/Presentation
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No

Command
The debug command to execute.

Your first user story

Almost like first love …

When Developing 1st User Story

- Decide about 3rd party libraries:
  - Boost
  - BMock
  - wxPython
  - PyFIT
- Compiler and IDE (VS 2008)
- Build System (bjam)
- Version Control (SVN)
- Continuous Integration (Hudson)
- Project layout
- Initial domain model (Media Player)

Power Up: Tune

What would be its acceptance test?

Secure Enable Interact

Power Up: Tune

fitLib.DoFixture	
start	ProgramGuideTester

Assuming Channel #2 set up as a default in non-volotile memory

LocalSettings
DefaultChannel
2

power up

After Power Up there is no any widget on the screen

check	widget type	None

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

the box is tuned to the default channel

check	current channel	2

LocalSettingsFixture

```
from fitLib.SetUpFixture import SetUpFixture
import ProgramGuide as pg

class LocalSettingsFixture(SetUpFixture, pg.LocalSettings):
    _typeDict = {}
    def __init__(self):
        pg.LocalSettings.__init__(self, self)
        self.defaultChannel = -1

    _typeDict["DefaultChannel.types"] = [None, "Int"]
    def DefaultChannel(self, ch):
        self.defaultChannel = ch

    def GetDefaultChannel(self):
        return self.defaultChannel
```

Defines Data Access interface

Controller

Unit Test

```
#include <stdafx.h>
#include "Controller/EventHandler.h"

namespace ProgramGuide
{
	namespace Controller
	{
		struct event_handler_tester
		{
			event_handler_tester()
				:handler_(&view_, &data_)
			{
				BMOCK_CREATE_METHOD_MOCK(View::MediaPlayer);
				BMOCK_CREATE_METHOD_MOCK(DataAccess::*);
			}
			View::Factory	view_;
			DataAccess::Factory	data_;
			EventHandler			handler_;
		};
```

Unit Test

```
        BMOCK_TEST(event_handler_tester, test_start)
        {
            const int CHANNEL = 2;
            View::MediaPlayer                   player;
            const DataAccess::LocalSettings     settings;
            DataAccess::Tuner                   tuner;

            BMOCK_EXPECT_RETURN(&settings, data_.GetLocalSettings());
            BMOCK_EXPECT_RETURN(CHANNEL, settings.GetDefaultChannel());
            BMOCK_EXPECT_RETURN(&tuner, data_.GetTuner());
            BMOCK_EXPECT(tuner.TuneTo(CHANNEL));
            BMOCK_EXPECT_RETURN(&player, view_.GetPlayer());
            BMOCK_EXPECT(player.SwitchOn());
            BMOCK_REPLAY;
            handler_.StartEvt();
            BMOCK_VERIFY;
        }
    }
}
```

Event Handler

```
#include <stdafx.h>
#include "EventHandler.h"

namespace ProgramGuide
{
	namespace Controller
	{
		EventHandler::EventHandler(View::Factory *v, DataAccess::Factory *d)
			:pView_(v)
			,pData_(d)
		{}

		void EventHandler::StartEvt()
		{
			const int ch = pData_->GetLocalSettings()->GetDefaultChannel();
			pData_->GetTuner()->TuneTo(ch);
			pPlayer_->SwitchOn();
		}
	}
}
```

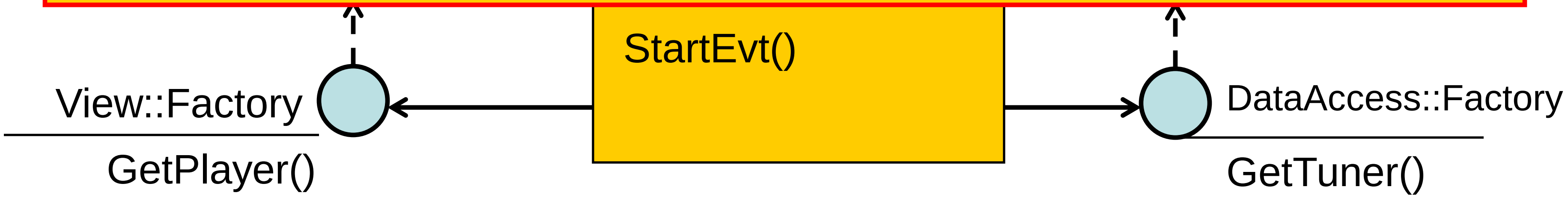
View

Factory

```
#pragma once
#include "MediaPlayer.h"

namespace ProgramGuide
{
	namespace View
	{
		struct Factory
		{
			Factory(HANDLE h=0)
				:self_(h)
			{}
			MediaPlayer *GetPlayer();
		private:
			HANDLE self_;
		};
	}
}
```

Factory

Avoid Direct Object Creation, Use Dependency Injection and Factories



Data Access

Secure Enable Interact

Factory

```
#pragma once
#include "LocalSettings.h"
#include "Tuner.h"

namespace ProgramGuide
{
	namespace DataAccess
	{
		struct Factory
		{
							Factory(HANDLE h=0)
									:self_(h)
							{}
			const LocalSettings	*GetLocalSettings() const;
			Tuner *				GetTuner();
		private:
			HANDLE				self_;
		};
	}
}
```

Presentation

Boost.Python Wrapper

```
#include <stdafx.h>
#include "Controller/EventHandler.h"
#include "View/Factory.h"
#include "DataAccess/Factory.h"

using namespace ProgramGuide;
using namespace ProgramGuide::Controller;
using namespace boost::python;

BOOST_PYTHON_MODULE(ProgramGuide)
{
                class_<EventHandler>("EventHandler", init<View::Factory *,DataAccess::Factory *>())
                  .def("StartEvt", &EventHandler::StartEvt)
                  ;

  class_<View::MediaPlayer>("MediaPlayer", init<HANDLE>())
                  ;
  class_<View::Factory>("ViewFactory", init<HANDLE>())
                  ;

  class_<DataAccess::Factory>("DataFactory", init<HANDLE>())
                  ;
  class_<DataAccess::LocalSettings>("LocalSettings", init<HANDLE>())
                  ;
  class_<DataAccess::Tuner>("Tuner", init<HANDLE>())
                  ;
}
```

Python Adapters

```
namespace ProgramGuide
{
	BMOCK_VOID_METHOD(DataAccess::Tuner,TuneTo,1,(IN(int,ch)))

	BMOCK_CONST_METHOD(int, DataAccess::LocalSettings,GetDefaultChannel,0,())

	BMOCK_CONST_METHOD(const DataAccess::LocalSettings *,DataAccess::Factory,
	                                        GetLocalSettings, 0, ())

	BMOCK_METHOD(DataAccess::Tuner *,DataAccess::Factory, GetTuner, 0, ())

	BMOCK_VOID_METHOD(View::MediaPlayer,SwitchOn,0,())

	BMOCK_METHOD(View::MediaPlayer *,View::Factory, GetPlayer, 0, ())
}
```

Stand By

How to specify acceptance test?

Secure Enable Interact

Stand By

fitLib.DoFixture	
start	ProgramGuideTester

power up

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

stand by

video and audio are off

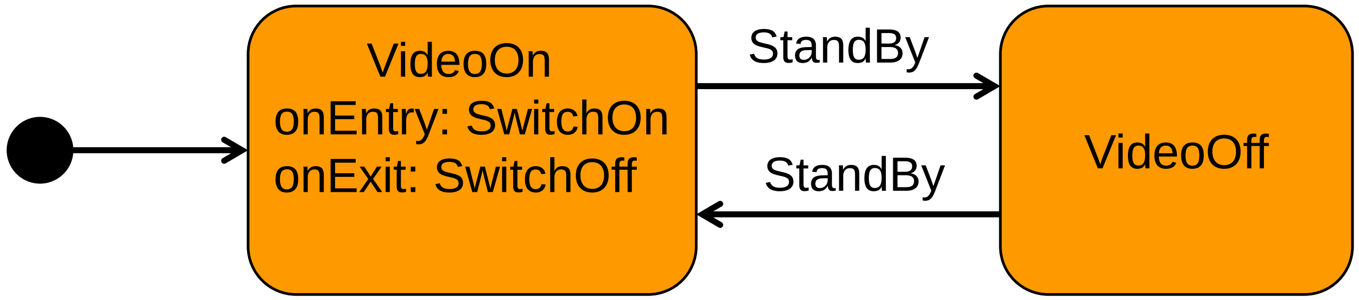
PlayerStatus	
Audio	Video
OFF	OFF

stand by

video and audio are on

PlayerStatus	
Audio	Video
ON	ON

State-dependent Behavior



Controller

Secure Enable Interact

Unit Test

```
#include <stdafx.h>
#include "Controller/StateMachine.h"

namespace ProgramGuide
{
	namespace Controller
	{
		struct state_machine_tester
		{
			static const int CHANNEL = 2;
			state_machine_tester()
				:sm_(&view_, &data_)
			{
				BMOCK_CREATE_METHOD_MOCK(DataAccess::*);
				BMOCK_CREATE_METHOD_MOCK(View::*);
			}

			View::Factory				view_;
			DataAccess::Factory			data_;
			View::MediaPlayer			player_;
			const DataAccess::LocalSettings	settings_;
			DataAccess::Tuner			tuner_;
			StateMachine				sm_;
		};
```

Unit Test

```
BMOCK_TEST(state_machine_tester, test_start_and_stand_by)
{
        BMOCK_CREATE_METHOD_MOCK(Controller::StateMachine::TuneToDefault);
        BMOCK_CREATE_METHOD_MOCK(Controller::StateMachine::VideoOn);
        BMOCK_CREATE_METHOD_MOCK(Controller::StateMachine::VideoOff);
        BMOCK_EXPECT(sm_.TuneToDefault());
        BMOCK_EXPECT(sm_.VideoOn());
        BMOCK_EXPECT(sm_.VideoOff());
```

Mocks Facilitate Programming by Intention

```
}

BMOCK_TEST(state_machine_tester, test_tune_to_default)
{
        BMOCK_EXPECT_RETURN(&settings_, data_.GetLocalSettings());
        BMOCK_EXPECT_RETURN(CHANNEL, settings_.GetDefaultChannel());
        BMOCK_EXPECT_RETURN(&tuner_, data_.GetTuner());
        BMOCK_EXPECT(tuner_.TuneTo(CHANNEL));
        BMOCK_REPLAY;
        sm_.TuneToDefault();
        BMOCK_VERIFY;
}
```

Unit Test

```
        BMOCK_TEST(state_machine_tester, test_video_on)
        {
            BMOCK_EXPECT_RETURN(&player_, view_.GetPlayer());
            BMOCK_EXPECT(player_.SwitchOn());
            BMOCK_REPLAY;
            sm_.VideoOn();
            BMOCK_VERIFY;
        }

        BMOCK_TEST(state_machine_tester, test_video_off)
        {
            BMOCK_EXPECT_RETURN(&player_, view_.GetPlayer());
            BMOCK_EXPECT(player_.SwitchOff());
            BMOCK_REPLAY;
            sm_.VideoOff();
            BMOCK_VERIFY;
        }
    }
}
```

StateMachine.h

```
#pragma once

namespace sc = boost::statechart;
namespace ProgramGuide
{
	namespace View { struct Factory; }
	namespace DataAccess { struct Factory; }
	namespace Controller
	{
		struct VideoOn;
		struct VideoOff;
		struct EvStandBy  : sc::event< EvStandBy > {};

		struct StateMachine : sc::state_machine< StateMachine, VideoOn >
		{
							StateMachine(View::Factory *v, DataAccess::Factory *d)
								:pView_(v)
								,pData_(d)
							{}
							void Start();
							void TuneToDefault();
							void VideoOn();
							void VideoOff();
			View::Factory		*pView_;
			DataAccess::Factory	*pData_;
		};
```

StateMachine.h

```
		struct VideoOn : sc::state< VideoOn, StateMachine >
		{
			VideoOn(my_context ctx)
				:sc::state<VideoOn, StateMachine>(ctx)
			{
				context< StateMachine >().VideoOn();
			}
			~VideoOn()
			{
				context< StateMachine >().VideoOff();
			}

			typedef sc::transition< EvStandBy, VideoOff > reactions;
		};

		struct VideoOff : sc::simple_state< VideoOff, StateMachine >
		{
			typedef sc::transition< EvStandBy, VideoOn > reactions;
		};
	}
}
```

StateMachine.cpp

```
#include <stdafx.h>
#include "StateMachine.h"
namespace ProgramGuide
{
	BMOCK_VOID_METHOD(Controller::StateMachine, Start,0,())
	{
		TuneToDefault();
		initiate();
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, TuneToDefault,0,())
	{
		const int ch = pData_->GetLocalSettings()->GetDefaultChannel();
		pData_->GetTuner()->TuneTo(ch);
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, VideoOn, 0, ())
	{
		pView_->GetPlayer()->SwitchOn();
	}
	BMOCK_END
	BMOCK_VOID_METHOD(Controller::StateMachine, VideoOff, 0, ())
	{
		pView_->GetPlayer()->SwitchOff();
	}
	BMOCK_END
}
```

Presentation

Secure Enable Interact

Event Handler

```
#include <stdafx.h>
#include "EventHandler.h"

namespace ProgramGuide
{
        using namespace Controller;

        EventHandler::EventHandler(View::Factory *v, DataAccess::Factory *d)
                :pStateMachine_(new StateMachine(v, d))
        {}

        void EventHandler::StartEvt()
        {
                pStateMachine_->Start();
        }

        void EventHandler::StandByEvt()
        {
                pStateMachine_->process_event( EvStandBy() );
        }

        void EventHandler::ShutDown()
        {
                pStateMachine_->terminate();
        }
}
```

Banner: State Machine

How about acceptance test?

Banner: State Machine

fitLib.DoFixture	
start	ProgramGuideTester

LocalSettings
IdleTimeout
15

power up		
check	widget type	None

The "?" (help) button brings Banner widget up

help		
check	widget type	Banner

The next "?" brings Banner widget down

help		
check	widget type	None

If no button is pressed the banner disapears after pre-defined timeout (in seconds)

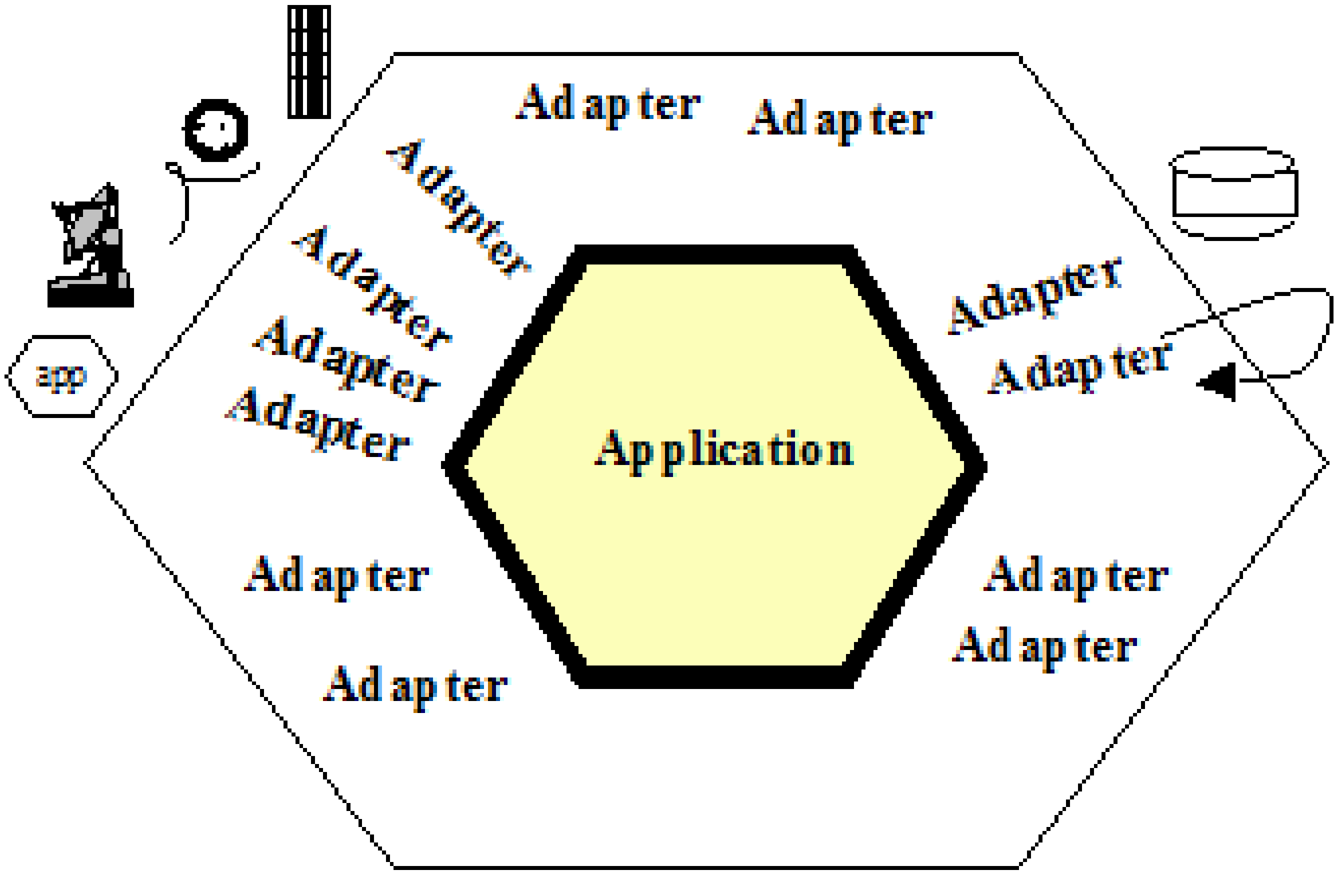
help		
tick	5	
check	widget type	Banner
tick	16	
check	widget type	None

Secure Enable Interact

How to deal with timer?

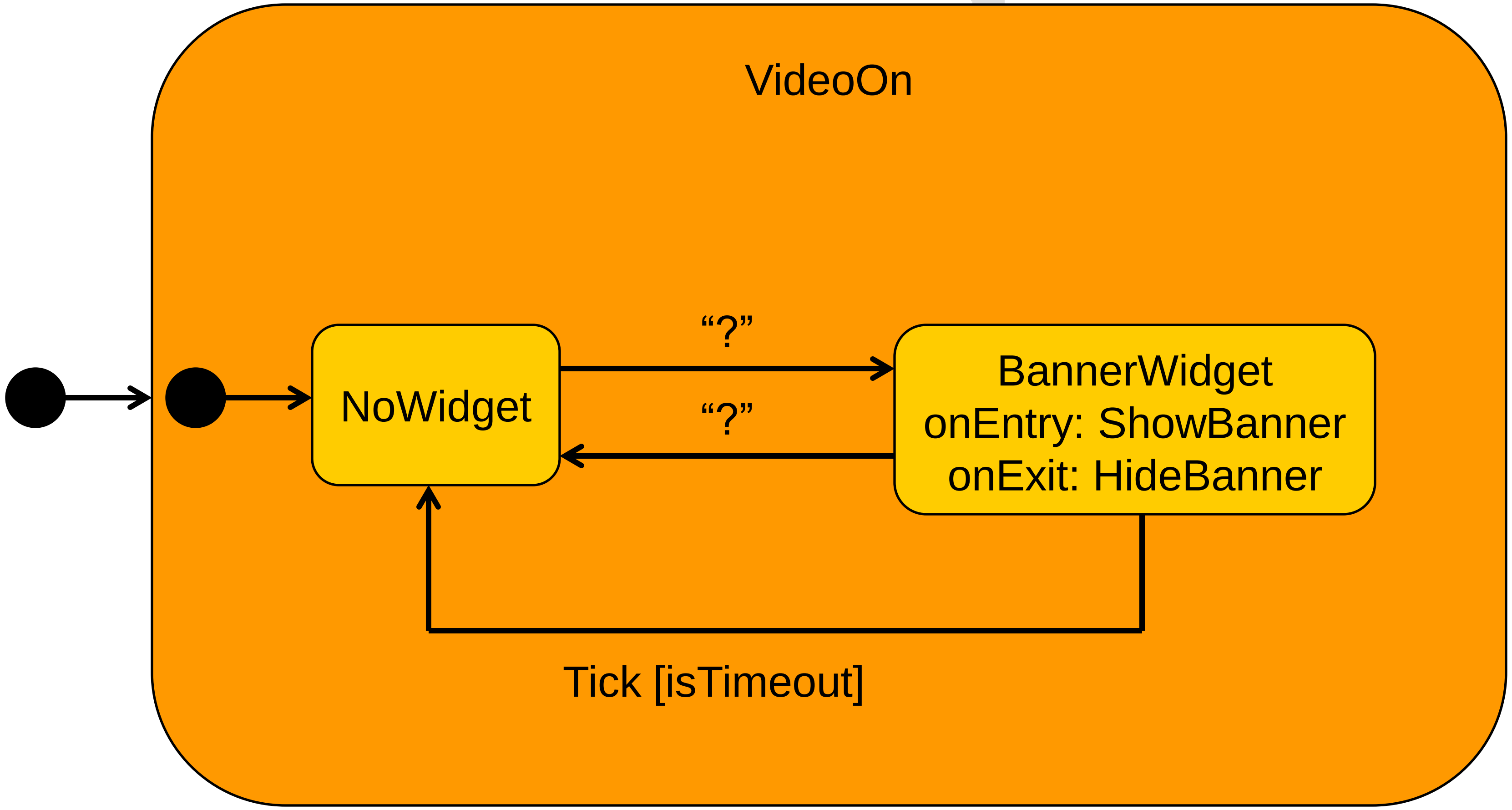
Any ideas?

Hexagonal Architecture



Controller

Banner State Machine



StateMachine.h

```
struct VideoOn : sc::state< VideoOn, StateMachine, NoWidget >
{
	VideoOn(my_context ctx);
	~VideoOn();

	typedef sc::transition< EvStandBy, VideoOff >		reactions;
	View::MediaPlayer					*pPlayer_;
};
struct NoWidget : sc::simple_state< NoWidget, VideoOn >
{
	typedef sc::transition< EvHelp, BannerWidget > reactions;
};
struct BannerWidget : sc::state< BannerWidget, VideoOn >
{
		BannerWidget(my_context ctx);
	sc::result  react( const EvTick & );
		~BannerWidget();

	typedef mpl::list<
		sc::transition< EvHelp, NoWidget >,
		sc::custom_reaction< EvTick >
	> reactions;

	View::Banner *pBanner_;
};
```

BannerWidget

```
#include <stdafx.h>
#include "StateMachine.h"

namespace ProgramGuide
{
        namespace Controller
        {
                BannerWidget::BannerWidget(my_context ctx)


                {

                }

                sc:
                {


                        return discard_event();
                }

                BannerWidget::~BannerWidget()
                {
                        context< StateMachine >().pView_->DisposeBanner();
                }
        }
}
```

Use Hexagonal Architecture to Completely Isolate System Under Test from Environment

Banner: Details

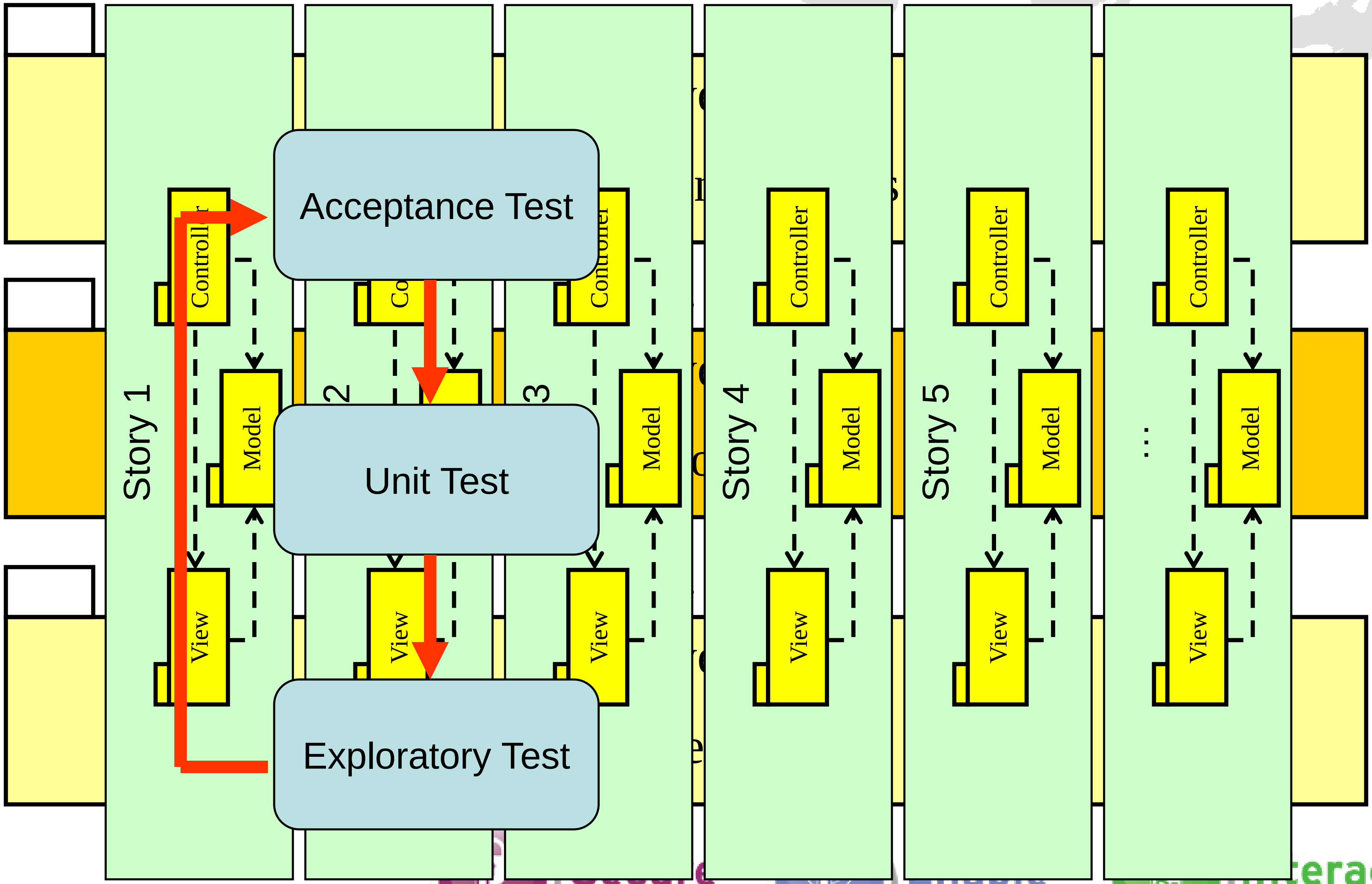
We do already have its acceptance test, but how many stories there are?

Secure Enable Interact

Banner Details Break Down

- Show Current Channel Title
- Show Current Time
- Show Current Program Title, Start Time and End Time
- Show Current Program Progress
- Show Current Program Parental Rating

Vertical Slicing



banner_test.cpp

```
#include <stdafx.h>
#include "View/Banner.h“

namespace ProgramGuide
{
	namespace View
	{
		static const int 		TIMEOUT = 15;
		static const std::string DATE = "2009-05-03";
		static const std::string TIME = "19:18";
		static const pt::ptime   CURRENT_TIME = pt::time_from_string(DATE+' '+TIME);
		struct banner_tester : public Banner
		{
			banner_tester()
			{
				BMOCK_CREATE_METHOD_MOCK(Model::*);
				SetTimeout(TIMEOUT);
				SetCurrentTime(CURRENT_TIME);
				test_set_channel();
			}
```

banner_test.cpp

```
    void test_set_channel()
    {
        const Model::Channel *channel = 0;
        BMOCK_EXPECT_RETURN(channel, pChannel_.operator->());
        BMOCK_EXPECT_RETURN(Model::ProgramIterator(),
                channel->GetCurrentProgram(CURRENT_TIME));
        BMOCK_REPLAY;
        SetChannel(Model::ChannelIterator());
        BMOCK_VERIFY;
    }
};
```

banner_test.cpp

```
BMOCK_TEST(banner_tester, test_is_time_out_false)
{
    BMOCK_CREATE_METHOD_MOCK(View::Banner::GetElapsed);
    BMOCK_EXPECT_RETURN(double(5), GetElapsed());
    BMOCK_REPLAY;
    BOOST_CHECK(!IsTimeout());
    BMOCK_VERIFY;
}

BMOCK_TEST(banner_tester, test_is_time_out_true)
{
    BMOCK_CREATE_METHOD_MOCK(View::Banner::GetElapsed);
    BMOCK_EXPECT_RETURN(double(TIMEOUT), GetElapsed());
    BMOCK_REPLAY;
    BOOST_CHECK(IsTimeout());
    BMOCK_VERIFY;
}
```

banner_test.cpp

```
BMOCK_TEST(banner_tester, test_show)
{
      BMOCK_CREATE_METHOD_MOCK(View::Banner::Show*);
      BMOCK_EXPECT(ShowCurrentTime());
      BMOCK_EXPECT(ShowChannelData());
      BMOCK_EXPECT(ShowProgramData());
      BMOCK_REPLAY;
      Show();
      BMOCK_VERIFY;
}

BMOCK_TEST(banner_tester, test_show_current_time)
{
      BMOCK_CREATE_METHOD_MOCK(
                  "void View::Banner::ShowCurrentTime(const char *)");
      BMOCK_EXPECT(ShowCurrentTime(TIME.c_str()));
      BMOCK_REPLAY;
      ShowCurrentTime();
      BMOCK_VERIFY;
}
```

banner_test.cpp

```
static const char *CHANNEL = "Channel 2";
BMOCK_TEST(banner_tester, test_show_channel_data)
{
        BMOCK_CREATE_METHOD_MOCK(View::Banner::ShowChannelTitle*);
        const Model::Channel *channel = 0; //we do not care for the pointer
        BMOCK_EXPECT_RETURN(channel, pChannel_.operator->());
        BMOCK_EXPECT_RETURN(CHANNEL, channel->GetTitle());
        BMOCK_EXPECT(ShowChannelTitle(CHANNEL));
        BMOCK_REPLAY;
        ShowChannelData();
        BMOCK_VERIFY;
}
```

banner_test.cpp

```
static const char *PROGRAM_TITLE = "Godfarther I";
static const std::string TIME_1 = "19:00";
static const pt::ptime START_TIME = pt::time_from_string(DATE+' '+TIME_1);
static const std::string TIME_2 = "22:00";
static const pt::ptime END_TIME = pt::time_from_string(DATE+' '+TIME_2);
BMOCK_TEST(banner_tester, test_show_program_data)
{
	BMOCK_CREATE_METHOD_MOCK(View::Banner::ShowProgramTitle*);
	BMOCK_CREATE_METHOD_MOCK(View::Banner::ShowProgramStartTime*);
	BMOCK_CREATE_METHOD_MOCK(View::Banner::ShowProgramEndTime*);
	BMOCK_CREATE_METHOD_MOCK(View::Banner::ShowProgramProgress*);
	const Model::Program *program = 0;
	BMOCK_EXPECT_RETURN(program, pProgram_.operator->());
	BMOCK_EXPECT_RETURN(PROGRAM_TITLE, program->GetTitle());
	BMOCK_EXPECT(ShowProgramTitle(PROGRAM_TITLE));
	BMOCK_EXPECT_RETURN(program, pProgram_.operator->());
	BMOCK_EXPECT_RETURN(START_TIME, program->GetStartTime());
	BMOCK_EXPECT_RETURN(program, pProgram_.operator->());
	BMOCK_EXPECT_RETURN(END_TIME, program->GetEndTime());
	BMOCK_EXPECT(ShowProgramStartTime(TIME_1.c_str()));
	BMOCK_EXPECT(ShowProgramEndTime(TIME_2.c_str()));
	BMOCK_EXPECT(ShowProgramProgress(END_TIME - START_TIME
	                                                    , CURRENT_TIME - START_TIME));
	BMOCK_REPLAY;
	ShowProgramData();
	BMOCK_VERIFY;
}
```

banner_test.cpp

```
        BMOCK_TEST(banner_tester, test_show_program_progress)
        {
            BMOCK_CREATE_METHOD_MOCK(
                "void View::Banner::ShowProgramProgress(long)");
            BMOCK_EXPECT(ShowProgramProgress(10)); //18 mins out of 3 hours
            BMOCK_REPLAY;
            ShowProgramProgress(END_TIME - START_TIME
            , CURRENT_TIME - START_TIME);
            BMOCK_VERIFY;
        }
    }
}
```

Banner.cpp

```
#include <stdafx.h>
#include "Banner.h"
#include "TimeString.h"

namespace pt = boost::posix_time;

namespace ProgramGuide
{
	void View::Banner::Show()
	{
		ShowCurrentTime();
		ShowChannelData();
		ShowProgramData();
	}

	BMOCK_VOID_METHOD(View::Banner,ShowChannelData, 0,())
	{
		ShowChannelTitle(pChannel_->GetTitle());
	}
	BMOCK_END
```

Banner.cpp

```
BMOCK_VOID_METHOD(View::Banner,ShowCurrentTime, 0,())
{
	ShowCurrentTime(TimeString(currentTime_));
}
BMOCK_END

BMOCK_VOID_METHOD(View::Banner,ShowProgramData, 0,())
{
	ShowProgramTitle(pProgram_->GetTitle());
	const pt::ptime start = pProgram_->GetStartTime();
	const pt::ptime end   = pProgram_->GetEndTime();

	ShowProgramStartTime(TimeString(start));
	ShowProgramEndTime(TimeString(end));
	ShowProgramProgress(end - start, currentTime_ - start);
}
BMOCK_END
```

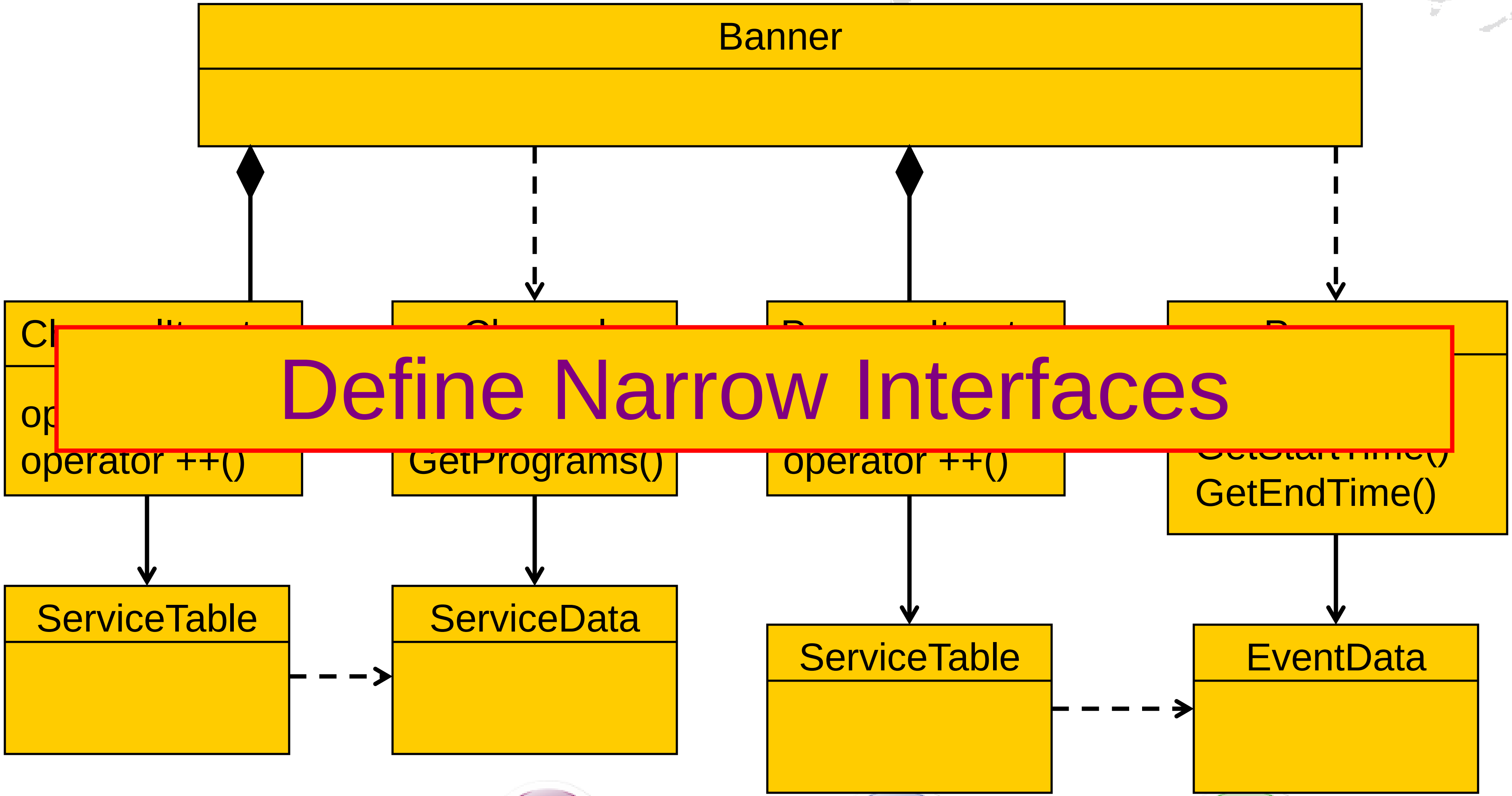
Banner.cpp

```
BMOCK_VOID_METHOD(View::Banner,ShowProgramProgress, 2,(
	IN(pt::time_duration, duration),IN(pt::time_duration, progress)))
{
	ShowProgramProgress (
		100L*progress.total_seconds() / duration.total_seconds() );
}
BMOCK_END

void View::Banner::SetTimeout(double t)
{
	timeout_ = t;
}

void View::Banner::SetChannel(const Model::ChannelIterator &ch)
{
	pChannel_ = ch;
	SetProgram();
}
```

Banner.cpp

```
    BMOCK_VOID_METHOD(View::Banner, SetProgram, 0, ())
    {
        pProgram_ = pChannel_->GetCurrentProgram(currentTime_);
    }
    BMOCK_END

    void View::Banner::SetCurrentTime(pt::ptime ct)
    {
        currentTime_ = ct;
    }

    bool View::Banner::IsTimeout() const
    {
        return timeout_ <= GetElapsed();
    }
}
```

Iterator

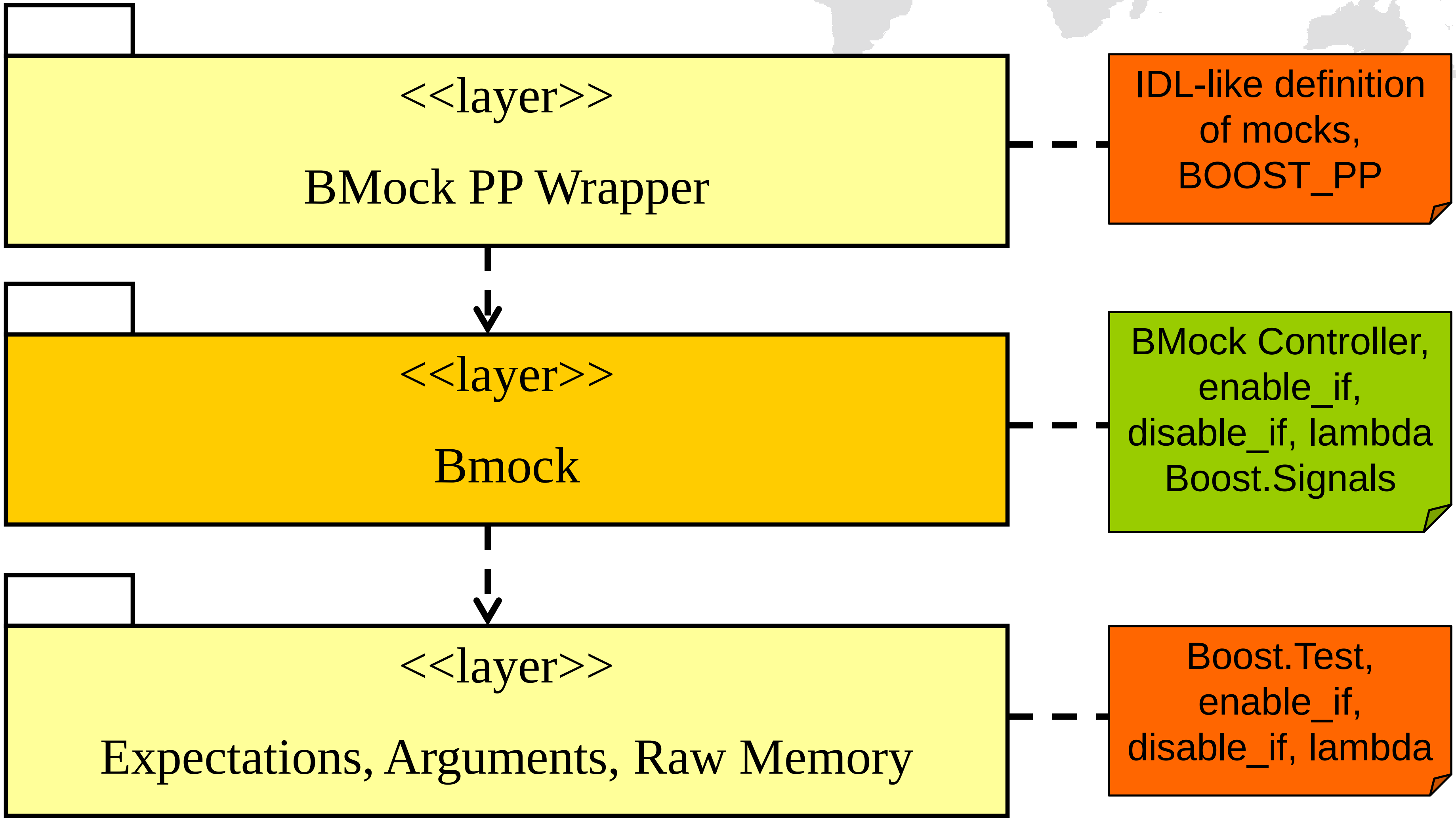


Define Narrow Interfaces

BMock Under the Hood

How would you build it?

Layers


<<layer>>
BMock PP Wrapper
<<layer>>
Bmock
<<layer>>
Expectations, Arguments, Raw Memory
IDL-like definition of mocks, BOOST_PP
BMock Controller, enable_if, disable_if, lambda Boost.Signals
Boost.Test, enable_if, disable_if, lambda

Secure Enable Interact

BMock Features

Feature	BMock	Google Mock	MockItNow
Defining Mocks	Static and dynamic for functions and methods through IDL-like macro wrappers	For virtual methods through inheritance and a cumbersome cut/paste. Mocking non-virtual methods is supported if class under test is a template	Dynamically for functions and methods using profiler API
Dealing with RAW-memory arguments	Full support for IN, OUT, and IN_OUT RAW memory arguments	Unclear, perhaps through advanced matchers	???
Support for matchers (ignore, greater, less, not equal)	Only partial support through IN_OUT arguments	Good support for matchers	Provides some support for matchers

Secure Enable Interact

BMock Features

Feature	BMock	Google Mock	MockItNow
Support for stubs	Stubs are fully supported	Support through AtLeast() and WillRepeatedly() clauses	???
Support for template classes and functions	Templates are not supported directly, but could be	Template classes are supported	Not supported
Integration with Boost.Test	Fully integrated (could be integrated with other testing frameworks)	Initially targeted for Google.Test, but could be used with other franeworks	Does not assume any unit testing framework, throws exception when some mock check fails
Delegating to real objects	Not supported, in each test each function/method is either mocked or not	More flexibility through WillByDefault and Ivoke()	Supported

BMock Features

Feature	BMock	Google Mock	MockItNow
Check object properties in mocked methods	Not supported, even **this** is ignored	Support through custom matchers	???
Simulating side effects	Supported through BMOCK_CALLBACK	Supported through DoAll and MockMutator	
Mocking constructors and destructors	Supported	Supported	Supported
IntelliSense, VisualAssist	Macro wrappers might put some limitations on code browser	No restrictions	No restrictions

Secure Enable Interact

Summary

- TDD is about preventing bugs
- Testability is major design decision factor
- Number of test cases grows exponentially
- User story scope is critical success factor
- Use Dependency Injection and factories
- Mocks facilitate programming by intention
- Apply hexagonal architecture pattern
- Define narrow interfaces

Secure Enable Interact

Next Steps

- Try your hands on Boost.Test
- Attend my tutorial on Boost.Preprocessor
- Drop me a line if you want to play with Bmock:

asher.sterkin@gmail.com

asterkin@nds.com