HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Agile Frameworks with Test-based Development

or

What is the cost of Test-based Development?

Prof. Peter Sommerlad

HSR - Hochschule für Technik Rapperswil

Institute for Software

Oberseestrasse 10, CH-8640 Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Contact

- **Peter Sommerlad**
  - peter.sommerlad@hsr.ch
  - I am a software engineer by heart and soul. As co-author of "Pattern-oriented Software Architecture - A System of Patterns" I won the Software Productivity Award 1996. In addition to developing software I write patterns and shepherd other pattern authors, thus helping them improving their publications. I am co-authoring a new book on Security Patterns due 2005.
  - since 09/2004 professor for informatics at HSR

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

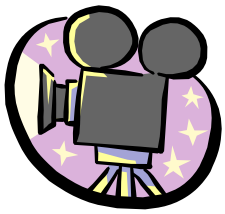
Agenda

- **What it is about?**
  - Application Framework
  - Test-based Development
  - Our Infrastructure
- **Basis – Situation in 1997**
- **Initiation to Test-based Dev. - Vision in 1998**
- **First Phase of test-based Development – 1998 to 2001**
- **Second Phase of test-based Development – since 2002**
- **Outlook and Summmary - today**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

What is an Application Framework?

- **object-oriented class library**
- **„main program“ lives in the library**
  - application architecture founded by the framework
  - construction of application families
- **Hollywood-Principle: Don‘t call us, we call you!**
  - control flow is from framework to application components
- **Extendability and Configurability**
  - applications created by subclassing framework components
  - and configuring them together with framework parts
  - simple applications can be created by just (re-)configuring framework components (e.g., by automated wizards)
- **Examples:**
  - Smalltalk's class library
  - Java: e.g., Servlet Engine, Application Server
  - C++: ET++, MFC, "cappf" (the framework this talk is about, still looking for a good name)



HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

AF: Product Families and Reuse

- **Application Frameworks create product families**
  - similar applications in a common context
  - reuse of common infrastructure
  - "delta" development, high reusability
- **Development of AF often evolutionary**
  - designing application frameworks on the "drawing board" leads to unnecessary complexity and features (YAGNI - you ain't gonna need it)
  - use at least to concrete applications to drive your design
  - factor commonalities of related applications into a framework
- **Reuse of application architecture and infrastructure**
  - for a family of applications

➢ **High Quality Requirements on Application Frameworks**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

What is Test-based Development?

Automated Tests guarantee Software Quality

- **Unit Testing**
  - For each component/class/method with a non-trivial implementation implement one or more testing methods.
  - Test should be able to run in isolation. Required environmental objects are simulated by so-called "mock-objects" to reduce dependencies.
  - Often tests cover both border cases and normal behavior.
  - Unit tests check the **technical** quality of the implementation.
- **Functional Testing**
  - Relevant external behavior is tested on a system level.
  - Testability can be improved through isolation of subsystems with mock-objects.
  - Check quality with respect to the **domain**.
- **Ideally: Test-First**
  - Write test(s) and interface first, let the test fail, then write the corresponding implementation until the test runs OK.

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

How does a Test-case look like?

- **Extract of a test for a Bitset implementation**

```
void REBitSetTest::ManyTests() {
        StartTrace(REBitSetTest.testCase);
        REBitSet empty;
        REBitSet full(true);
        t_assert(empty.IsSubSet(full));
        t_assert(empty.IsSubSet(empty));
        t_assert(full.IsSubSet(full));
        t_assert(!full.IsSubSet(empty));
        t_assert(full == full);
        t_assert(full.IsEqual(full));
...
```

  - ugly: too much tested in a single overloaded test
  - practical & compact: helper objects instantiated once

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Essential Infrastructure

- **Development Environment**
  - past: SNiFF+ (WindRiver), more modern: Eclipse
- **Repository**
  - cvs: cost free, well proven, sufficient
- **Workspaces**
  - per developer/project, based on common repository
- **Daily Build**
  - nightly cron job based on clean checkout from repository
  - automatic emails on errors (and success)
- **„Sharpen your Tools“**
  - Don't be afraid of home-grown scripts (treat them as software)
  - automate everything that is done more than once and that requires more than one manual step
- **Developer Communication and Collaboration Tools**
  - whiteboards for design sessions
  - WikiWeb for persistent information
  - automatic emails triggered by check-ins and build failures

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

The Application Framework

- **C++ Application Framework for (Server) Applications**
  - Web Server, Internet Banking, Web-To-Host, Web Applikationen
  - Product: Frontdoor – Web/FTP Protection Reverse Proxy with SSO
- **Infrastructure = Foundation classes**
  - String, Anything, (SSL-)SocketStream, MmapStream, SysLog, Tracing
- **Framework for protocol handling, multi-threading**
  - Leader-Follower Thread Pool, Acceptor, RequestProcessor, Dispatcher
- **Framework for request handling**
  - Session, SessionHandler, Context, Role, Page, Action
- **Framework for output generation**
  - Renderer (HTML, XML, recursive scripting)
- **Framework for unified data access**
  - DataAccess, ParameterMapper, ResultMapper, DataAccessImpl
- **Framework for initialisation and application bootstrap**
  - AppBooter, Module, Application, Server
- **Testing Framework**
  - Unit Testing, Configured Tests, Performance Tests

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Problems around 1997

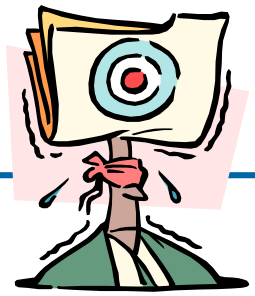
- **First framework applications successfully deployed**

but

- **Developers hedge their "own" versions of the framework**
  - local variants, because no repository access at a customer's site
  - effort for project spanning consolidation feared
  - individual "hacks" of the framework to corner-cut weaknesses
  - individual preferences of developer "prima donnas"
- **correction of core abstractions and infrastructure required**
  - application experience uncoverd improvement potential
  - back-integration of application code into framework core
  - deficits in communication among developer community

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Problem Reasons



- **fear of changing the framework**
  - risk: destabilising of existing code
  - risk: change propagation into „done“ applications
  - risk: increased and repeated (manual) test efforts
  - risks are hard to calculate
- **quality problems in the framework**
  - FIXME comments: quick hacks, trial and error
  - usage of harmful C-legacy: sscanf, vsprintf, char *
  - partially unclean behaviour with multi-threading
  - partially unclean memory management
- **effort across several projects hard to calculate**
  - improvements delayed into phases of lower work load
  - individual project calculations don't fit with frameworks
  - leadership supporting framework approach required

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Changes required too much Courage

- **and sweating of fear that something goes wrong**
- **Example: elimination of `char *` by a String class**
  - bad experiences with string classes made initial designers reluctant to introduce one.
  - C experience of some developers also

  BUT
  - `sscanf()` and `sprintf()` considered extremely harmful
  - char arrays as buffers extremly dangerous
    - buffer overflows are well know today
- **Courage, encouragement and personal demonstration was required when changing the framework to use "String"**
- **Interfaces became syntactically incompatible partially**
- **BUT: risky implementations where reduced**
- **String class allowed automation and later significant optimization of memory management**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Vision 1998

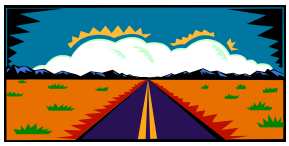
1998: Demonstration of Test-first Programming by Kent Beck and Erich Gamma

- „We need to do that!“

- **If the framework would have automated tests, then changes should be possible without problems.**
- **Variants of techniques could be checked against each other easily.**
- **Improvements of framework technology might be possible without impact on applications.**
- **Impact analysis of (interface-) changes becomes possible**
- **Couple Daily Build with automatic testing.**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Solution: Automatic Testing



- **Adapted CPPUnit a Unit-Testing Framework to our needs**
  - simpler (more portable) C++ (no templates, no exceptions)
  - macro-based test registration (no reflection aka RTTI)
- **Started with tests infrastructure (foundation) classes**
  - found potential problems in corner cases
  - re-implementation (refactoring) of internal structures for optimization became possible
- **Tests for core abstractions of the framework**
  - default-implementions ensured by tests
  - difficulties on the border to functional/application tests
  - not all classes were testable in isolation (framework cyclic dependencies)
  - extension of some core classes required for better testability (e.g., Context)
- **Extended test framework for special framework abstractions**
  - configurable tests
  - scriptable/configurable tests for Renderer, Action, DataAccess

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Testframework for C++ Unit Testing

Examples of macros defined

- **t_assert(condition)**
  - check if condition is true
- **t_assertm(condition,message)**
  - provide an additional message, if condition is not self-explaining
- **assertEqual(expected,actual)**
  - compare actual to expected
  - be careful with double (equality is usually not OK)
    - use assertDoublesEqual(expected,actual,delta)
- **assertEqualm(expected,actual,message)**
  - ditto with a specific message
- **per data type a corresponding assertEqualXXX(expected,actual)**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Specialities

- **Daily Builds with automatic tests on sevaral platforms**
  - Solaris 2.6, 2.7, Linux, in debug and optimized mode each (+Windows)
- **configurable tests and scripted tests**
  - variation of parameters without code duplication
  - fast creation of tests for different parameters
  - if errors pop up in applications, tests scenarios can sometimes be created instantaneously without code&compile cycles
- **end-to-end tests vs. Unit Tests**
  - frameworks often hinder testability through closely interlocked mechanisms
  - some classes depend on almost all of the framework and are testable only after "booting" a complete framework application
- **performance (of) tests**
  - timing of tests is important (also to optimize overall testing time)
- **load tests**
  - require substantial infrastructure to be useful
  - are not practical in scheduled automatic test runs

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Example of configured Test / Scripting

```
/StringTokenizeRendererListOfTokensTest {
     /Env {
     /InString        "abc;def;ghi;jkl"
     /InToken         ";"
   }
   /Renderer {
      /StringTokenizeRenderer {
        /String       { /Lookup InString }
        /Token { /Lookup InToken }
        /RenderToken     "1;3"
      }
   }
   /Expected "abcghi"
}
```

Testname

Parameters

Class

Script / Test Object

Result

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Results (1)

- **Stability**
  - better coverage of corner cases
    - e.g., 0, 1, some, many, negative ?

- **Refactoring of the Framework enabled**
  - e.g., reduction of unnecessary locking, optimized (thread-local) memory management
- **Portability on other platforms ensured**
  - AIX, Windows NT, parts of the framework on an exotic IBM mainframe OS (TPF)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Results (2)

- **enabled exchange of (bad) implementations**
  - eliminate C legacy and "FIXME" comments
- **better interface design of new code**
  - test-first and testability are good guides for design
- **flexibility becomes easier**
- **developer trust in "foreign" code improved**
  - allowed better consolidation of different coding streams
  - reduced better controllable risk by writing a test case before a change to code is applied

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Example: Thread-local Memory Management

- **idea: optimize memory management**
  - every thread manages own memory pool for "transient" objects
  - no more locking required on allocation or de-allocation
- **unit testing of new memory mangement**
  - guarantee funcitonality
  - help design the MM API
- **unit tests for classes relying on the new MM**
  - guarantee that these classes work also with the new MM
  - help also to uncover incompatibilites with indirect dependencies

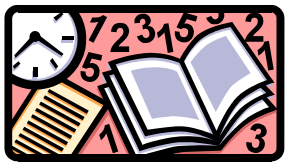
- ➢ **fundamental change of infrastructure classes became possible without changing existing (application) code**
- ➢ **performance gain significant with multi-processors under load**
- ➢ **predictability and stability of resource usage (memory) increased**

Remaining Hurdles (V1)

anno 200[01]:

- **application code without automated tests**
  - especially from customer's developers
  - lack of experience in test-based development
  - locked into a specific framework release
  - fear of incompatible upgrades (even simple ones)
  - procrastination of refactorings even with minimal interface changes
  - impact analysis of framework changes still hard
- **"dark corners" in the framework (design)**
  - initial design not done with unit tests and testability in mind
  - sometimes refactorings kept unneeded semantics
  - unused features and semantics where kept and manifested by tests written after the code
- **tests run too slow and the whole suite is run too infrequently (especially when run by developers)**

Lessons Learned (V1)



- **you need time to learn test-based development**
  - writing tests is programming
  - good tests are easily comprehensible as good code
  - tests require Refactoring to become or stay in good shape
  - back then literature and experience on unit testing was scarce
- **thinking about tests and writing them down (also manual ones) improves understanding of requirements**
  - what you can not or don't want to test is often not relevant
  - "you can test everything that is relevant" is almost always true
  - most tests can and should be automated
  - use Use Cases to derive Test Cases
- **programming tests (first) improves interface design dramatically**
  - developer is "victim" of his own design decisions
  - simpler (less complex) design is favored, because of testability
- **ugly tests are often an indication of bad design or lack of testability of the tested code**
  - Refactor!

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Solution: Automatic Testing of Applications

- **embed applications into Daily Build**
  - source code incompatibilities are recognized immediately
- **educate application developers in automated testing**
  - requires training and coaching
- **for new application releases implement tests**
  - for changes and new code always write automated tests
  - for existing code new tests are written only partially
- **shared repository for framework and applications across team and company boundaries**
  - complete Daily Build in each organization
  - configuration differences automatically "patched"
  - scaleability of this approach limited to a few sites (2-4)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Situation today

- **impact analysis of changes is simpler, because (almost) all code depending on the framework is in the repository**
  - innovative changes become possible again
- **application deployment is tagged in the repository. small patches or fixes are branched on that tag.**
  - working with cvs-branches requires care
  - new releases are developed on the current HEAD version
- **refactorings breaking interfaces are possible again**
  - necessary changes of applications are obvious and done immediately
  - but deployment is only done when needed or with a new release of the application
- **important major refactorings can still be elaborate**
  - design mistake take revenge someday
  - test-first development from the beginning would have helped
- **tests take still too long**
  - nightly builds instead continuous build (6 platforms)
  - false positives misguide or lead to "broken windows"

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Problem False Positives

- **Reasons:**
  - timing dependencies (second flip)
  - external dependencies (server unavailable)
    - DNS - lookup, socket tests beyond „localhost“

```
Running SocketStreamTest
!!!FAILURES!!!
Test Results:
Run:  4   Failures: 1   Errors: 0
(28 assertions ran successfully in 2851 ms)
There was 1 failure:
1) line: SocketStreamTest.cpp:186 SocketStreamTest: socket
!= NULL; socket creation to [www.switch.ch:80] failed
```


HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Lessons Learned (V2)

- **tests shouldn't depend on external resources outside the control of the test runner**
- **bring external resources under your control (e.g., DB) in a consistent initial state before you run tests (automatically)**
- **writing tests for existing code is much harder and more effort than coding test-based (write tests while coding)**
  - some badly testable code should be better rewritten by implementing it test-first.
- **Refactoring without automatic tests is dangerous**
- **write tests demonstrating bugs instead of debugging**
  - reproduce errors
  - verify hypthesis about code behavior
  - retire your debugger or forbid it

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Vision and Wishes today

- **automatic testing at light speed**
  - always and continually run all tests, no manual work required
  - every check-in should trigger tests, auto-reject code that fails
- **still dreaming of always using the current version and auto-deployment when all tests run OK**
  - no more manual testing required (including system tests)
- **customers honor software quality well before purchasing and not just after successful operation and changes**
  - how to get economic success and market awareness?
- **enable customers to write their own automated functional tests without requiring programmer knowledge**
  - existing approaches:
    - **FIT (fit.c2.com) and fitnesse (fitnesse.org)**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Essentials - Pragmatic Programming

- **write automatic test and let them run**
  - Refactoring becomes possible
  - interface design and testability improved
  - requirements are understood better
  - aver every check-in, make test run 100% OK
- **repository encourages experimentation**
  - work speeds up, since going back is always possible
  - try alternative implementations without risk
  - fewer integration problems when check-ins are frequent
- **automate ALL repetitive tasks**
  - deployment of applications
  - creating test data
  - cron-jobs for tests and builds
  - mail-messages at every check-in or test failure

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was it successful? (1) Yes and No.

- **For Developers: a clear YES!**
  - better code, better design, refactoring enabled
  - much less "debugging stress"
  - much easier to introduce new ideas
  - sustainable code base
- **For Customers: Yes and No.**
  - test-based development seems to be more expensive and requires learning by own developers.
  - solution quality almost too good, especially when requirements change (customers learn that changes become easy, quickly)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was it successful? (2) Yes and No.

- **As a Service Provider: commercially NO.**
  - happy customers canceled support contracts, because they consider them unnecessary
  - quality is valued but its price not payed
  - worse quality provides more customer contacts and better customer binding (= more business)
    - ➢ easier to learn about new and potential projects!
  - new customers are hard to convince about advantages, especially since the service market grew tougher
  - framework-based development and frameworks are hard to sell (especially for a small company)
- **Therfore: Framework as Open Source in 2005**
  - ➢ stay tuned... watch HSR's intstitute for software

Literature

- Kent Beck: Test-driven Development by Example
- Dave Astels: Test-driven Development
- Lisa Crispin, Tip House: Testing Extreme Programming
- Andy Hunt, Dave Thomas, Mike Clark:
  Pragmatic Starter Kit
  http://pragmaticprogrammer.com
- Johannes Link: Unit Tests mit Java

Advertisement :-)

- Pattern-oriented Software Architecture:
  A System of Patterns
  (Buschmann,Meunier,Rohnert,Sommerlad,Stal)
- Security Patterns (2005, M. Schumacher et al)
- Questions about Patterns, etc.: peter.sommerlad@hsr.ch