INSTITUTE
FOR
SOFTWARE

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INFORMATIK

Automated Unit Testing
A Practitioner's and Teacher's Perspective

Prof. Peter Sommerlad

HSR - Hochschule für Technik Rapperswil
Institute for Software

Oberseestraße 10, CH-8640 Rapperswil

peter.sommerlad@hsr.ch

http://ifs.hsr.ch

http://wiki.hsr.ch/PeterSommerlad

Peter Sommerlad

peter.sommerlad@hsr.ch

INSTITUTE FOR SOFTWARE

- **Work Areas**
  - Refactoring Tools (C++,Ruby, Python,...) for Eclipse
  - **Decremental Development (make SW 10% its size!)**
  - Modern Software Engineering
  - Patterns
    - Pattern-oriented Software Architecture (POSA)
    - Security Patterns
- **Background**
  - Diplom-Informatiker (Univ. Frankfurt/M)
  - Siemens Corporate Research - Munich
  - itopia corporate information technology, Zurich (Partner)
  - Professor for Software HSR Rapperswil, Head Institute for Software

Credo:

- **People create Software**
  - communication
  - feedback
  - courage
- **Experience through Practice**
  - programming is a trade
  - Patterns encapsulate practical experience
- **Pragmatic Programming**
  - test-driven development
  - automated development
  - Simplicity: fight complexity

Is that Testing?

INSTITUTE FOR SOFTWARE

- **“it compiles!”**
  - no syntax error detected by compiler
- **“it runs!”**
  - program can be started
- **“it doesn’t crash”**
  - … immediately with useful input
- **“it runs even with random input”**
  - the cat jumped on the keyboard
- **“it creates a correct result”**
  - a single use case is working with a single reasonable input

What is Testing?

- **All on the previous slide, but much more!**
- **Manual Testing**
  - sometimes useful and needed
    - UI testing, usability testing, user testing with a plan
  - but automation is much better!
    - no ad-hoc testing!
- **Automated Testing** ← **Today's topic**
  - unit tests
  - functional tests
  - integration, load and performance tests
  - code quality tests (lint, compiler, code checkers)

What does Testing mean?

- **You want a correctness guarantee!**
  - how do you define "correctness"?
  - "correctness" against what specification?
  - what kind of guarantee?
    - 6 months, 2 years, lifetime?
- **Alternatives to Testing?**
  - code reviews
  - walkthroughs
  - inspection
  - mathematical proofs of correctness
    - hard, hard to specify understandable specifications
    - but, can be used to construct code with proven correctness

Unit Testing

INSTITUTE FOR SOFTWARE

- **Is not "Testing" in the classic sense:**

> Program testing can be used to show the presence of bugs, but never to show their absence! - *E.W. Dijkstra*

But

- **Is Built-In Quality Assurance**
- **Allows Regression Testing**
- **Enables Refactoring**
- **Is Change Insurance**
- **Improves Built Automation**

Structure of a typical Unit Testing Framework

INSTITUTE FOR SOFTWARE

- **Test Assertion / Check statement**
  - used in
- **Test (member-)function / method**
  - defined in
- **Test Case subclass bundling tests**
  - its objects contained in
- **Test Suite collecting test objects**
  - executed by
- **Test Runner (often in a main() function)**
  - delivers result
- **OK or Failure**

Software Engineering without Test Automation

INSTITUTE FOR SOFTWARE

- **repeated manual testing**
  - error prone (different testing each time)
  - inefficient and slow
  - expensive man power
- **results in fear of changing existing code**
  - requires courage (sometimes too much)
  - large amounts of unused or bad code remain

- **assumed risks:**
  - destabilizing of existing code
  - change propagation into „done" parts
  - increased and repeated manual test efforts

- **Advantages**
  - repeatability - regression
    - insurance for change, portability, extension
    - no (or very low) cost for re-testing
  - well-defined specification given by tests
    - test-code is program code with well-defined semantics
  - repeatability, repeatability, repeatability, ...
- **Drawbacks**
  - need to write and maintain also test code
    - tests also require refactoring
  - test code is program code
    - is the right thing tested? (instead of implemented?)

Why Automated Testing?

INSTITUTE FOR SOFTWARE

- **better stability**
  - coverage of corner cases
- **Refactoring enabled**
  - there is more than your IDE's Refactoring menu
- **portability to other platforms ensured**
- **better interface design of new code**
  - test-first and testability are good guides for design:
    - lower coupling, higher cohesion
    - developer suffers interface design immediately
- **developer trust in "foreign" code improved**
  - control risk by writing tests before changing code

When?

- **Become "test-infected". Once you are used to unit testing your code, you get addicted.**
  - That's a fact I observed many times.
  - I regret every piece of code I want to change where I don't have tests for, you might also.
- **Write your tests close to writing your code!**
  - Some say: Test-First or Test-Driven Design (TDD)
  - modern: Behavior-Driven Design (BDD)
  - Retrofitting existing code with tests will show you its design deficiencies
    - hard to write tests -> entangled design, too complex
    - easy to write tests -> orthogonal design, simpler
- **At least write tests before you change code!**

Practitioner's Experience Time and Understanding

INSTITUTE FOR SOFTWARE

- **you need time to learn test-based development**
  - writing tests is programming
  - good tests are as comprehensible as good code
  - tests require Refactoring to stay in good shape
- **thinking about and writing tests down improves understanding of requirements**
  - what you can not or don't want to test is often not relevant to create
    - ➢ know when you are done!
  - "you can test everything that is relevant" is almost always true (there are exceptions)
  - most tests can and should be automated
  - use Use Cases to derive Test Cases

Practitioner's Experience Observations

INSTITUTE FOR SOFTWARE

- **programming tests (first) improves interface design dramatically**
  - developer is "victim" of own design decisions
  - simpler (less complex) design is favored, because of its better testability
- **ugly tests are an indication of bad design or lack of testability of the code**
  - Refactor, because of high coupling
- **write tests demonstrating bugs instead of debugging**
  - reproduce errors
  - verify hypothesis about code behavior
  - retire your debugger or forbid its manual use

Practitioner's Experience Take Care

- **tests shouldn't depend on external resources outside the control of the test runner**
  - o bring external resources under your control (e.g., DB) in a consistent initial state or de-couple
  - o external resources might slow down your tests!
- **writing tests for existing code is harder**
  - o sometimes testing against a façade can help
- **tests should run as fast as possible**
  - o ms instead of minutes, run tests often
- **don't test the platform**
  - o unless you don't trust it

Practitioner's Experience Customer Perspective

INSTITUTE FOR SOFTWARE

- **customers honor software quality, but**
  - how to demonstrate higher quality and changeability before purchasing and not just after successful operation and changes?
  - how to get economic success and market awareness of test automation?
- **test-based development seems to be more expensive**
  - used to "debugging phase" after delivery
- **solution quality almost too good, especially when requirements change**
  - customers quickly learn that changes are easy
  - economic viability can be critical

Practitioner's Experience Market

INSTITUTE FOR SOFTWARE

- **economic viability can be critical**
- **happy customers canceled support contracts, because they considered them unnecessary**
  - quality is valued but its price not payed
- **worse quality provides more customer contacts and better customer binding**
  - easier to learn about new and potential projects!
  - meaning more business with existing clients
- **new customers are hard to convince about advantages**
  - especially when the service market grew tougher
  - today, situation may be better again

Test Automation as a Teacher

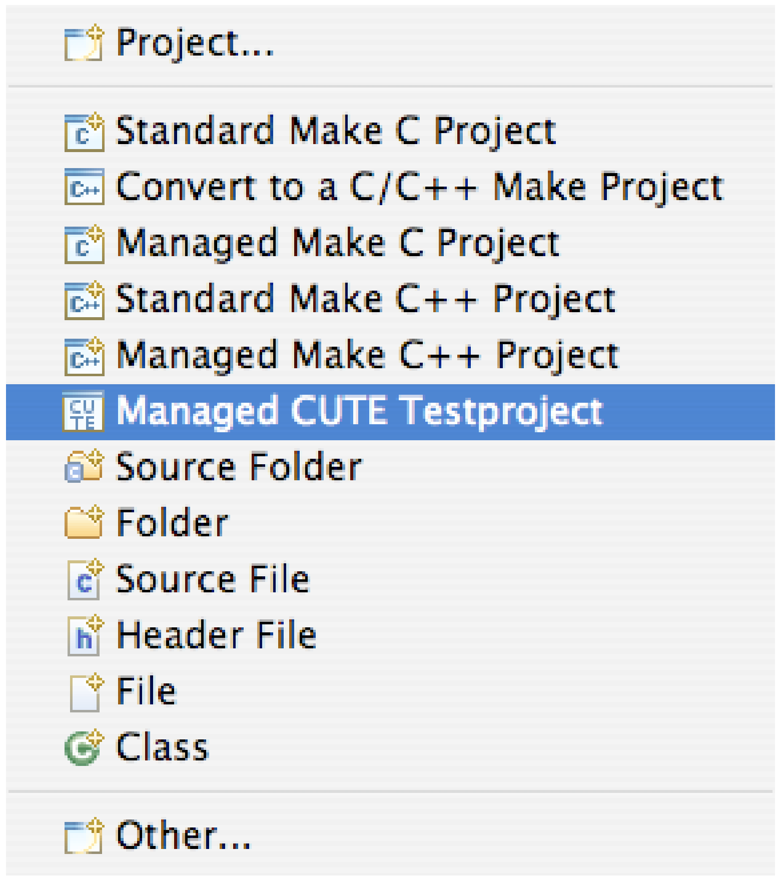
INSTITUTE
FOR
SOFTWARE

- **since 2004 - HSR Rapperswil**
- **adaptation of curriculum regarding change to bachelors degree starting in 2005**
  - o influence content of existing courses
    - ➢ more modern agile and pragmatic approaches
  - o new courses
    - ➢ emphasis on software engineering
    - ➢ less technology driven
- **new C++ unit testing framework CUTE**
  - o with Eclipse CDT UT plug-in
  - o for teaching C++ test-driven
  - o easier to use than CPPUnit and its derivates

Green-Bar for C++
CUTE Plug-In for Eclipse

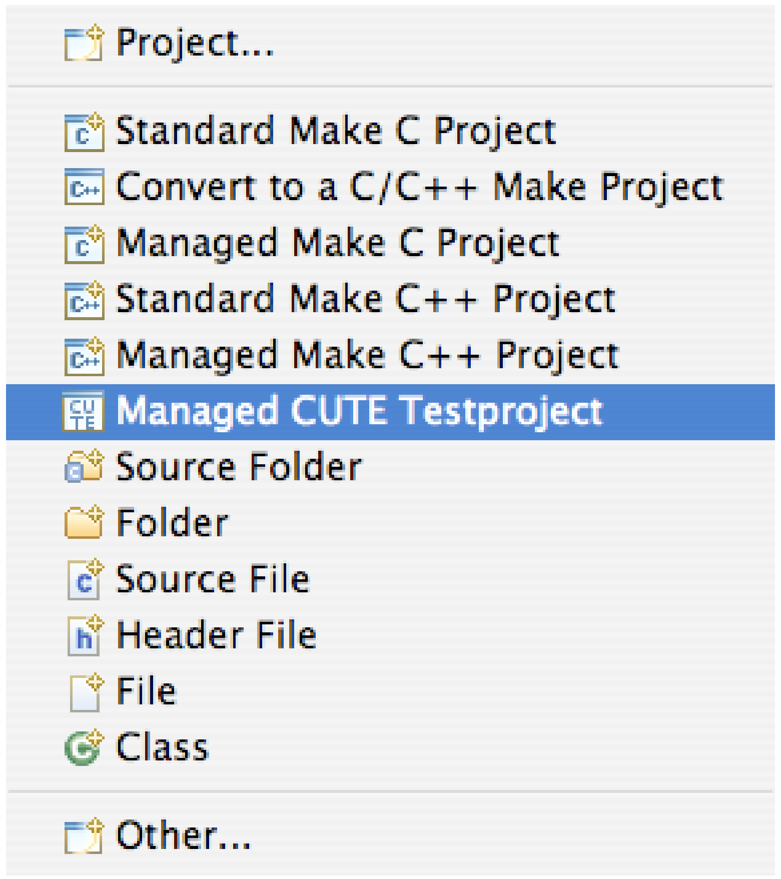
Green-Bar for C++ CUTE Plug-In for Eclipse

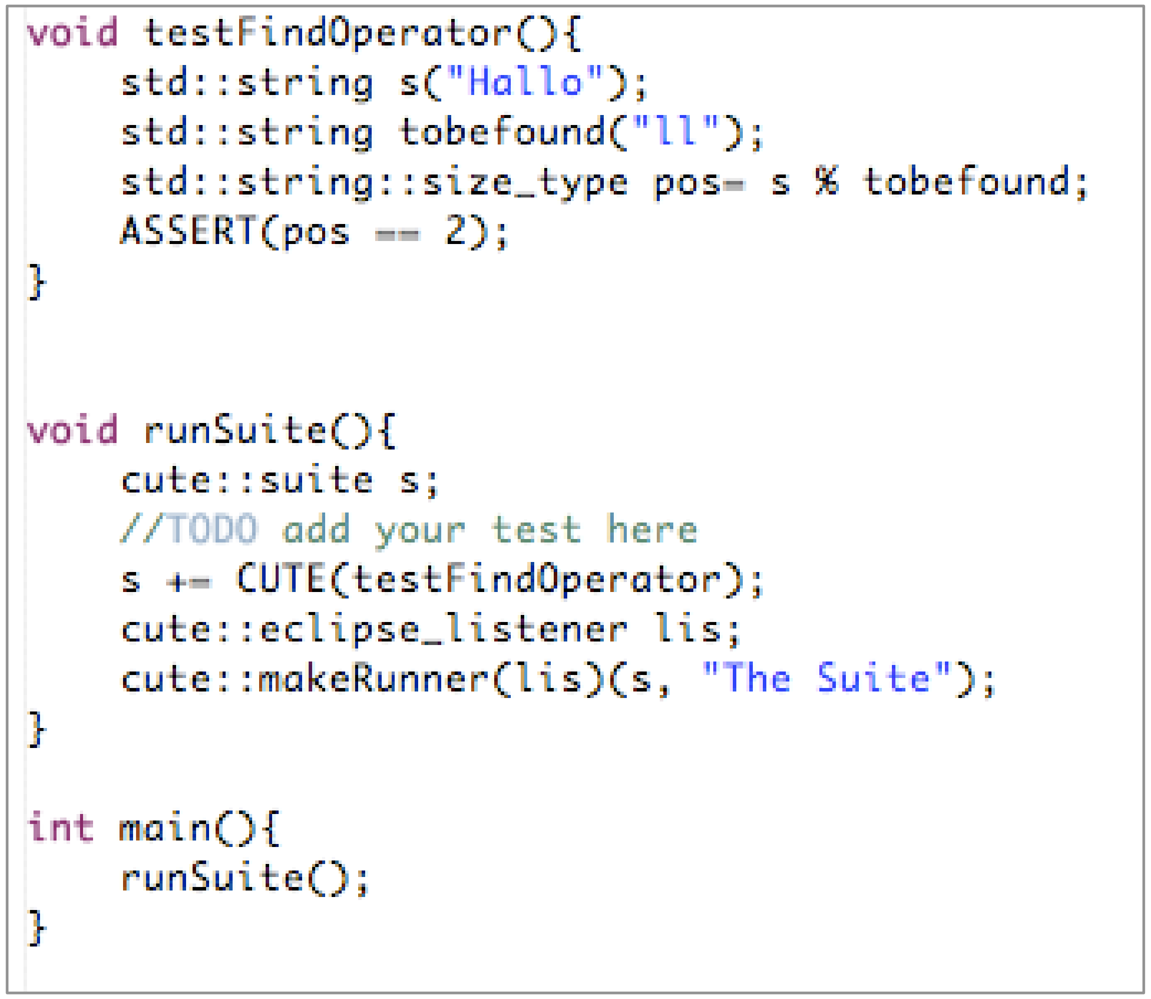
INSTITUTE FOR SOFTWARE



Green-Bar for C++ CUTE Plug-In for Eclipse

INSTITUTE FOR SOFTWARE





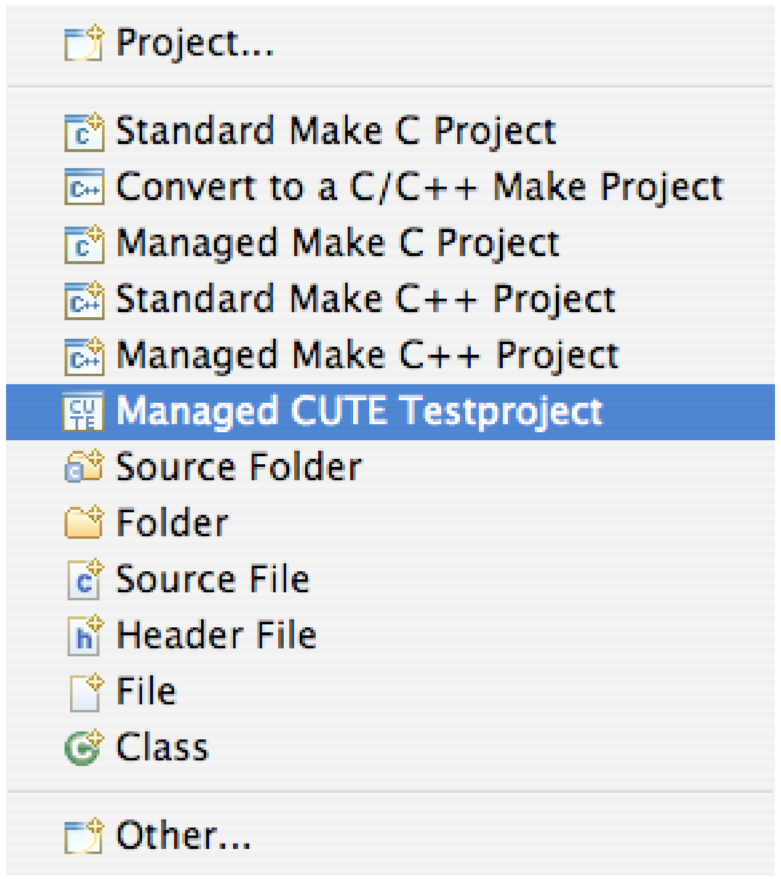
```
void testFindOperator(){
    std::string s("Hallo");
    std::string tobefound("ll");
    std::string::size_type pos= s % tobefound;
    ASSERT(pos == 2);
}


void runSuite(){
    cute::suite s;
    //TODO add your test here
    s += CUTE(testFindOperator);
    cute::eclipse_listener lis;
    cute::makeRunner(lis)(s, "The Suite");
}

int main(){
    runSuite();
}
```

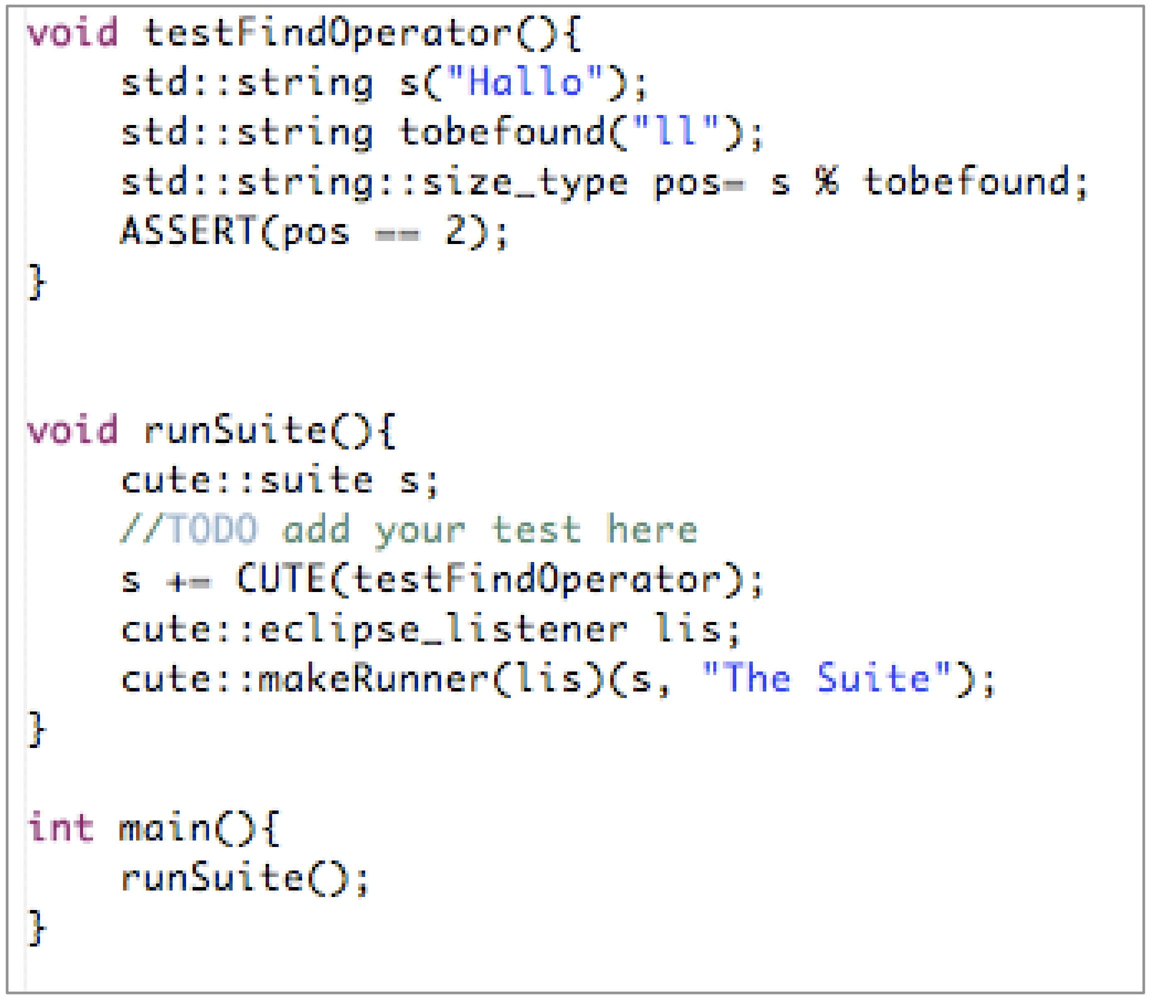
Green-Bar for C++ CUTE Plug-In for Eclipse

INSTITUTE FOR SOFTWARE


Project...

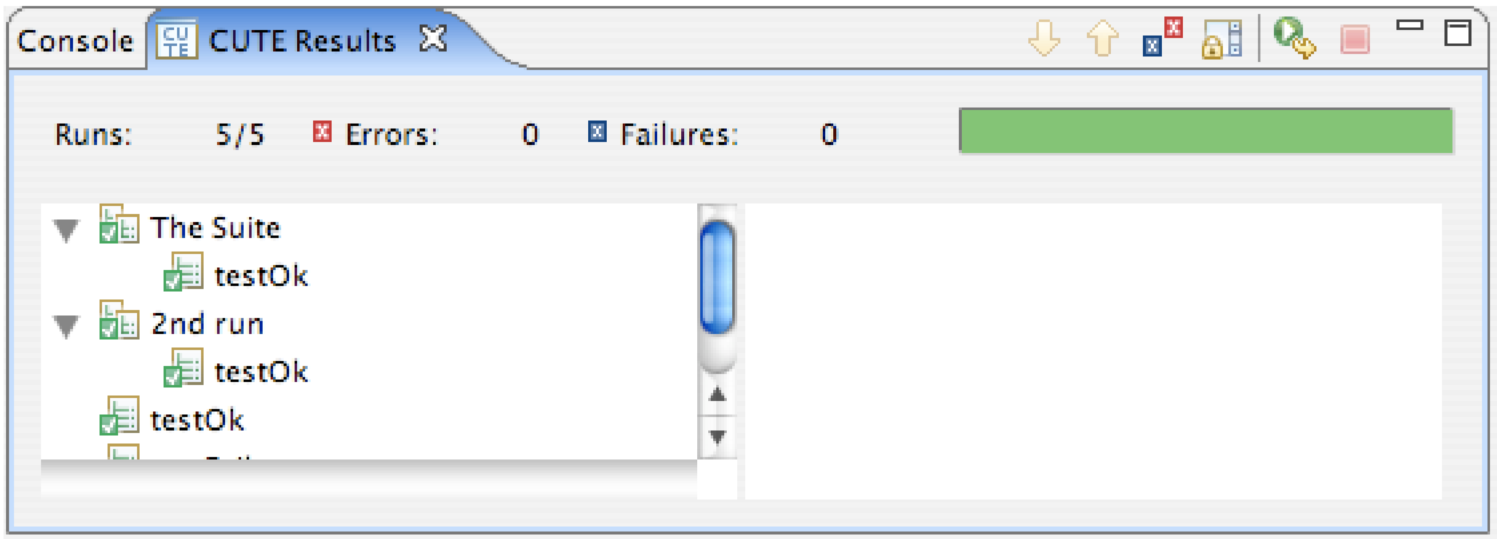
Standard Make C Project
Convert to a C/C++ Make Project
Managed Make C Project
Standard Make C++ Project
Managed Make C++ Project
Managed CUTE Testproject
Source Folder
Folder
Source File
Header File
File
Class

Other...


```
void testFindOperator(){
    std::string s("Hallo");
    std::string tobefound("ll");
    std::string::size_type pos= s % tobefound;
    ASSERT(pos == 2);
}


void runSuite(){
    cute::suite s;
    //TODO add your test here
    s += CUTE(testFindOperator);
    cute::eclipse_listener lis;
    cute::makeRunner(lis)(s, "The Suite");
}

int main(){
    runSuite();
}
```


Console | CUTE Results

Runs: 5/5 Errors: 0 Failures: 0

The Suite
    testOk
2nd run
    testOk
testOk

Test Automation at HSR Modules

INSTITUTE FOR SOFTWARE

- **Prog1 - Java (1st semester)**
  - o next year: unit testing from the beginning
- **Prog3 - C++ (2nd semester)**
  - o in summer: CUTE with Eclipse plugin
- **SE1 - Software Engineering 1 (3rd semester)**
  - o introduction to Unit Testing in Java
  - o formal exercises, writing tests
- **SE2 - Software Engineering 2 (4th semester)**
  - o advanced automated testing
  - o Test-driven Design and Refactoring
  - o FIT and fitnesse
- **term projects and thesis projects (5-6 sem.)**

Teacher's Experience Student's Projects

INSTITUTE FOR SOFTWARE

- **Good students "get it" easier**
  - o need some up-front motivation
  - o soon get the benefits of safer refactoring
  - o write tests for understanding existing code
  - o accept code reviews and immediately refactor
  - o create own infrastructure for test automation
- **Students more in need of test automation don't**
  - o create worse, hard to test design
  - o high coupling gets in the way of testing
  - o in some cases successful in motivating writing tests and students kept it

Teacher's Experience Lectures and Exercises

- **Life Programming**
  - show writing and running tests
    - hard to set up, needs courage
  - Test-First Demo tended to be a bit boring
    - Roman Numbers converter
- **Writing Tests and Refactoring require exercise**
  - combine topics of testing and Refactoring
  - Refactoring challenge was motivating for students
  - teacher must review code and give feedback
- **Things to look out for at student's exercises**
  - platform tests -> testing compiler and libraries
  - test code duplication -> Refactoring test code

Teacher's Experience Colleagues

INSTITUTE FOR SOFTWARE

- **Not many of our staff have similar experience**
- **"network" professors' student projects are often software development**
  - much worse quality
  - hard to teach the colleagues (nobody has time!)
- **Students that worked on ongoing projects told me about the need for cleaning up the code**
  - do you have ideas what to do about it?
- **Inconsistent value system and opinions about software quality**
  - confusing for students

Test Automation Potential Dangers

- **programmers create useless/bad tests**
  - o just to follow the rules, but without value
  - o i.e., too many tests for the same functionality
  - o hard to refactor tests/code by not testing against an interface but the implementation
  - o only happy path testing
  - o wrong granularity of tests
  - o false positives ignored
- **test-first development can drive design to become procedural code, not objects**
  - o testing object interaction requires more set-up code to have all needed objects in place

Test Automation Summary

- **be pragmatic instead of dogmatic**
  - just asking for test automation is not enough
  - motivation and encouragement with feedback
- **learn “good design sense”**
  - good tests vs. bad tests, feedback
- **fast tests: run them often**
  - future: tests run with typing in an IDE, like the compiler today
- **unforgettable tests**
  - run them as part of automatic build
  - inform all stakeholders about results
  - no “broken windows” - always green bar

My Suggestions

INSTITUTE FOR SOFTWARE

- **Learn how to automate your tests tomorrow**
  - good literature available now
  - not only unit tests useful or to be automated
- **Teach others about test automation**
  - if not yet done, make it part of standard software engineering curriculum
  - requires practice not only theory and feedback by experienced person
- **Make automated tests part of your daily build**
  - or even better of your continuous build
  - it is unprofessional to create software without test automation today

Final Suggestion

INSTITUTE
FOR
SOFTWARE

- **Remember**
  - Only Code tells the Truth
- **Therefore**
- **Use Test Automation and Refactoring to**

Simplify your Code

- **Questions?**