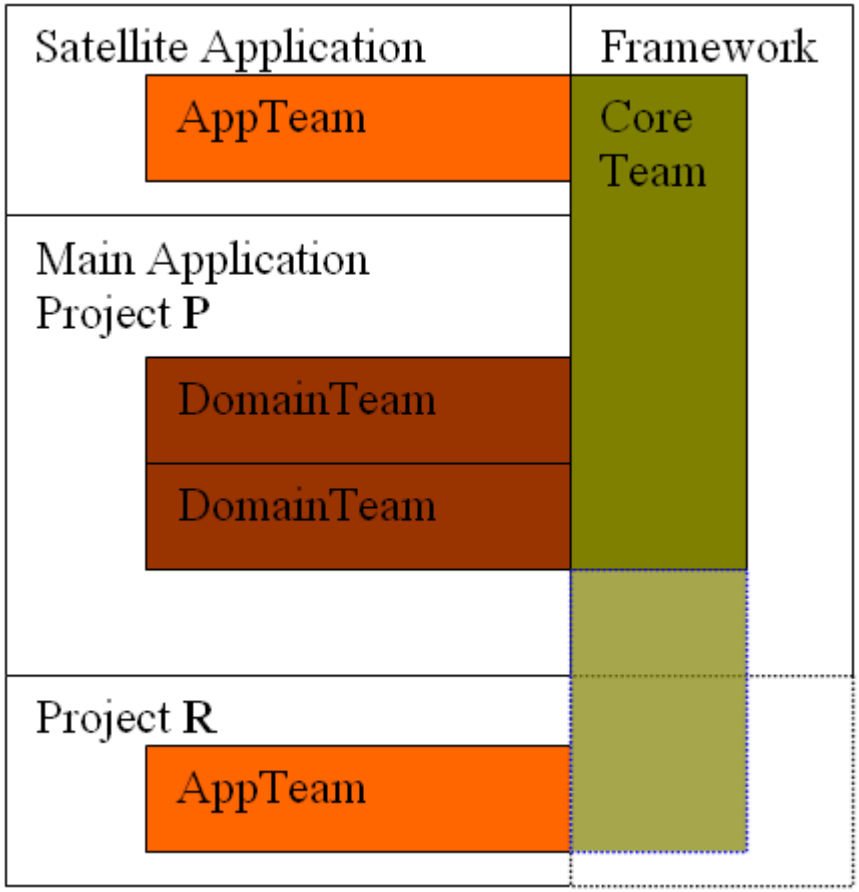
Introducing Unit tests to a large evolving application

by Mr. O and Mr. D

Introducing unit testing in a large evolving application

- A brand new application: Project R
- A 6 year old application: Project P

Team structure



Project R

- Context: Project specs
  - New satellite Application
  - Short development time
  - Scope still evolving
  - Rather simple business logic
    - Input + configuration (GUI)
    - Calculations + monitor (GUI)
    - Export results

Project R

- Goal
  - Deliver project
  - and write with new technology
  - and thus a new framework
  - and add unit testing/TDD
  - with a brand new team
  - and generate metrics
  - and …

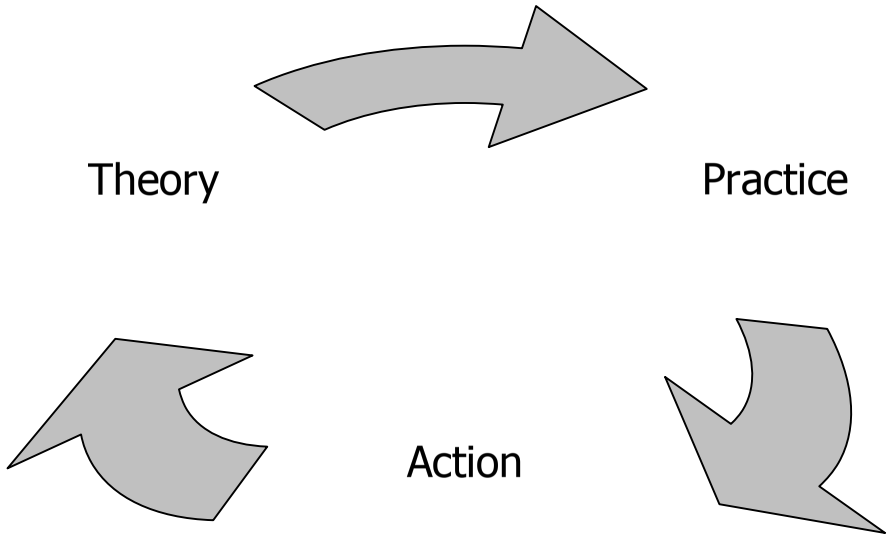
Project R

- Approach: force changes
  - Just do it & adapt along the way
  - Conservatives will follow if they see it working

- What about Unit Tests ?

Project R

- Several of my theories got into a fight with reality



Project R

- Theory:
  - Unit Tests are evident
  - Unit Tests are easy to write
  - Unit Tests document the code
  - Unit Tests are easy to maintain
  - Unit Tests lead to better design
  - Unit Tests keep your code alive

Project R : Theory

- Evident
  - Write
  - Document
  - Maintain
  - Design
  - Alive
- Unit Tests are evident
  - Better design
  - Refactorable code base
  - More features , less debugging
  - Proof enough to find on the internet.
    - Try to find a serious article telling Unit Tests are a silly thing to do.
    - I dare you ☺

Project R : Practice

- Evident
  - Write
  - Document
  - Maintain
  - Design
  - Alive

- Half the team not convinced...

- Early adopters needed to

  explain,
  show,
  prove,
  ...
  to conservative team members.

Project R : Action

- Evident
  - Write
  - Document
  - Maintain
  - Design
  - Alive

- Early adopters just did it.
- Conservative members were left the choice.
- Conservative members were left behind...
- Unit Tests got broken all the time by developers that didn’t check them.
- Early adopters end up isolated

  - ALL team members should at least verify Tests
  - More support necessary
  - Choking hazard !

Project R : Theory

- Selfevident
- **Write**
- Document
- Maintain
- Design
- Alive

- **Unit Tests are easy to write**
  - 1 feature, 1 test
  - 1 bug, 1 test

Project R : Practice

- Selfevident
- **Write**
- Document
- Maintain
- Design
- Alive

- Throw away code
- Change in code rippled through Unit Tests
- Write Unit Tests AFTER code

  → Your system is just not testable…

Project R : Action

- Selfevident
- **Write**
- Document
- Maintain
- Design
- Alive

- **Unit Tests are actual code**
  - keep them DRY !
  - Keep them focused
- **Teach**
  - Object-Mother pattern
  - Test-Driven-Development

Project R : Theory

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- Unit Tests document the code they test
  - How to use the class
  - Intention of the class & its methods
  - Contexts for classes

    → documented in test name

Project R : Practice

- Selfevident
- Write
- **Document**
- Maintain
- Design
- Alive

- **Transfer code to other team member**
  - Transfer didn’t include the Unit Tests
  - New member couldn’t read the tests

Project R : Action

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- No action yet,
- but perhaps reflect on this :
  - coding = communicating.
  - compiler likes cryptic as well as clear code.
  - code is read a lot.
  - stay alert for misunderstandings.
  - prevent abuse.

Project R Theory

- Selfevident
- Write
- Document
- **Maintain**
- Design
- Alive

- **Unit Tests maintain themselves**
  - Unit Tests yell immediately.
  - Production code yells only in production.

Project R : Practice

- Selfevident
- Write
- Document
- **Maintain**
- Design
- Alive

- **Cries for help were ignored…**
  - To much blood for 1 nurse
  - No feedback → No benefit

Project R : Action

- Selfevident
- Write
- Document
- **Maintain**
- Design
- Alive

- **Continuous Build**
- **Online reports**
  - Unit Test status
  - Unit Test coverage

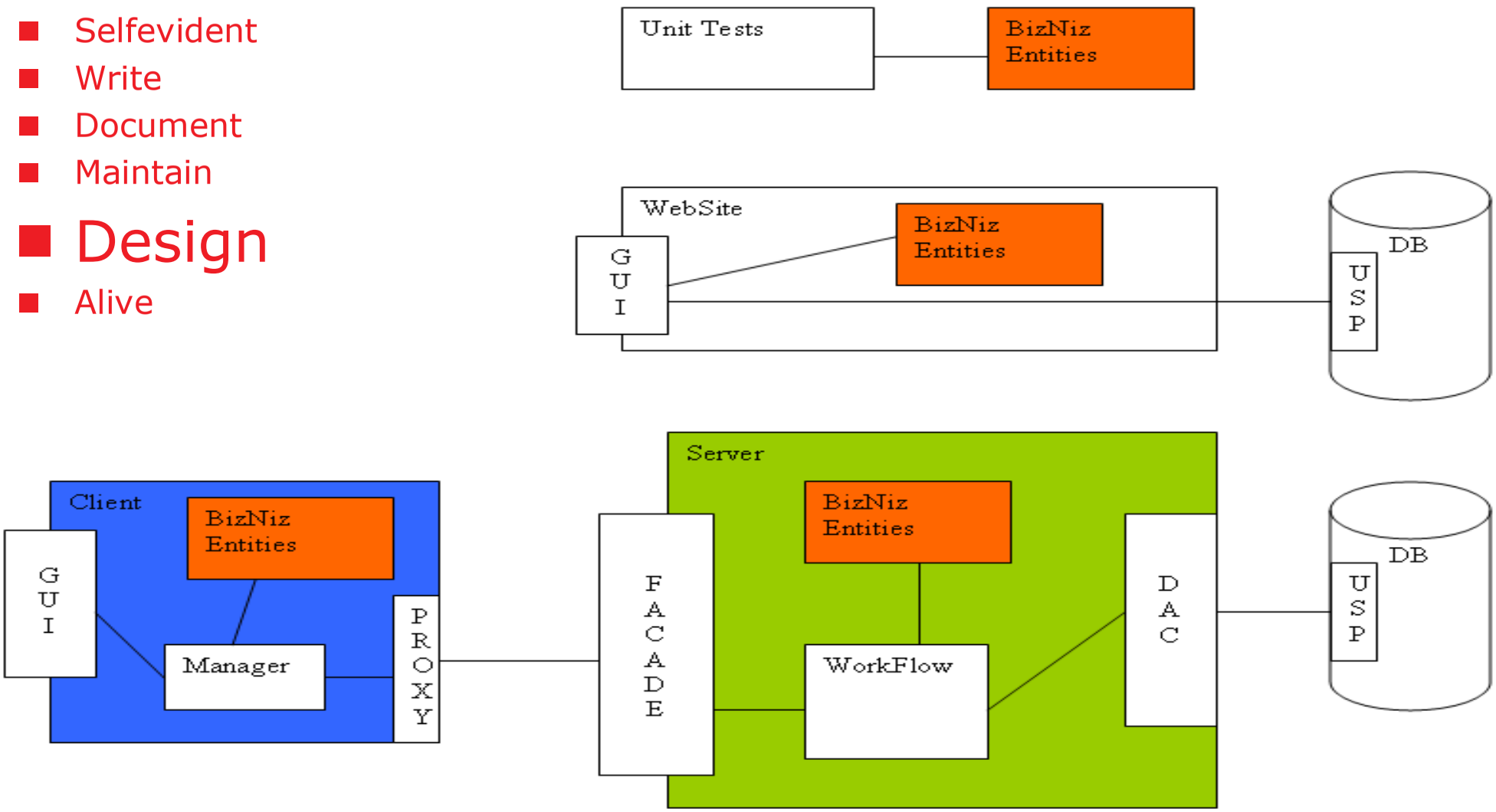
Project R : Theory

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- Unit Tests naturally leads to a better design
  - object model can exist in several contexts:
    - Client
    - Server
    - Unit Tests
    - Simplified GUI

Project R : Practice

- Selfevident
- Write
- Document
- Maintain
- **Design**
- Alive



Project R : Action

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- theory stayed on its feet
  - lots of untestable code was written
  - design makes refactoring possible

Project R : Theory

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- Unit Test keep code alive
  - Refactor safely
  - Enhance safely
  - Fix safely

Project R : Practice

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- Not enough coverage
- Test are not kept green

Project R : Action

- Selfevident
- Write
- Document
- Maintain
- Design
- Alive

- Whine
  - Didn’t help ☺

Project R : Reality check

- Theory:
  - Unit Tests are evident
    – if you know what they are
  - Unit Tests are easy to write
    – easy as production code
  - Unit Tests document the code
    – if you talk to humans
  - Unit Tests are easy to maintain
    – if all look after them
  - Unit Tests lead to better design
    – even thinking about
  - Unit Tests keep your code alive
    – if you keep the unit tests alive

Project R : summarized

- All truth passes through three phases
  - First it is ridiculed
  - Second it is violently opposed
  - Then it is accepted as self-evident

Schopenhauer (1788-1860)

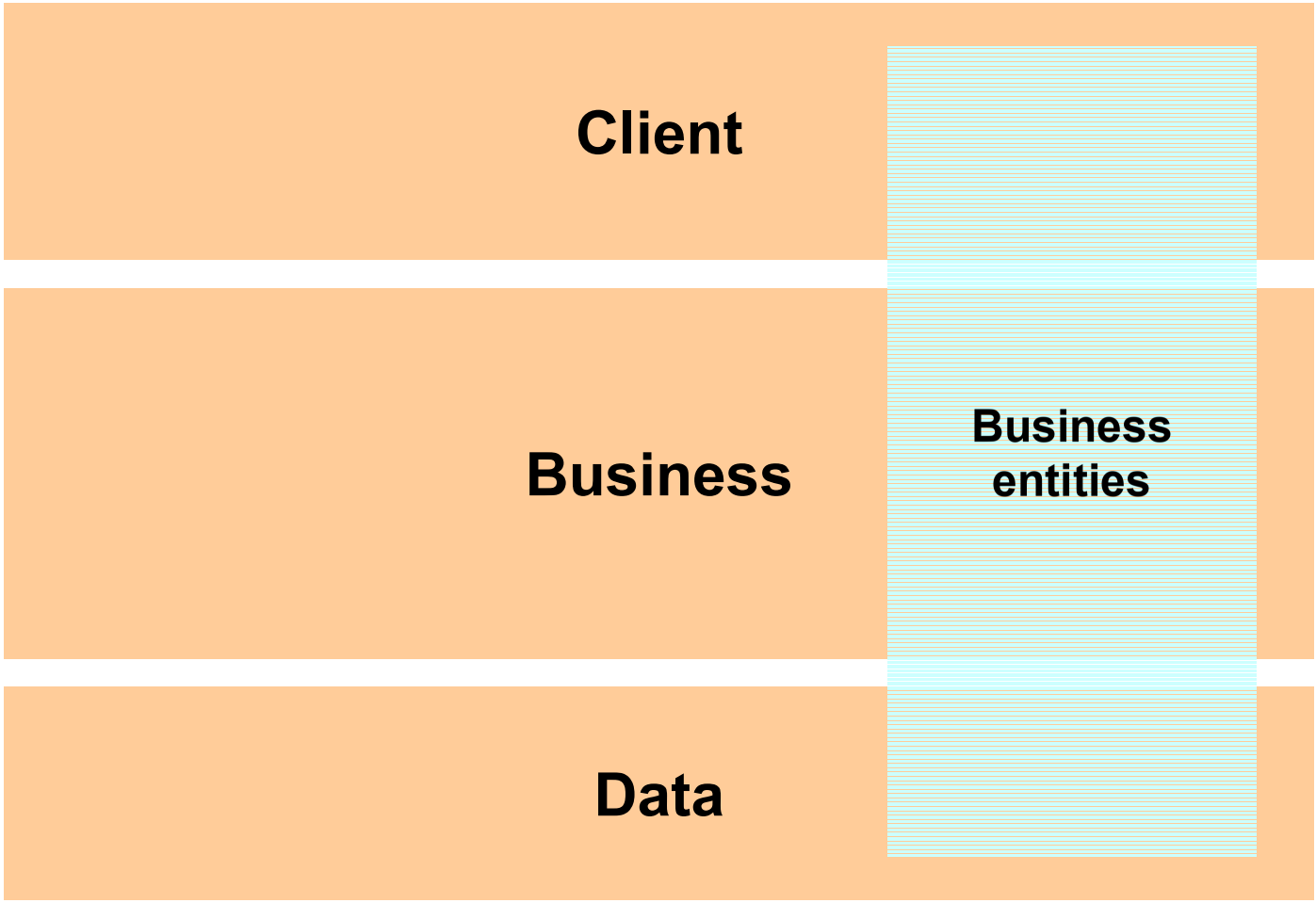
Project R

- Looking back
  - Experiments with Unit Tests
    only started after the application became stable
  - They got dropped under pressure
- One year later
  - Unit Tests were reintroduced
  - Supported by the Core Team / new technology
  - Units Tests are a deliverable
  - Team members experience the benefit

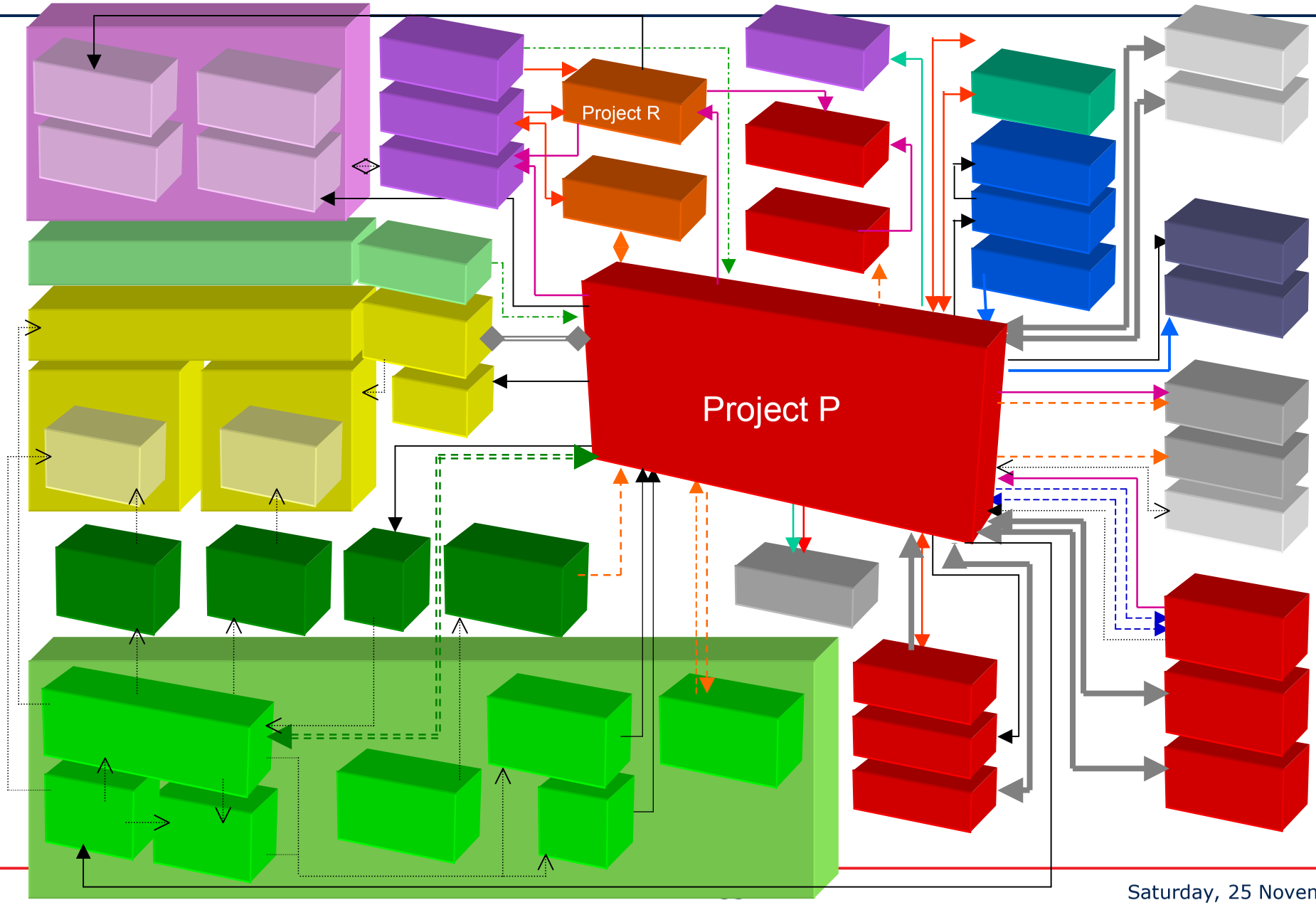
Project P

- Context
- Goals
- Introducing unit tests
- Problems and decisions
- Future of unit testing

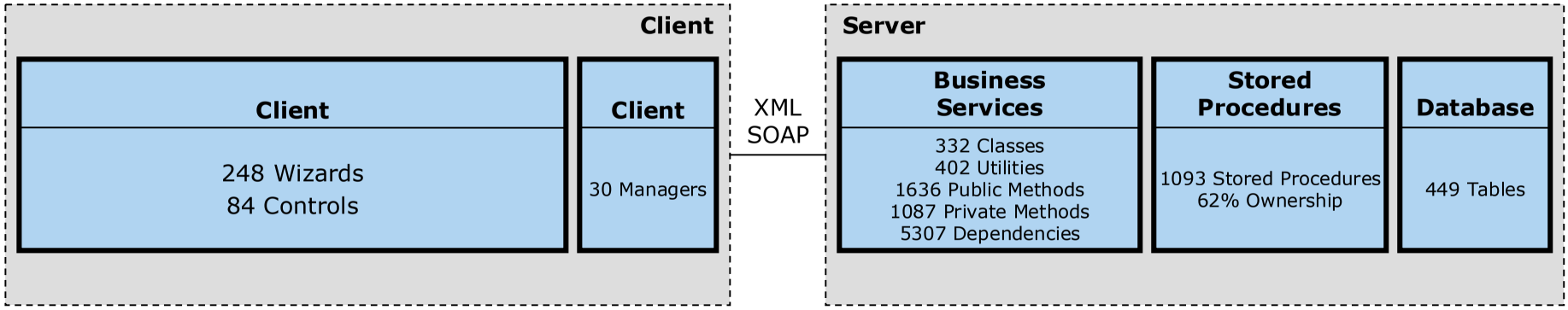
Project P – An overview (layers)



Project P – An overview (interfaces)



Project P – An overview (metrics)



- 100 additional stored procedures per release

Release	1.6	1.7	1.8
# Tables	329	360	369
# Stored Procedures	617	723	821

Project P – An overview

- 6 years ongoing development
- 60 – 100 people at a time
- Lots of people have come and gone
- Many scopes / analysis documents are lost
- A few people have seen the complete lifecycle
  - they are the backbone of project P

Project P – Goals

- In current release: Teach & Try
  - Convince dev leads
  - Roll out / explain what, how, when
  - Message: Write unit tests
    - Especially new code
    - While changing code
  - Unit testing not required for everything (yet)

Project P – Introducing unit tests

- Reflection & Decision
  - Only business rules / validation
  - No dependencies
    - No 3rd party interfaces
    - No database calls
  - Refactoring where necessary
  - → problem of the chicken and the egg:

    To refactor adequately one needs unit tests

    To create unit tests one must refactor.

Project P – Introducing unit tests

- Proof of concept
  - Set an example:

    Introduce unit tests according to the guidelines in a central component that can serve as an example
  - Few fully worked out unit tests on routines that abide by the rules

Project P – Introducing unit tests

- Communicate to the team
  - Go through dev leads
  - One spokes person appointed for advice
  - No organized follow up on dev leads
  - Mails with unit test results send to dev leads daily
  - Dev leads should make everybody create at least a few unit tests

  - → everything went too quickly
  - Developers contacted the spokes person directly
  - Dev leads out of the loop

Project P – Problems and decisions

- Code not testable
  - Business rule code is intermingled all over the place with database calls
  - Many code paths go to external systems
  - → We need major refactoring
  - → Decision: do refactoring where needed, but not structured

Project P – problems and decisions

Code not testable – example

- Old code (database call)

```
static private decimal CalculateRevenue(ContractEntity contract, DateTime startDate, DateTime endDate)
{
        RevenueEntityCollection revenues = RevenueFactory.SearchByContract(contract, startDate,endDate);
        decimal amount = 0.0M;
        foreach (RevenueEntity revenue in revenues)
        {
              …
              …
        }
        return amount;
}
```

Project P – problems and decisions

Code not testable – example

- New code

```
static private decimal CalculateRevenue(ContractEntity contract, DateTime startDate, DateTime endDate)
{
        RevenueEntityCollection revenues = RevenueFactory.SearchByContract(contract, startDate,endDate);
        return Revenue.Calculate(revenues, contacts.TotalNrOfContacts());
}
```

```
static private decimal Calculate(RevenueEntityCollection revenues, long totalNrOfContacts)
{
        decimal amount = 0.0M;
        foreach (RevenueEntity revenue in revenues)
        {
            …
            …
```

Project P – problems and decisions

- Unit test code somewhat complex
  - Not very maintainable
  - Does not speak to the developer's minds
  - → Decision: use factory methods
  - Even better: use the object mother pattern (not in decision)

Project P – problems and decisions

- Unit test code somewhat complex
  - Example (old code)

```
[Test]
public void NumOfUsers_2UsersAdded_Returns2()
{
   MyObject x = makeDefaultMyObject();
   x.AddUser("a","b");
   x.AddUser("c","d");
   Assert.AreEqual(1,x.NumOfUsers);
}

[Test]
public void HasUsers_2Users_ReturnsTrue()
{
   MyObject x = makeDefaultMyObject();
   x.AddUser("a","b");
   x.AddUser("c","d");
   Assert.IsTrue(x.HasUsers);
}
```

Project P – problems and decisions

- Unit test code somewhat complex
  - Example (new code)

```
[Test]
public void NumOfUsers_2UsersAdded_Returns2()
{
  MyObject x = makeAndInitializeMyObject();
  Assert.AreEqual(2,x.NumOfUsers);
}

[Test]
public void HasUsers_2Users_ReturnsTrue()
{
  MyObject x = makeAndInitializeMyObject();
  Assert.IsTrue(x.HasUsers);
}

private MyObject makeAndInitializeMyObject(MyObject x)
{
  MyObject x = new MyObject(“my”, “default”, “settings”);
  x.AddUser("A","b");
  x.AddUser("C","D");
  return x;
}
```

Project P – Problems and decisions

- No overview
  - Because of little follow up we lost track
  - Continuous integration not set up yet for coverage
  - → not a lot of useful information
  - → Decision: need to increase frequency of meetings
  - Maybe include coverage for next iteration/release

Project P – Problems and decisions

- Too little feedback
  - Mail with unit test results only gets sent to dev leads
  - Not every dev lead reads this mail instantly
  - Mail not considered of high importance
  - The mail content does not get dispatched to developers
  - → Decision: make unit testing more important by following up closely and requiring it as a deliverable
  - → Render dev leads more involved and pro-active

Project P – Problems and decisions

- Not a high coverage reached
  - Because of too little code testable
  - Because of perspective it's of too little importance
  - → Decision: draw more code into business entities to render it more testable through custom validation and xml schema's
  - → requires refactoring
  - → along the way with next iteration/release

Project P – Problems and decisions

- Focus
  - No information directly available to know where to write tests to have the maximum return
  - → Decision: create a unit test for every new bug that is about to get fixed
  - → Decision: identify problem areas by looking at issue tracking
  - → concentrate work on a top 3 of subsystems that are peak issue creators
  - → watch these peaks go down
    - If not, review and re-assess your work on these peaks
    - If yes, work on the next top 3 and re-iterate

Project P – Reality Check

- Problems encountered
  - Decisions taken
- Overall
  - The developers that caught on to it, really like the outcome
  - The dev leads that do follow up, really believe in the system
  - The managers that understand the goodness, follow them up

Project P – New Goals

- In NEXT release
  - Introduce organized unit testing with close follow up
  - Pro-active dev leads
  - Review unit testing along with code
  - Unit testing required
  - Get reports from issue tracking
  - Create unit test for every bug
  - Create unit tests for new development/features

Project P - Future of unit testing

- Possibilities
  - Add 3rd party interface testing
  - Add database unit testing
  - Add mocks
  - Add inversion of control
  - Add coverage reports
  - ... ?

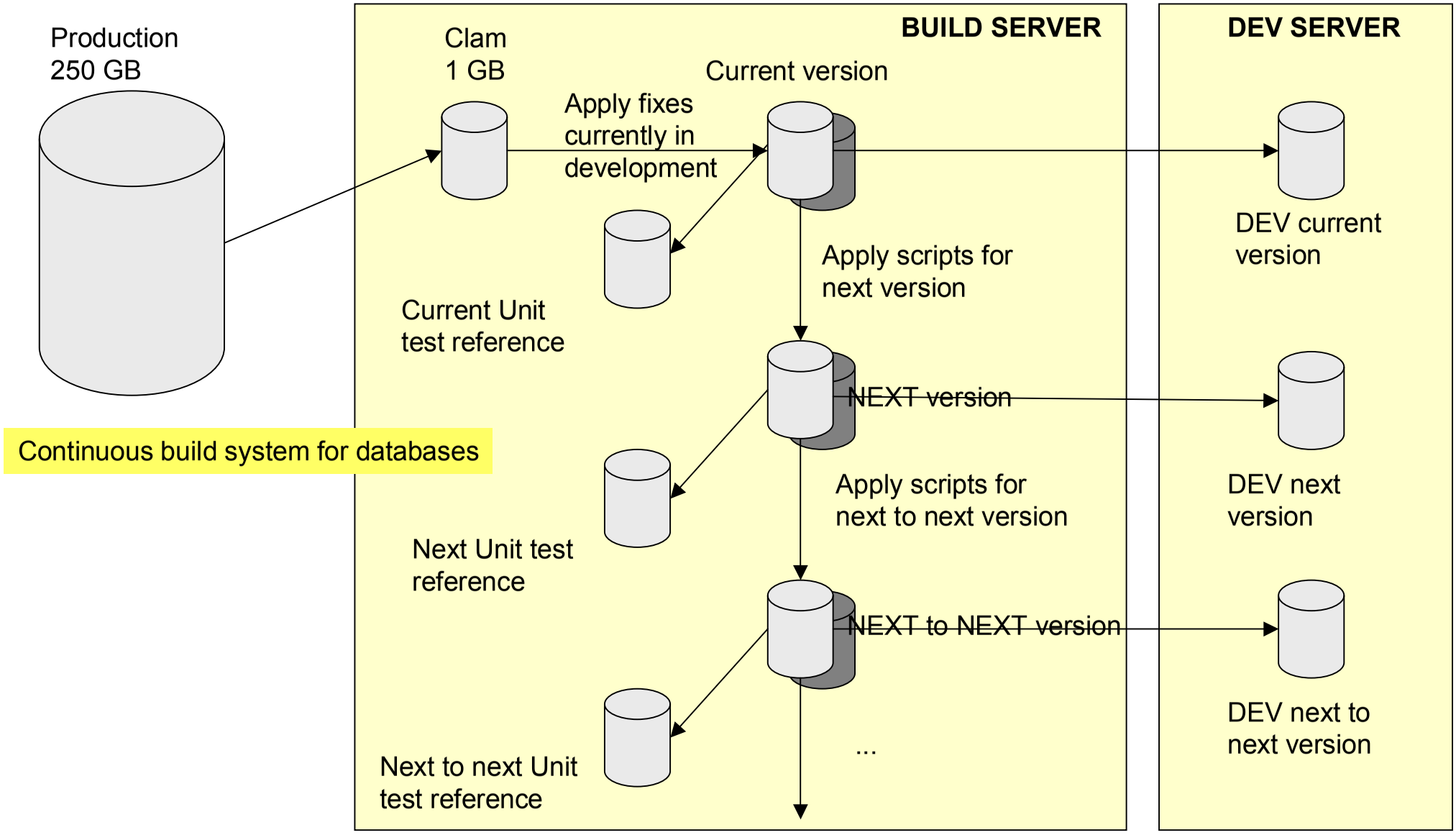
Project P - Future of unit testing

- Possibility: Add 3rd party interface testing
  - Analyze how the project uses the 3rd party interface
  - For each kind of use, write a unit test to make sure the interface works as expected
  - Gives feedback on the actual behaviour of the 3rd party in place

Project P - Future of unit testing

- Possibility: Add database unit testing
  - Makes all calls to the database work and opens a lot more possibilities to unit test
  - Possible implementation:
    - For each unit test, create a set of data and store this in a reference unit test database
    - Build this database every day
  - Possible implementation
    - Use (N)DbUnit
    - Store the unit test data in your unit test
    - Maintain the unit test data in your unit test

Project P – Future of unit testing



Continuous build system for databases

Project P - Future of unit testing

- Possibility: Mocks
  - Is the least known and understood for now in the team
  - Records the uses of the external component
  - Allows fine-grained tests of usage
  - Allows writing of unit tests for one component without needing the used sub-components

Project P - Future of unit testing

- Possibility: Add inversion of control
  - Create an external configuration source
  - This configuration should allow selection of the used component
  - For unit tests: configure a stub or mock
  - For non-unit tests: configure the real thing
  - → allows e.g. testing of your code in cases the component is unavailable

Project P - Future of unit testing

- Possibility: Add coverage reports
  - Gives feedback on the coverage of your unit tests
  - Makes the unit tests more visible to
    - DEV
    - Management

Project P - Future of unit testing

- Possibility: ???

Questions ?

Links of interest

- Why Unit Test anyway ?
  - http://www.awprofessional.com/articles/article.asp?p=379759&rl=1
- F-22 fighter software proves Unit Testing works.
  - http://www.iplbath.com/pdf/p0015.pdf
- What not to Unit Test.
  - http://www.artima.com/forums/flat.jsp?forum=106&thread=126923
- How to Unit Test.
  - http://junit.sourceforge.net/doc/testinfected/testing.htm
- Do Unit Tests slow you down mrs. Brown ?
  - http://www.xprogramming.com/publications/SP99%20Extreme%20for%20Web.pdf
- Inspiring people:
  - Martin Fowler, Kent Beck, Pascal Van Cauwenberghe