

Testing Research Software Nonsense or Necessity

Unit tests in the scientific
software development
process

Contents

- Background
- Unit tests and reasons
- An application
- Testing: When, what, why and how
- Programmers wisdom

Forword

- Wrong people but spread the word
- Objective in life:
convince two people two use unit tests
=> Success so far:
0.75 achieved, 3.25 to go

Background

- Computer guy with interest in Machine Learning
- Masters in Computer Science, University of Würzburg, Germany
- 10 Years Research and Development at Siemens
 - Machine learning in automatization processes (Image&Signal processing, Quality Control, Robotics)
- 1.5 Years Development and Research at XXX
 - Text and data mining, Speech processing
- Now PhD student with a topic in bioinformatics

Unit test – What is it?

```
int factorial(int n) {  
    int fac = 1;  
    for(int i=1; i<=n; i++)  
        fac = fac*i;  
    return fac;  
}
```

```
void testFactorial() {  
    assertEquals(0, faculty(0));  
    assertEquals(1, faculty(1));  
    ...  
    assertEquals(0, faculty(-1));  
    assertEquals(3628800, faculty(10));  
}
```



fails



fails

Requirements

- short
- simple
- independent

Plenty of reasons not to test

- It's a terrible waste of time
- My boss doesn't care
- Tests are for losers
- I am a scientist. I don't make mistakes
- My research is so incredibly difficult and complex that it can't be tested
- ...and nobody is interested in my research results anyway
- Matlab works with matrices
- Everything in Python is so simple, you can't do it wrong
- I implement in C/C++ and the code works nicely even with bugs
- I work with Perl and, ehm...
- I wanna write papers not tests
- Testing does not guarantee that my code is 100% bug-free
- Revenge (I suffered, so everybody who wants to use my code has to suffer as well)
- We are all gonna die anyway

Just a few reasons to test

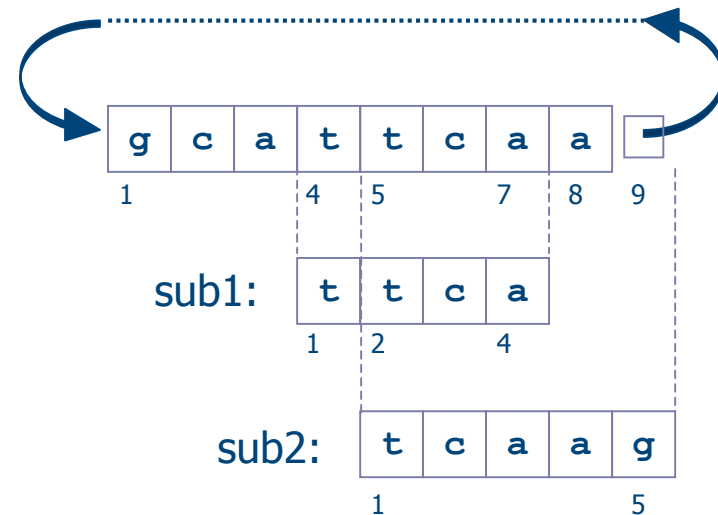
- Saves time – yes, it does!
- Improves quality of code, results and papers
- You can build on it. It scales
- More fun in life 😊 - very important

A "simple" problem, hmm

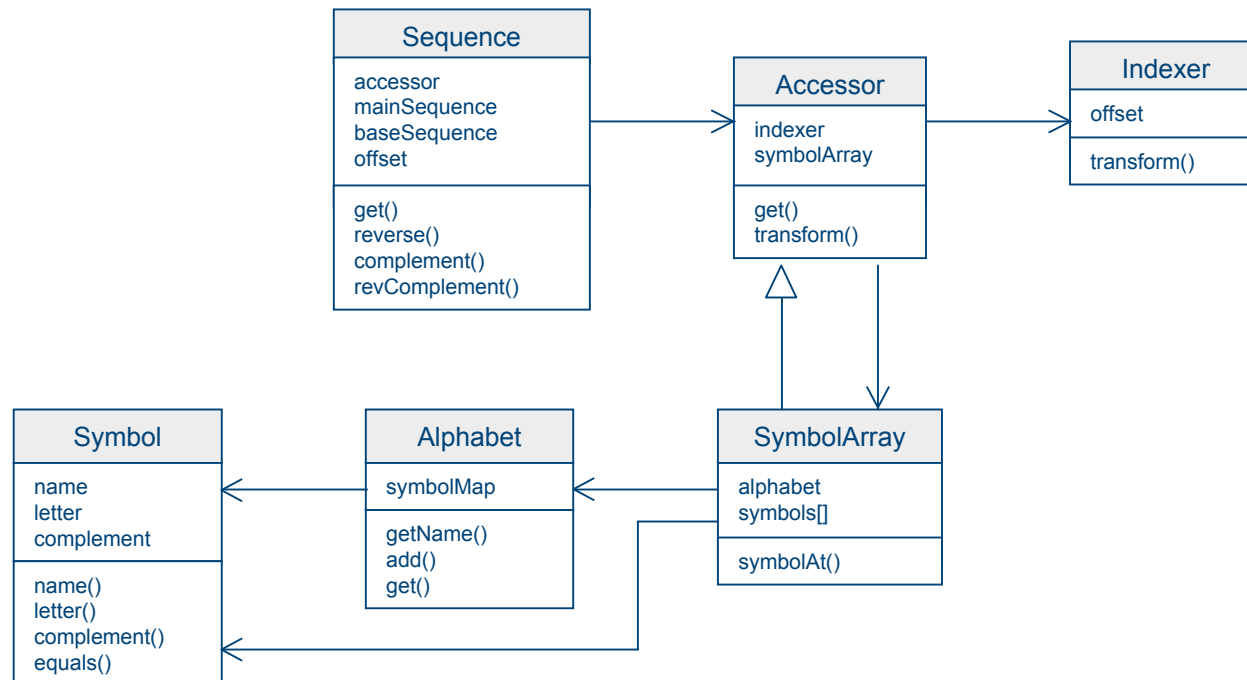
- Model of biological sequence data

atatgactacggatacgttatatca...

- Alphabet -> ambiguity symbols
- Strand -> reverse complement
- Circular or linear
- Features
- Sub-sequences
- Indexing

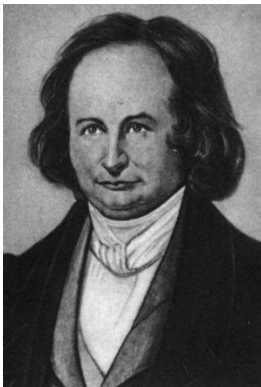


Class diagram



Jacobi

Java for computational biology



Karl Gustav Jacob Jacobi
1804 - 1851
German Mathematician

Library for biological sequence analysis

- Ease of use
- Smart indexing system
- Advanced pattern description
- **Tested**

But wait, there is more ...

Unit tests

Runs: 409/409 ❌ Errors: 0 ❌ Failures: 0



Automatic Test Generation

Continuous Testing

Test coverage

Package ^A	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	46	70% 1308/1856	79% 213/270	2,458
net.sourceforge.cobertura.ant	10	48% 152/314	60% 26/43	1,871
net.sourceforge.cobertura.check	3	0% 0/150	0% 0/27	2,429
net.sourceforge.cobertura.coveragedata	10	N/A	N/A	2,239
net.sourceforge.cobertura.instrument	6	82% 243/296	91% 41/45	2,538
net.sourceforge.cobertura.merge	1	86% 30/35	100% 8/8	5.5
net.sourceforge.cobertura.reporting	3	86% 115/134	100% 25/25	2,882
net.sourceforge.cobertura.reporting.html	4	88% 445/508	96% 59/72	4,308
net.sourceforge.cobertura.reporting.html.files	1	87% 39/45	100% 4/4	4.5
net.sourceforge.cobertura.reporting.xml	1	100% 122/122	100% 8/8	1,421
net.sourceforge.cobertura.util	8	63% 156/246	84% 32/38	3,067
someotherpackage	1	83% 5/6	N/A	1.2

Metrics

Metric	Total	Mean	Std. Dev.	Maximum
Number of Static Methods (avg/max per type)	71	0,245	0,483	4
Total Lines of Code	13405			
Method Lines of Code (avg/max per method)	9243	6,669	8,051	129
Number of Methods (avg/max per type)	1316	4,538	6,27	62
Number of Classes (avg/max per packageFragment)	290	8,529	9,974	46
Number of Packages	34			
Number of Interfaces (avg/max per packageFragment)	20	0,588	1,003	4
Afferent Coupling (avg/max per packageFragment)		27,853	54,678	250
Normalized Distance (avg/max per packageFragment)		0,437	0,311	1
Specialization Index (avg/max per type)		0,107	0,344	2,5
Instability (avg/max per packageFragment)		0,488	0,326	1
Number of Attributes (avg/max per type)	256	0,883	1,499	10
Weighted methods per Class (avg/max per type)	2186	7,538	11,448	109
Number of Overridden Methods (avg/max per type)	77	0,266	0,744	6
Number of Static Attributes (avg/max per type)	45	0,155	0,847	13
Nested Block Depth (avg/max per method)		1,158	0,493	5
Lack of Cohesion of Methods (avg/max per type)		0,09	0,208	0,856
McCabe Cyclomatic Complexity (avg/max per metho)		1,577	1,579	28
Number of Parameters (avg/max per method)		0,682	0,982	6
Abstractness (avg/max per packageFragment)		0,099	0,154	0,625
Efferent Coupling (avg/max per packageFragment)		7,676	8,761	44
Number of Children (avg/max per type)	97	0,334	1,342	13
Depth of Inheritance Tree (avg/max per type)		2,334	1,099	7

I can't test cos ...

- I don't use Java and have no JUnit lib
=> **there are Unit test libs for Perl, Python, Matlab, C/C++, ...**
 - my methods are
 - too big
 - too complex
 - don't know what to expect**=> indicates a serious (design) problem**
 - everytime I refactor it breaks my tests
=> that isn't refactoring
 - everytime I add functionality/redesign I have to rewrite many tests
=> test are not independent
- => the harder it is to test, the more important and rewarding it is**

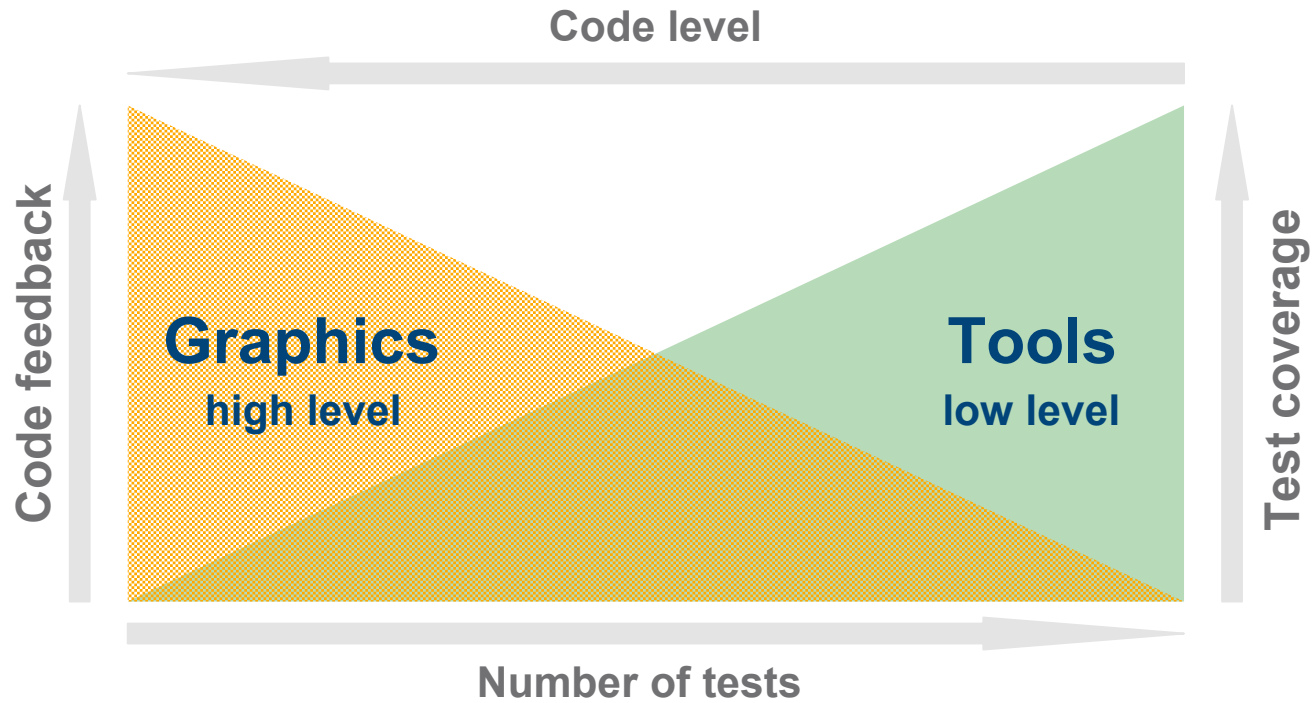
That's tough

- Machine learning algorithms
- Stochastic algorithms
- Hybrid software (e.g. diff. languages)
- User interfaces / graphical applications
- Multi-threaded software
- Distributed software

You don't have to, if ...

- it's a one time job
... but you never know
- it's tiny
... but things tend to grow
- it's a test/experiment
... but you want to trust it
- it's a graphical front end
... but think about nuclear power stations, aircrafts,...
- the test is as complex as the code
... but sometimes even Setters/Getters break

Test intensity spectrum



Two zones model

Research

- not tested
- highly volatile
- crappy
- high-level

Tools

- tested
- stable
- clean
- low-level

Unit tests – What's the use?

- test code
- allow for fast and save refactoring
- ensure extensibility and robustness
- document code
- serve as templates
- measure code quality

Unit tests save time !

- less debugging time
- less printf time
- more aggressive programming style
- less thinking about the dumb stuff
- copy&paste advantage
- faster refactoring

Why test research software

- Unit tests promote "better" software (simpler, faster, more robust, extensible)
=> Easier to understand, faster to change/extend
=> More experiments with trustworthy results in the same time
- Software is the fundament of your research:
Faulty software => faulty results
- Tests give you trust in your code and results
- If computer scientists don't test, why should anyone else?
- Research software requires MORE testing than most other code because nonsense is typically harder to recognize

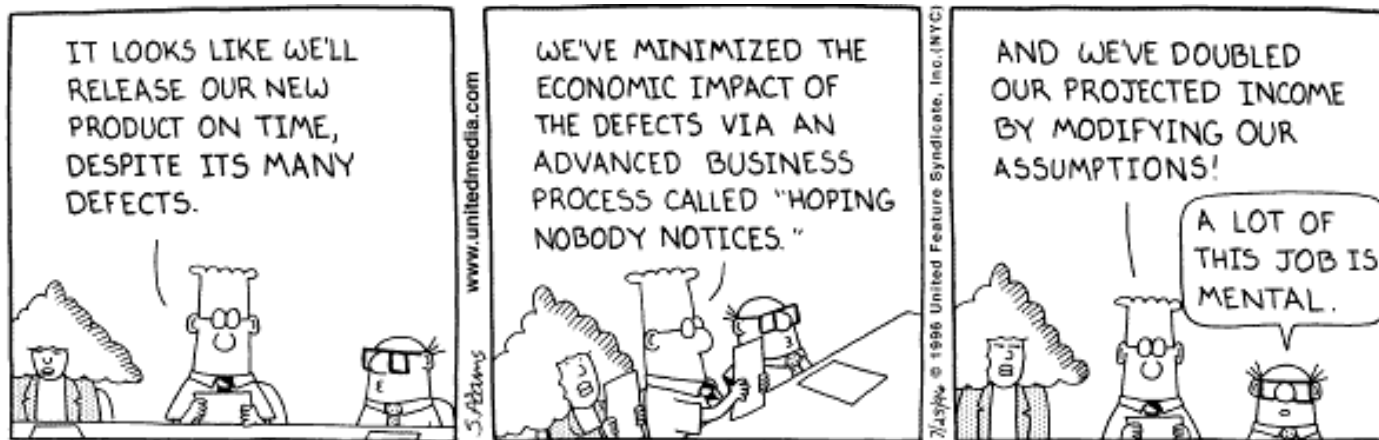
Keeping up with the Jones'

- Molecular Biologists:
Detailed description of tests that ensure proper function of chemicals == testing the code
- Bold idea:
Why not report unit test coverage or other measures of code quality to increase confidence in results

Programmers wisdoms

- Test == Trust
- Forget about late/black-box tests – if you can do better
- Test now – or never – better ever
- Replace debug time by test time
- Agile development (Tests, Patterns, Start simple)
- Build libraries, Bottom up
- Documentation and code belong together
- Tests and code belong together
- Code duplication is disgusting!
- Files are evil! Files are out!
- Complexity is your dead: The 10 finger limit
- Simplicity is the ultimate sophistication – Leonardo da Vinci

Questions ?



<http://itee.uq.edu.au/~stefan>

Links

- <http://thc.org/root/phun/unmaintain.html>
- <http://www.extremeprogramming.org/>
- http://en.wikipedia.org/wiki/Unit_test
- http://en.wikipedia.org/wiki/Software_testing
- <http://www.kaner.com/pdfs/metrics2004.pdf>
- <http://www.bullseye.com/coverage.html>
- http://auteurs.blog-city.com/coverage_tools_reduces_test_quality.htm
- <http://www-128.ibm.com/developerworks/java/library/j-cq01316/?ca=dnw-704>



Abstract

Testing Research Software: Nonsense or Necessity

Modern software development paradigms, such as Agile development, Extreme programming or Test first methods, are characterized by short development cycles and an emphasis on software patterns and unit tests.

Emerged and driven by needs in an industrial environment the question is if these techniques are useful in the scientific software development process as well.

With a focus on the test aspect only and I will discuss the benefits of unit testing for research software in the context of a library for biological sequence analysis.

Test tool requirements

- Ease of use
- 100% automated
- Immediate graphical feedback
- Tight coupling between coding and testing

My code is right, because

- it looks right
- the compiler didn't complain
- I used the debugger to check it
- I've got the results I wanted to get
- there are plenty of printf's
- I write everything to a file so that I can check it
- ...