HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Was extreme Qualität kostet?

Testbasierte Entwicklung mit Application Frameworks

Prof. Peter Sommerlad

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Kontakt

- **Peter Sommerlad**
  - peter.sommerlad@hsr.ch
  - Ich bin Software Engineer von ganzem Herzen. Als Co-Author von "Pattern-oriented Software Architecture - A System of Patterns" gewann ich 1996 den Software Productivity Award. Seit 1997 wende ich Patterns und agile Software Entwicklungsverfahren in der Schweiz in Projekten für Kunden an. Neben der Arbeit kümmere ich mich als Autor und "Shepherd" um Pattern Literatur.
  - seit 09/2004 Professor für Informatik an HSR

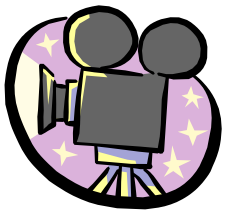
HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Agenda

- **Begriffserklärungen**
  - Application Framework
  - Testbasierte Entwicklung
  - Infrastruktur
- **Ausgangslage – Situation 1997**
- **Initialzündung und Vision – 1998**
- **Erste Phase testbasierter Entwicklung – 1998 bis 2001**
- **Zweite Phase testbasierter Entwicklung – ab 2002**
- **Ausblick und Resümee - ab 2004**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was ist ein Application Framework?



- **Objektorientierte Klassenbibliothek**
- **„Hauptprogramm“ liegt in der Bibliothek**
  - Applikationsarchitektur liegt im Framework
  - Konstruktion von Applikationsfamilien
- **Hollywood-Prinzip: Don‘t call us, we call you!**
  - Kontrollfluss vom Framework zu Applikationskomponenten
- **Erweiterung und Konfiguration**
  - Applikationen bilden Unterklassen von Frameworkkomponenten
  - Applikationskomponenten werden per Konfiguration mit dem Framework zur Applikation verknüpft.
  - einfache Applikationen können allein durch Konfiguration erstellt werden (z.B. mittels sog. Wizards)
- **Beispiele:**
  - Smalltalk Klassenbibliothek
  - Java: z.B. Servlet Engine, Application Server
  - C++: ET++, MFC, yafac (das angesprochene)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

AF: Produktfamilien und Reuse

- **Application Frameworks dienen der Gestaltung von Produktfamilien**
  - gleichartige Anwendungen in gemeinsamen Kontext
  - Wiederverwendung gemeinsamer Infrastruktur
  - „Delta"-Entwicklung, hoher Wiederverwendungsgrad
- **Enstehung oft evolutionär**
  - „Reissbrett-Entwurf" von Application Frameworks führt oft zu unnötiger Komplexität und „Design auf Vorrat" (YAGNI – you ain't gonna need it)
  - Mindestens 2 konkrete Applikationen sollten die Ausgangsbasis bieten
  - Herausfaktorisieren der Gemeinsamkeiten von Applikationen führt zu einem Application Framework
- **Wiederverwendung von Applikationsarchitektur und Infrastruktur**
  - für eine Familie von Applikationen

➢ **Hoher Qualitätsanspruch an das Framework**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was ist Testbasierte Entwicklung?

Automatisierte Tests stellen die Software Qualität sicher

- **Unit Testing**
  - für jede Komponente/Klasse/Methode, die eine nicht-triviale Implementerung hat werden eine oder mehrere Testmethode implementiert
  - Die Tests sollten isoliert ablaufen. Benötigte Umfeld-Objekte werden mittels sog. „mock-objects" simuliert um Abhängigkeiten zu vermeiden.
  - Oft werden Tests für Grenzfälle und das Normalverhalten erstellt.
  - Überprüfen der **technischen** Qualität der Implementierung
- **Functional Testing**
  - Relevantes externes Verhalten wird auf Systemebene getestet.
  - Testbarkeit durch Isolation von Teilsystemen mittels mock-objects verbesserbar.
  - Überprüfen der **fachlichen** Qualität
- **Idealerweise: Test-First**
  - Zuerst Test und Schnittstelle realisieren, dann implementieren

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Wie sieht ein Testcase aus?

- **Ausschnitt aus Tests für eine Bitset Implementierung**

```
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));
...
```

  - Unschön: zu viel in einem Test getestet
  - Praktisch: Hilfsobjekte nur einmal nötig

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Essentielle Infrastruktur

- **Entwicklungsumgebung**
  - bisher: SNiFF+ (WindRiver), moderner: Eclipse
- **Repository**
  - cvs: gratis, ausreichend, überall verfügbar
- **Workspaces**
  - pro Entwickler/Projekt, basierend auf Repository
- **Daily Build**
  - auf Basis des Repository via cron-job nachts,
  - Mail-Benachrichtigung
- **„Sharpen your Tools“**
  - Keine Angst vor scripts zum „hausgebrauch“
  - alles automatisieren was manuell mehr als einen Arbeitsschritt braucht
- **Kommunikation zwischen Entwicklern**
  - WikiWeb für persistente Infos
  - automatische Mails bei Check-Ins / Build-Fehlern
  - Whiteboards für Designdiskussion

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Das Application Framework

- **C++ Application Framework für Server Applikationen**
  - u.a. Web Server, Internet Banking, Web-To-Host, Web Applikationen
  - Produkt: Frontdoor – Web/FTP Protection Reverse Proxy mit SSO
- **Infrastruktur = Basisklassen**
  - String, Anything, (SSL-)SocketStream, MmapStream, SysLog, Tracing
- **Framework für Protokollhandling, Multi-Threading**
  - Leader-Follower Thread Pool, Acceptor, RequestProcessor, Dispatcher
- **Framework für Request Handling**
  - Session, SessionHandler, Context, Role, Page, Action
- **Framework für Output Generierung**
  - Renderer (HTML, XML, recursive scripting)
- **Framework für Data Access**
  - DataAccess, ParameterMapper, ResultMapper, DataAccessImpl
- **Framework für Initialisierung und Applikationsstart**
  - AppBooter, Module, Application, Server
- **Testframework**
  - Unit Testing, Configured Tests, Performance Tests

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Probleme anno 1997

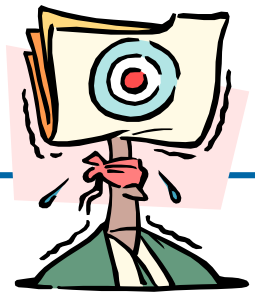
- **Erste Framework Applikationen erfolgreich erstellt**

aber

- **Entwickler halten sich „eigene“ Frameworkversionen vorrätig**
  - lokale Varianten weil bei Kunden kein Zugriff aufs Repository
  - projektübergreifender Konsolidierungsaufwand gescheut
  - „hacks“ am Framework um Schwächen zu umgehen
  - Individuelle Präferenzen von Entwickler „Primadonnen“
- **Korrektur von Kernabstraktionen und Infrastruktur sinnvoll**
  - Erfahrung aus dem Applikationen zeigte Verbesserungspotentiale
  - Rückintegration von Applikationscode ins Kernframework
  - Kommunikationsdefizit in Entwicklergemeinschaft

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Ursachen



- **Furcht vor Änderungen am Framework**
  - Risiko: Destabilisierung existierenden Codes
  - Risiko: Change Propagation in „fertige“ Applikationen
  - Risiko: Mehraufwand für wiederholtes Testen
  - Risiken sind schwer einzuschätzen
- **Qualitätsmängel im Framework**
  - FIXME Kommentare: quick hacks, trial and error
  - Verwendung von C-legacy considered harmful: sscanf, vprintf, char *
  - Teilweise unsauberes Verhalten im Multi-threaded Fall
  - Teilweise unsauberes Memory Management
- **projektübergreifender Aufwand schwer zu kalkulieren**
  - Verzögerung von Verbesserungen hinein in „Leerlaufphasen“
  - individuelle Projektkalkulation nicht Frameworktauglich
  - Führungswille „pro Framework“ notwendig

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Änderungen erforderten sehr viel Mut

- **und Angstschweiss, dass etwas schief geht**
- **Beispiel: Elimination von char * durch eine String Klasse**
  - schlechte Erfahrung mit String Klassen hatte dazu geführt, dass man auf eine solche ursprünglich verzichtet hatte
  - C-Erfahrung einiger Entwickler ebenso

  aber
  - sscanf() und sprintf() considered extremely harmful
  - char arrays als buffer extrem gefährlich
    - Overflow Problematik
- **Mut, Überzeugungsarbeit und persönliches Vormachen waren notwendig**
- **Schnittstellen wurden teilweise inkompatibel**
- **Aber: riskante Lösungen wurden reduziert**
- **String Klasse erlaubte Optimierung des Memory Mgmt**

Vision 1998

1998: Demo von Test-first Programming von Kent Beck und Erich Gamma

- „Das müssten wir auch machen!“

- **Wenn das Framework mit automatischen Tests abgestützt wäre, dann sollten Änderungen am Framework ohne Probleme machbar sein.**
- **Technische Varianten könnten gegeneinander abgewägt werden.**
- **Technische Verbesserungen könnten idealerweise ohne Implikation auf Applikationen erfolgen.**
- **Impakt-Abschätzung von (Schnittstellen-) Änderungen wäre möglich.**
- **Daily Build mit automatischen Tests verbinden.**

Lösung: Automatische Tests?

Vorgehen:

- **Adaption von CPPUnit einem Unit-Testing Framework auf die Bedürfnisse**
  - einfacheres (portableres) C++ (keine Templates, keine Exceptions)
  - Macro-basierte Registrierung von Tests (keine Reflection)
- **Tests für Infrastrukturklassen**
  - Potentielle Qualitätsmängel in Grenzfällen gefunden
  - Re-Implementierung von internen Strukturen zur Optimierung möglich
- **Tests für Kernabstraktionen des Frameworks**
  - Default-Implementierungen abgesichert
  - Schwierigkeiten im Grenzbereich zu Funktionstest
  - Nicht alle Klassen isoliert testbar
  - Erweiterung einiger Kernklassen (z.B. Request Context) für bessere Testbarkeit
- **Erweiterung des Testframeworks für spezielle Frameworkklassen mit vielen abgeleiteten Klassen**
  - konfigurierbare Tests
  - konfigurierbare Tests für Renderer, Actions, DataAccess

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Testframework für Unit Tests

Beispiele für definierte Makros

- **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)**
  - dito with a specific message
- **jeweils pro Datentyp ein assertEqualXXX(expected,actual)**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was ist speziell?

- **Daily Build mit automatischen Tests auf verschiedenen Plattformen**
  - Solaris 2.6, 2.7, Linux, jeweils im debug und optimierten Mode (+Windows)
- **konfigurierbare Tests / Test Scripting**
  - Parametervariation ohne Code-Duplikation
  - Schnelle Tests von Parametervarianten
  - Bei Fehlern in Applikationskontext lassen sich teilweise ohne code&compile entsprechende Testscenarien einbauen
- **end-to-end Tests vs. Unit Tests**
  - Frameworks bieten durch ihre konzertierten Mechanismen eine Hürde für einfache Testbarkeit.
  - Manche Klassen hängen praktisch vom ganzen Framework ab und lassen sich nur nach einem „boot-strap“ einer umfassenden Applikation gut testen
- **Performance Tests**
  - Timing von Tests ist sinnvoll (auch um Gesamttestzeit zu optimieren)
- **Lasttests**
  - brauchen noch etwas mehr Infrastruktur um sinnvoll zu sein, lassen sich praktisch kaum in automatischen Testablauf einbinden

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Beispiel für konfigurierten Test / Scripting

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

}
```

Testname

Parameter

Klasse

Script / TestObjekt

Resultat

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was hat es gebracht? (1)

- **Stabilität**
  - Grenzfälle besser abgedeckt
    - z.b. 0, 1, viele, ganz viele, negativ ?
- **Refactoring im Framework wurde möglich**
  - z.B. Reduktion von unnötigem Locking, optimiertes Memory Mgmt
- **Portabilität auf andere Plattformen abgesichert**
  - AIX, Windows NT, Teilbereiche auf exotisches IBM Host OS (TPF)

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Was hat es gebracht? (2)

- **Austauschbarkeit von (ungünstigen) Implementierungen**
  - Elimination von C-Legacy und FIXMEs
- **Besseres Interface Design von neuen Dingen**
  - Test-first und Testability sind gute Ratgeber fürs Schnittstellendesign
- **Einfachere Flexibilisierung**
- **Vertrauen in „fremden“ Code bei Entwicklern gestiegen**
  - Konsolidierung von Code
  - reduziertes Risiko, bzw. Risiko einschätzbar, indem man Testcase schreibt, bevor man Änderung macht

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Beispiel: Thread-local Memory Management

- **Idee für Optimierung des Memory Management**
  - jeder Thread hat eigenen Memory Pool für „transiente“ Objekte
  - kein Locking für Speicheranforderung notwendig
- **Unit Tests für neues Memory Management**
  - stellen Funktion sicher
  - API Design Hilfe
- **Unit Tests für Klassen, die das neue MM benutzen**
  - stellen sicher, dass auch diese Klassen weiterhin funktionieren
  - wirken auch bei eher indirekten Abhängigkeiten

➢ **fundamentale Änderung in der Infrastruktur möglich ohne existierenden Code zu beeinträchtigen**

➢ **Performance Gewinn bei Last auf Servern spürbar**

➢ **Vorhersagbarkeit der Resourcennutzung und Stabilität erhöht**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Verbleibende Schwierigkeiten (V1)

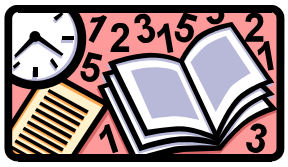
anno 200[01]:

- **Applikationscode ohne Tests**
  - insbesondere von Entwicklern bei Kunden
  - Keine Erfahrung in Testbasierter Entwicklung
  - Fixierung auf bestimmten Framework Release
  - Angst vor inkompatiblen Upgrades
  - Refactoring mit minimalen Schnittstellenänderungen wurde gescheut
  - Auswirkung von Änderungen auf Applikationen immer noch schwer einschätzbar
- **„Dunkle Ecken“ im Framework**
  - ursprüngliches Design nicht mit Unit-Tests und Testability
  - teilweise unnötige Semantikerhaltung bei Refactorings
  - ungenutzte Features und Semantik wurde beibehalten und durch entsprechende Tests manifestiert

- **Tests laufen zu langsam und werden zu selten gestartet**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Lessons Learned (V1)



- **Testbasiert zu entwickeln braucht Zeit um es zu lernen**
  - Schreiben von Tests ist programmieren
  - Gute Tests sind leicht verständlich, wie guter Code
  - Tests brauchen Refactoring um gut zu werden/zu bleiben
  - damals keine Literatur über Unit Testing und wenig Erfahrung
- **Formulieren von Tests (auch händische) fördert das Verständnis für die Requirements**
  - was man nicht testen kann ist oft auch nicht relevant
  - „man kann alles relevante testen“ stimmt fast
  - fast alle Tests lassen sich automatisieren
  - vom Use Case zum Test Case
- **Tests zu programmieren fördert das Schnittstellendesign**
  - Entwickler ist „Opfer“ seiner eigenen Designentscheidungen
  - Einfacheres Design wird favorisiert, weil es einfacher zu testen ist
- **Unschöne Tests sind oft Zeichen von schlechtem Design im getesteten Code oder schlechter Testability**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Lösung: Tests auch für Applikationen

- **Einbindung der Applikationen in Daily Build Prozess**
  - Quellcode-Inkompatibilitäten werden sofort erkannt
- **Ausbildung der Applikationsentwickler im Schreiben von Tests**
  - Coaching erforderlich
- **Beim Erstellen neuer Releases von Applikationen Tests implementiert**
  - Nachtesten vorhandenem Codes (nur teilweise)
  - Für Änderungen und Neues immer Tests
- **Gemeinsames Repository über Team- und Firmengrenzen hinweg**
  - DailyBuild jeweils in eigener Infrastruktur komplett
  - Konfigurationen werden mittels scripts automatisch „gepatcht“
  - Skalierbarkeit des Ansatzes leider begrenzt

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Situation heute

- **Impact Analyse von wünschenswerten Änderungen einfach, da (fast) aller abhängiger Code im Repository**
  - kann wieder innovativ bei Änderungen sein
- **Jedes Deployment einer Applikation wird mit einem Tag im Repository markiert und bei kleinen Patches/Fixes auf dem entsprechenden Branch gearbeitet**
  - nicht nur Vorteile, arbeiten mit cvs-branches erfordert Sorgfalt
  - neue Releases werden auf aktuellem Stand entwickelt
- **Refactorings mit Schnittstellenänderung sind wieder möglich**
  - Änderungen an Applikationscode werden sofort erkannt und korrigiert
  - Deployment aber nur bei Bedarf bzw. neuem Release der Applikation
- **Grundlegende aber wichtige Refactorings können immer noch aufwendig sein**
  - Designfehler rächen sich irgendwann
  - Test-first von anfang an hätte helfen können
- **Tests immer noch relativ aufwändig**
  - Nightly Builds statt continuous Build (6 Plattformen)
  - False Positives trügen oft

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Falsche Fehler (False Positives)

- **Ursachen:**
  - Timing Abhängigkeiten (Sekundensprung)
  - Externe Abhängigkeiten (Server nicht Verfügbar)
    - DNS - Lookup, Socket Tests jenseits „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 sollten möglichst nicht von externen unkontrollierbaren Resourcen abhängen**
- **Kontrollierbare externe Resourcen (z.B. DB) vor jedem Test automatisch in definierten Initialzustand bringen**
- **Existierenden Code nachträglich mit Testfällen abzusichern ist aufwändiger als gleich Testbasiert zu entwickeln**
  - manchen schlecht testbaren Code implementiert man besser neu, indem man zuerst die Tests schreibt
- **Refactoring ohne automatische Tests ist zu gefährlich**
- **Bei Fehlern schreibt man einen Tests statt zu debuggen**
  - Fehler reproduzieren
  - Hypothesen über das Verhalten des Codes verifizieren
  - Debugger in Rente schicken oder verbieten

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Vision und Wünsche heute

- **Automatische Tests in Lichtgeschwindigkeit**
  - immer alles laufen lassen, keine manuelle Arbeit
  - jeder check-in triggert tests, wenn Fehler: ablehnen
- **Wunschtraum bleibt, dass immer auf aktuellste Version benutzt wird und automatisch ein deployed, falls alle Tests OK sind**
  - kein manuelles Testen (inkl. Systemtest)
- **Kunden schätzen die Qualität von Software vor der Auftragsvergabe und nicht erst nach erfolgreichem Betrieb und Änderungen**
  - wirtschaftlicher Erfolg ? Awareness am Markt ?
- **Kunden / Auftraggeber in die Lage versetzen selbständig automatische funktionale Tests in Ihrer Sprache zu formulieren**
  - Ansätze existieren:
    - **FIT (fit.c2.com) und fitnesse (fitnesse.org)**

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Essentielles - Pragmatisches Programmieren

- **Automatische Tests schreiben und laufen lassen**
  - Refactoring wird ermöglicht
  - Schnittstellendesign und Testbarkeit verbessert
  - Requirements besser verstanden
  - nach jedem check-in 100% OK aller Tests
- **Repository macht Mut etwas auszuprobieren**
  - schnelle Arbeitsweise möglich, da zurückgehen möglich
  - Risikoloses Probieren von Implementierungsalternativen
  - Wenig Integrationsprobleme, wenn regelmässig und oft eingecheckt wird
- **Repetitive Aufgaben automatisieren**
  - Deployment von Applikationen
  - Aufsetzen von Test-Initialwerten
  - cron-Jobs für Tests
  - mail-Benachrichtigung bei Check-Ins oder Test-Failures

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Hat es sich gelohnt? (1) Ja und Jein.

- **Aus Entwicklerperspektive ganz klar JA**
  - besserer Code, besseres Design, Refactoring
  - weniger „Debugging Stress“
  - einfacher neue Ideen einzubringen, sustainable code
- **Aus Kundenperspektive: Jein**
  - Testbasierte Entwicklung erscheint teuer und erfordert Lernaufwand bei eigenen Entwicklern
  - Ergebnisqualität fast zu gut, insbesondere bei Requirement Änderungen

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Hat es sich gelohnt? (2) Ja und Nein.

- **Aus Dienstleisterperspektive: heute Nein**
  - zufriedene Kunden kündigten Support & Wartungsverträge, weil nicht mehr notwendig
  - Qualität wird zwar geschätzt aber ihr Preis nicht gezahlt
  - Schlechtere Softwarequalität führt zu mehr Kundenkontakten und besserer Kundenbindung
    - (neue Projekte mit Potential erkennen!)
  - Neue Kunden sind schwer von Vorteilen zu überzeugen, insbesondere da der Markt härter geworden ist
  - Frameworkbasierte Entwicklung und Frameworks sind schwer zu verkaufen
- **deshalb: Framework als Open Source in 2005**
  - stay tuned...

Literatur

- Kent Beck: Test-driven Development by Example
- 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

Anzeige :-)

- Pattern-oriented Software Architecture:
  A System of Patterns
  (Buschmann,Meunier,Rohnert,Sommerlad,Stal)
- Security Patterns (2005, M. Schumacher et al)
- Fragen zu Patterns, etc.: psommerlad@hispeed.ch