Tools and Techniques

Track introduction

Tools you can use individually (part 1): Test Frameworks

The size of the task: Building software for a collaboration

What do you need to do the job?

I need to calculate the sum of primes less than 100:

```
int sumPrimes() {
   int sum = 0;
   for ( int i=1; i < 100; i++ ) {  // loop over possible primes
      bool prime = true;
      for (int j=1; j < 10; j++) { // loop over possible factors
         if (i % j == 0) prime = false;
      }
      if (prime) sum += i;
   }
   return sum;
}
```

This is quick, throw-away code

- Not well structured, efficient, general or robust
- I understand what I intended, because I wrote it just now

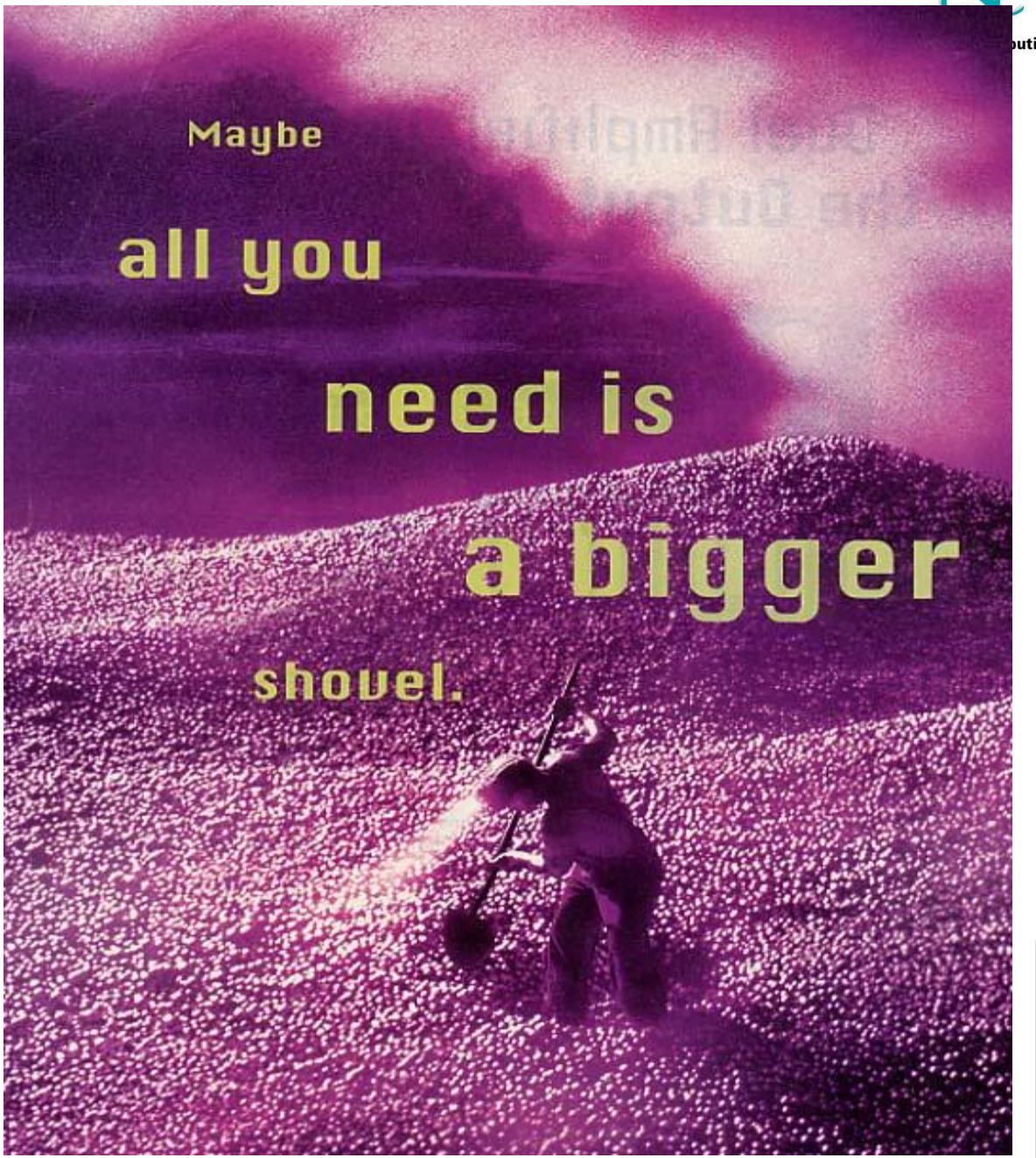
Already, I need an editor, compiler, linker, and probably a debugger

“Don’t worry, I’ll remember what I changed.”

“The answer looks OK, lets move on.”

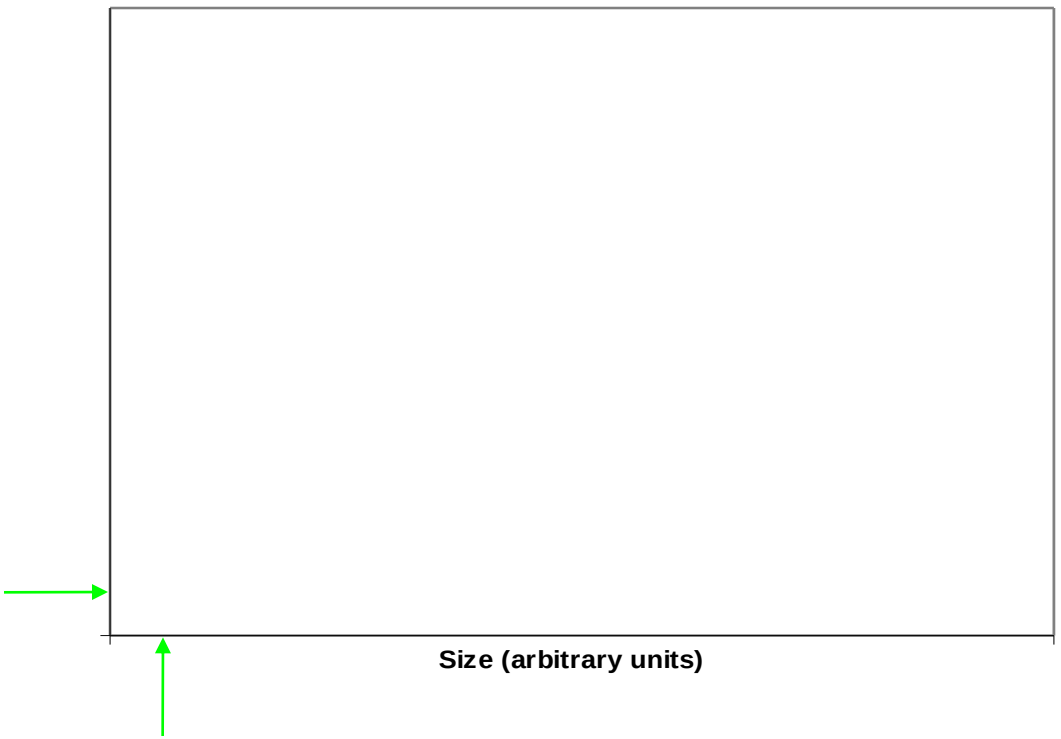
“Does anybody know where this value came from?”

“Your #%@!& code broke again!”



Projects come in different sizes

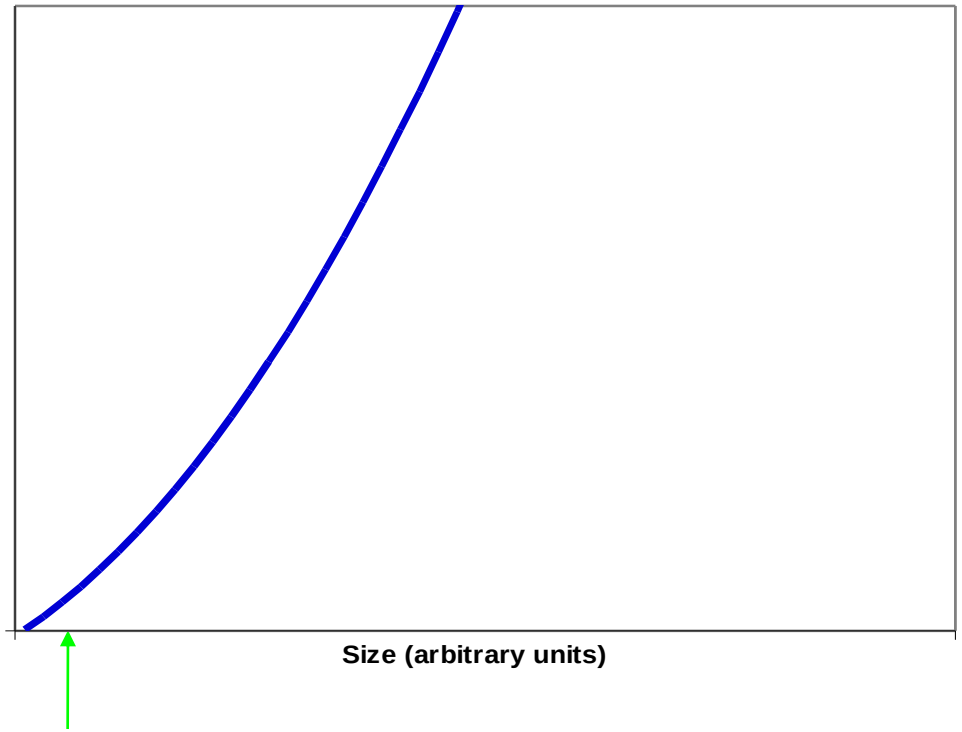
My sample program is a pretty small project!



Projects come in different sizes

My sample program is a pretty small project!

It can be done with a simple technique:



But that won't solve larger problems well

Projects come in different sizes

My sample program is a pretty small project!

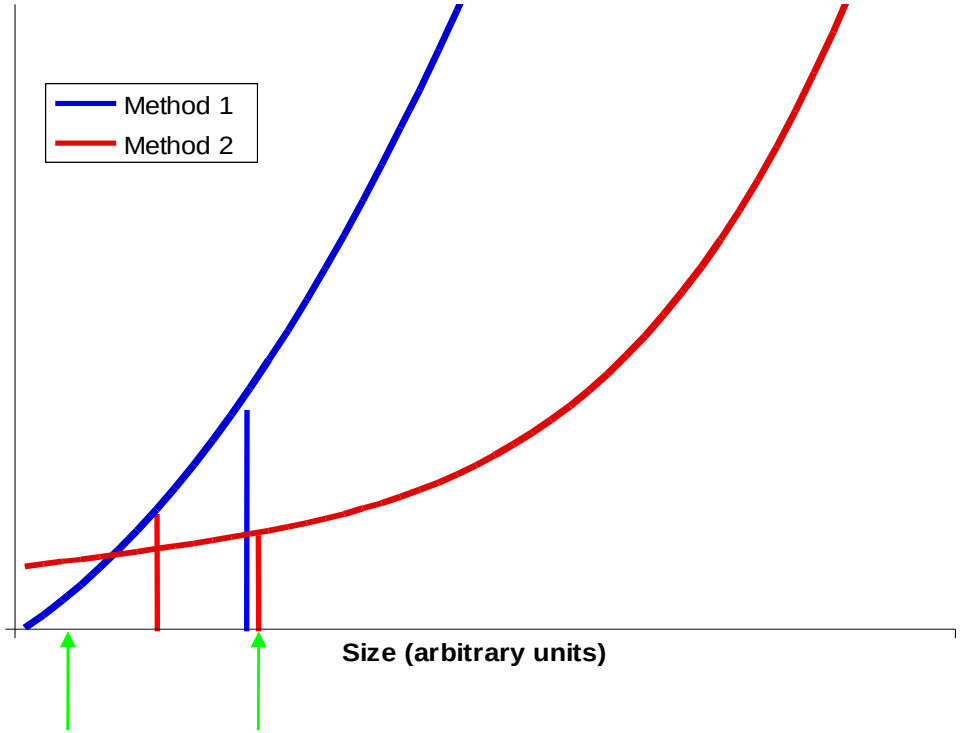
It can be done with a simple technique:

But that won't solve larger problems well

Projects come in different sizes

A larger project may need a different approach

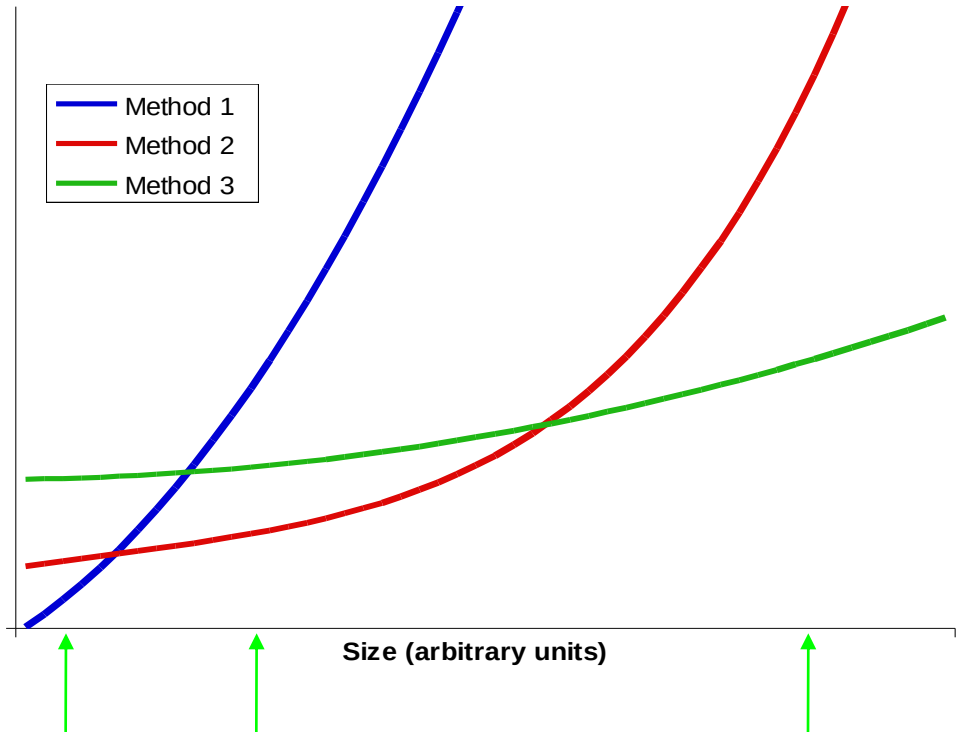
- **Those tend to require more effort up front**



What do you do when your project grows?

Projects come in different sizes

If you're trying to solve a really large problem:



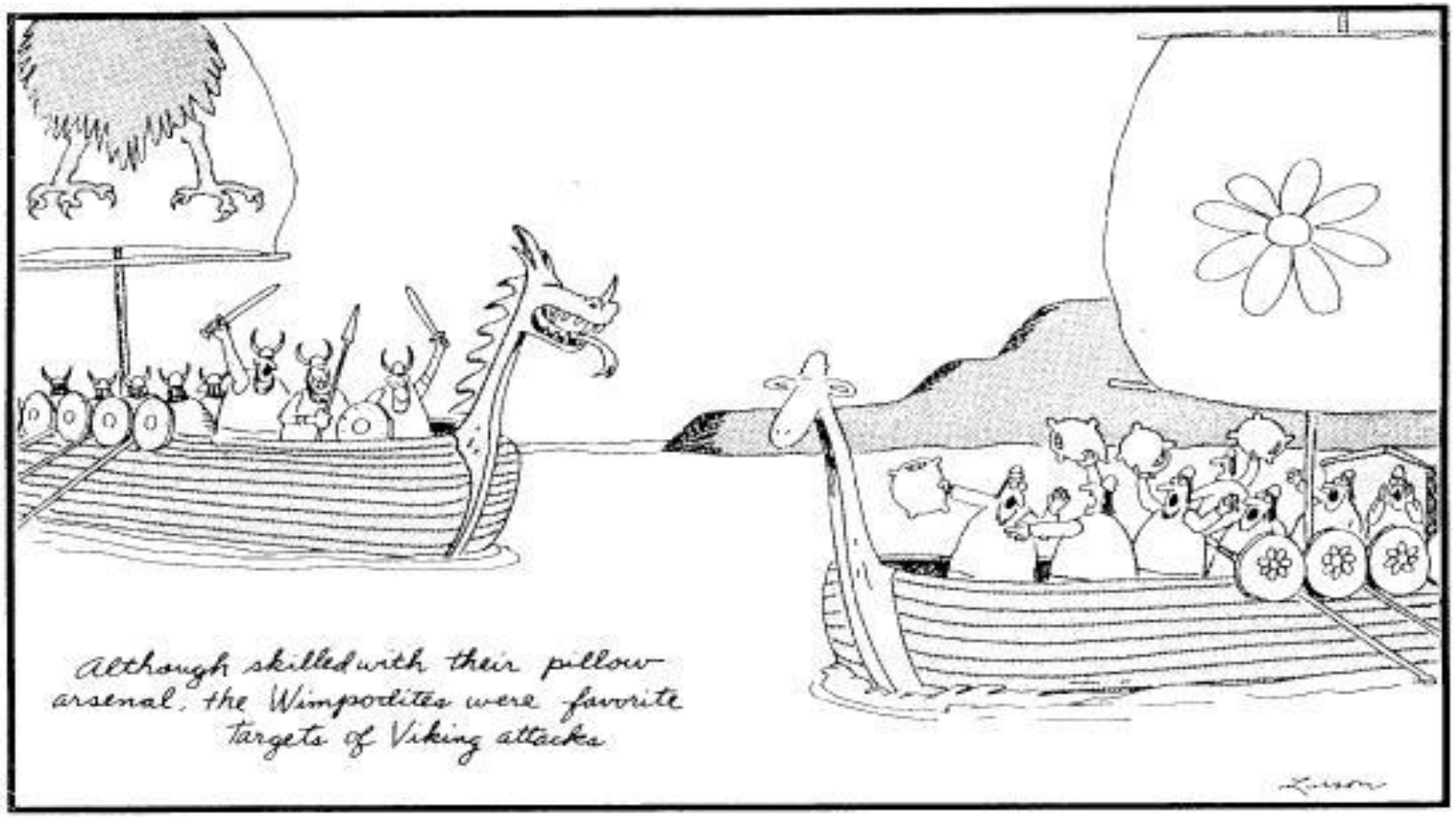
What has all this to do with us?

Our systems tend to be complex systems

- **HEP tends to work at the limit of what we know how to do**

“If you only have a hammer, wood screws look a lot like nails” - ??

“If you only have a screwdriver, nails are pretty useless” - Don Briggs



Larger projects have standard ways of doing things

To make it possible to communicate, you need a shared vocabulary

- **Standards for languages, data storage, etc.**

For people to work together, you have to control integrity of source code

- **E.g. CVS to provide versioning and control of source code**

Just building a large system can be difficult

- **Need tools for creating releases, tracking problems, etc.**

But individual effort is still important!

You can't build a great system from crummy parts

You want your efforts to make a difference

Good tools & methods can help you do a better job

"Whatever you do may seem insignificant, but it is most important that you do it." - Gandhi

"I've got it, too, Omar ... a strange feeling like we've just been going in circles."

The Base Technologies Track

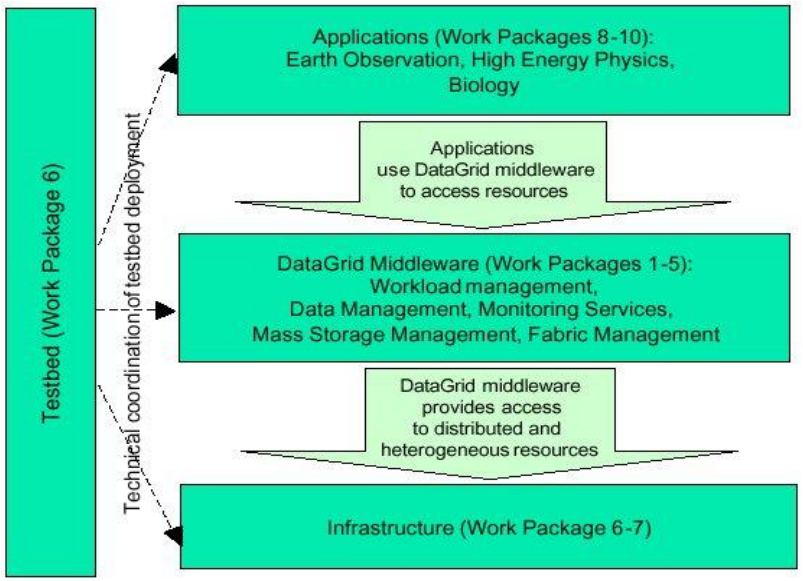
A spectrum of places to improve:

- **What you do in the next minutes**
- **What you do over the next years**

```
int sumPrimes() {
   int sum = 0;
   for ( int i=1; i < 100; i++ ) {  // loop over possible primes
      bool prime = true;
      for (int j=1; j < 10; j++) { // loop over possible factors
         if (i % j == 0) prime = false;
      }
      if (prime) sum += i;
   }
   return sum;
}
```

Three basic themes:

- **Individual tools & methods**
- **Working with existing code**
- **Working with large systems**



Organisation of the technical work packages in the DataGrid project

CERN School of Computing

Plan for this week:

Week 1 Version: 23 April

Dinners take place at Mariaspring Hotel (the hotel where participants are lodged), unless specified otherwise. Lunches take place at the Göttingen University cafeteria, unless specified otherwise.

Sun 16 Aug		Mon 17 Aug	Tue 18 Aug	Wed 19 Aug	Thu 20 Aug	Fri 21 Aug	Sat 22 Aug
Arrival	08:45 - 09:40	Opening Session	L Computer Security 1 A.Pace	L Secure Software 1 S.Lopienskik	L Computer Architecture & Perf. Tuning 3 S.Jarp A.Nowak	L ROOT Technologies 1 A.Naumann B.Bellenot	L ROOT Technologies 3 A.Naumann B.Bellenot
	09:50 - 10:45		L Tools and Techniques 3 B.Jacobsen	L Computer Architecture & Perf. Tuning 1 S.Jarp A.Nowak	L Introduction to Physics Computing 1 R.Frühwirth	L ROOT Technologies 2 A.Naumann B.Bellenot	E ROOT Technologies 2 A.Naumann B.Bellenot
	10:45		Announcements	Announcements	Announcements	Announcements	Announcements
	11:05	Coffee	Coffee	Coffee	Coffee	Coffee	Coffee
	11:30 - 12:25	L Tools and Techniques 1 B.Jacobsen	L Computer Security 2 A.Pace	L Secure Software 2 S.Lopienski	L Introduction to Physics Computing 2 R.Frühwirth	E ROOT Technologies 1 A.Naumann B.Bellenot	E ROOT Technologies 3 A.Naumann B.Bellenot
	12:30	Lunch	Lunch	Lunch	Lunch	Lunch	Outside Lunch
	13:30 - 14:30	Transportation to University Campus	Free Time Study Time* Sport Programme Today sport options	Free Time Study Time* Sport Programme Today sport options	Sport/Excursion afternoon Schedule overview All combinations Details of the activities, Detailed Schedule	Free Time Study Time* Sport Programme Today sport options*	Free Time *Optional : 14:00 Guided tour of famous scientific historical places in Gottingen* *TBC:18:00 Talk from Gustav Born at Mariaspring*
	14:30 - 15:30	Presentation of Sport/Social activities					
	15.30	Coffee	Coffee			Coffee	
	16:00 - 16:55	L Tools and Techniques 2 B.Jacobsen	E Tools and Techniques 3 B.Jacobsen	L Computer Architecture & Perf. Tuning 2 S.Jarp A.Nowak		E Computer Architecture & Perf. Tuning 1 S.Jarp A.Nowak	
	17:05 - 18:00	E Tools and Techniques 1 B.Jacobsen	E Tools and Techniques 4 B.Jacobsen	E Secure Software 1 S.Lopienski		E Computer Architecture & Perf. Tuning 2 S.Jarp A.Nowak	
	18:05 - 19:00	E Tools and Techniques 2 B.Jacobsen	E Tools and Techniques 5 B.Jacobsen	E Secure Software 2 S.Lopienski		E Computer Architecture & Perf. Tuning 3 S.Jarp A.Nowak	
	19:30	Welcome Reception Details TBC					
Dinner	20:15	Details TBC	Dinner at Mariaspring	Dinner at Mariaspring	Special Dinner at BurgPlesse castle at Bovenden	Dinner at Mariaspring	Dinner at Mariaspring
After Dinner event		TBD	TBD	TBD	TBD	TBD	TBD

CERN School of Computing

Tools you can use

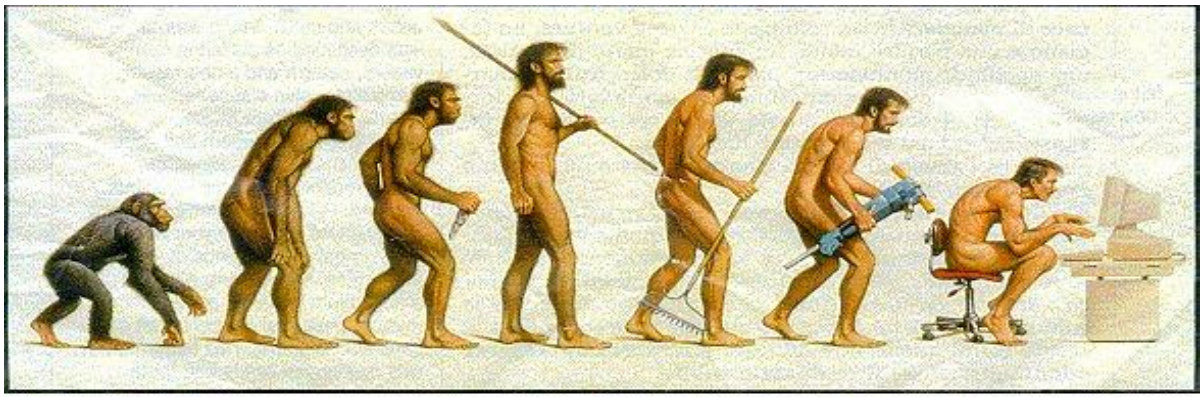
Knowing whether it works - JUnit

Toward an informed way of experimental working

Progress often comes from small, experimental changes

- **Allows you to make quick progress on little updates**
- **Without risk to the big picture**

How do you know those steps are progress?



Somewhere, something went terribly wrong

Testing

But don’t you see Gerson - if the particle is too small and too short-lived to detect, we can’t just take it on faith that you’ve discovered it.”

The role of testing tools

Remember our original example:

- **Simple routine, written in a few minutes**
- **“So simple it must be right”**

```
int sumPrimes() {
   int sum = 0;
   for ( int i=1; i < 100; i++ ) {  // loop over possible primes
      bool prime = true;
      for (int j=1; j < 10; j++) { // loop over possible factors
         if (i % j == 0) prime = false;
      }
      if (prime) sum += i;
   }
   return sum;
}
```

But it’s not right...

"Study it forever and you'll still wonder. Fly it once and you'll know.”
- Henry Spencer

How to test?

Simplest: Run it and look at the output

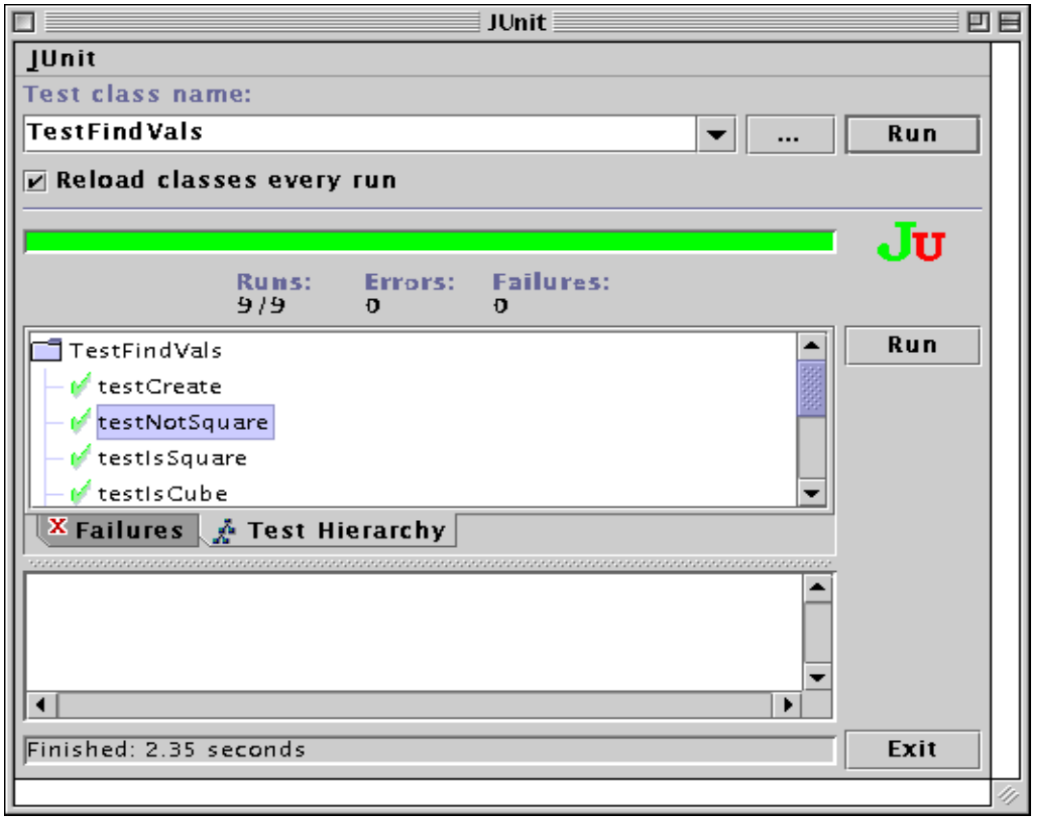
- Gets boring fast!
- How often are you willing to do this?

More realistic: Code test routines to provide inputs, check outputs

- Can become ungainly

Most useful: A test framework

- Great feedback
- Better control over testing



Testing Frameworks: CppUnit, Junit, et al

To test a function:

```
public class FindVals {
   // determine whether an number is a square
   boolean isSquare(int val) {
            double root = Math.floor(Math.pow(val, 0.5));
            if (Math.abs(root*root - val) < 1.E-6 ) return true;
            else return false;
   }
}
```

You write a test:

```
 public void testIsSquare() {
            FindVals s = new FindVals();
            Assert.assertTrue( s.isSquare(4) );
   }
```

Invoke a function

Check the result

Plus tests for other cases…

Embed that in a framework

Gather together all the tests

```
// define test suite
   public static Test suite() {
         // all tests from here down in heirarchy
         TestSuite suite = new TestSuite(TestFindVals.class);
         return suite;
   }
```

Junit uses class name to find tests

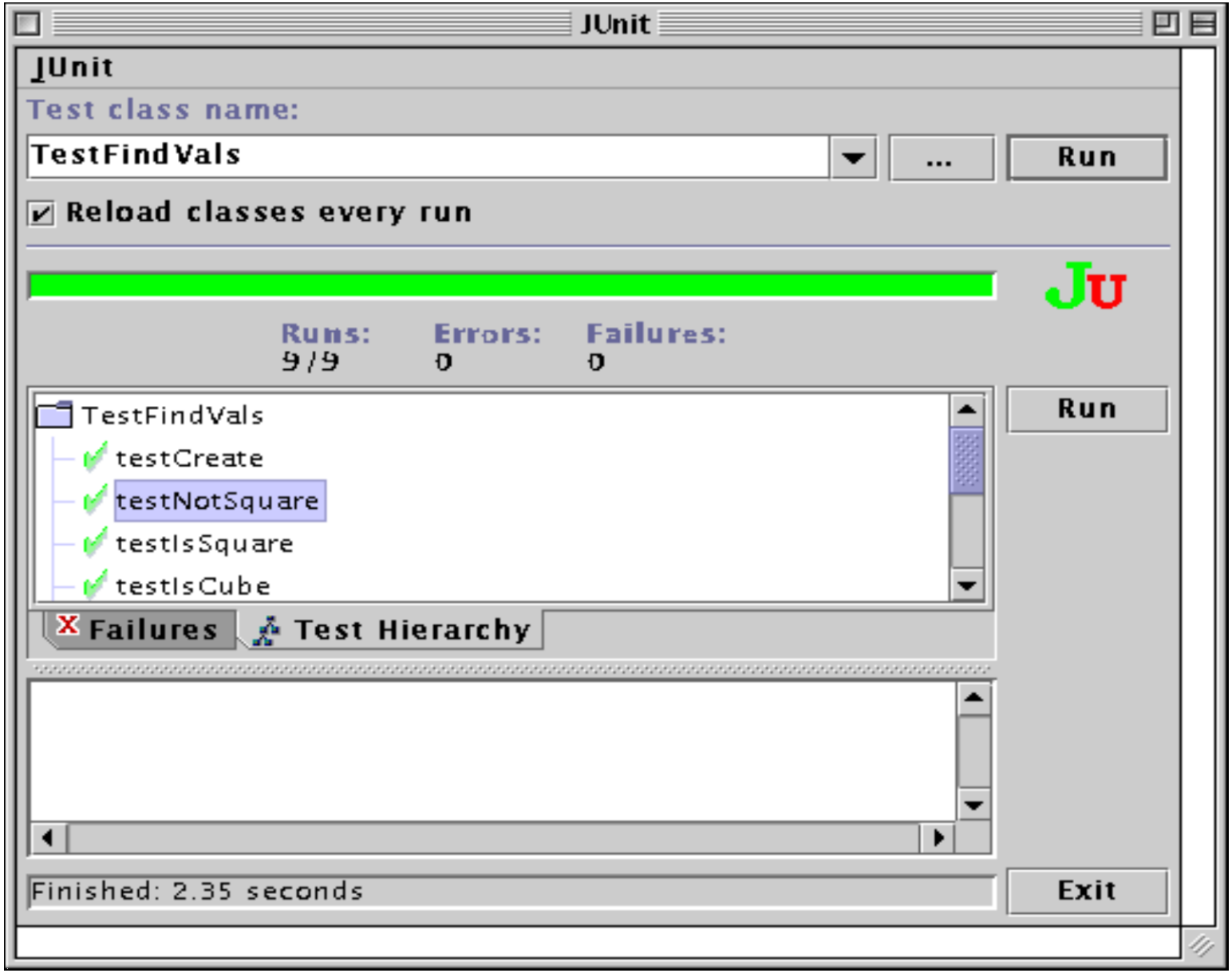
Start the testing

- **To just run the tests:** `junit.textui.TestRunner.main(TestFindVals.class.getName());`
- **Via a GUI:** `junit.swingui.TestRunner.main(TestFindVals.class.getName());`

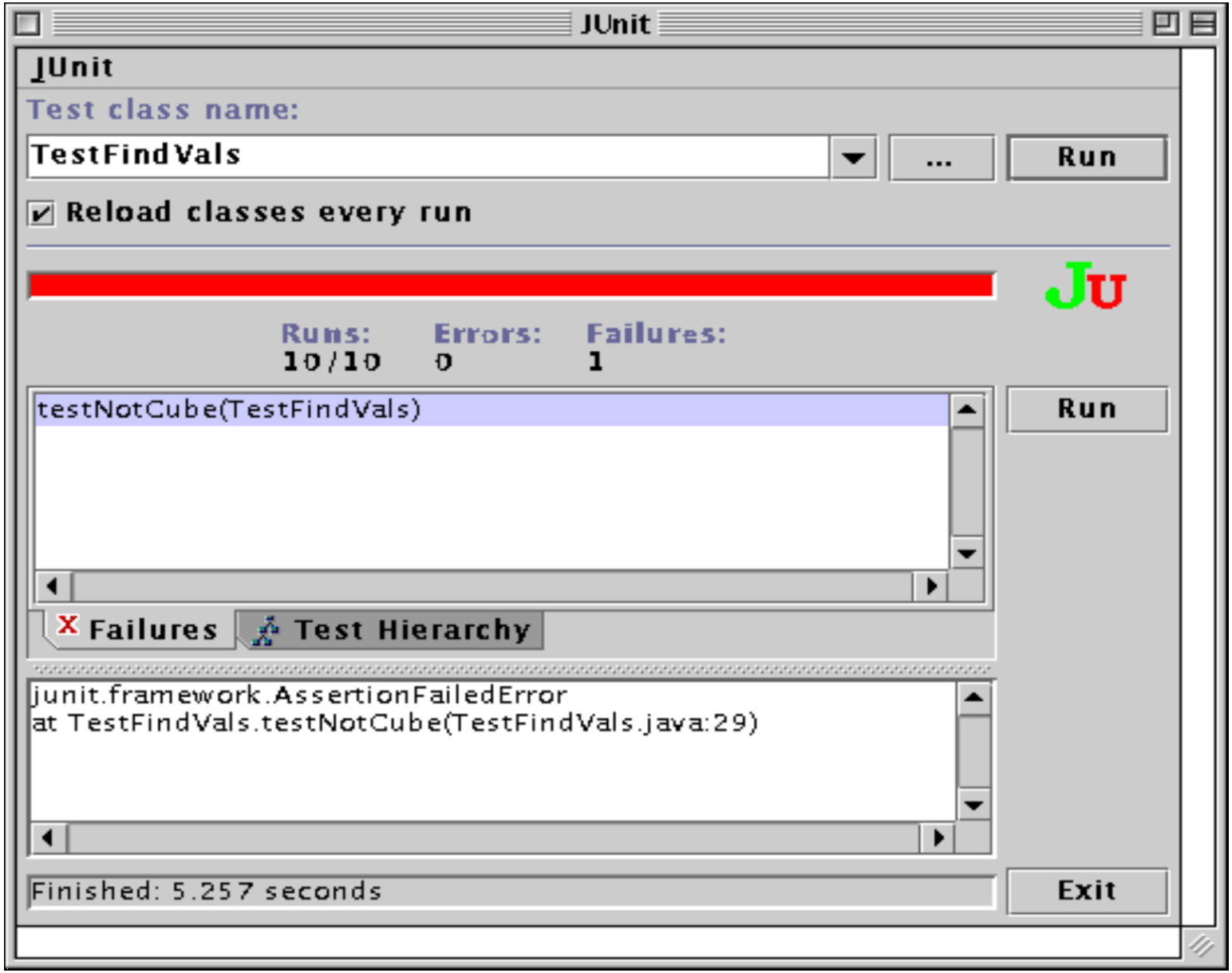
And that's it!

Invoke tests for my class

Running the tests



Running the tests



How JUnit works - one test:

```
public void testOneIsPrime() {
        SumPrimes s = new SumPrimes();
        Assert.assertEquals("check sumPrimes(1)", 1, s.sumPrimes(1));
}
```

This defines a “method” (procedure) that runs one test (line 1 and 4)

- JUnit treats as a test procedure any method whose name starts with “test”
- The tests will be run in the order they appear in the file

Line 2 creates an object “s” to be tested

Line 3 checks that sumPrimes(1) returns a 1

Assert is a class that checks conditions

assertEquals(“message”, valueExpected, valueToTest) does the check

If the check fails, the message and observed values are displayed

If the check fails:

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.

Other views:

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.

Why?

One test isn't worth very much

- Maybe saves you a couple seconds once or twice

But consistently building the tests as you build the code does have value

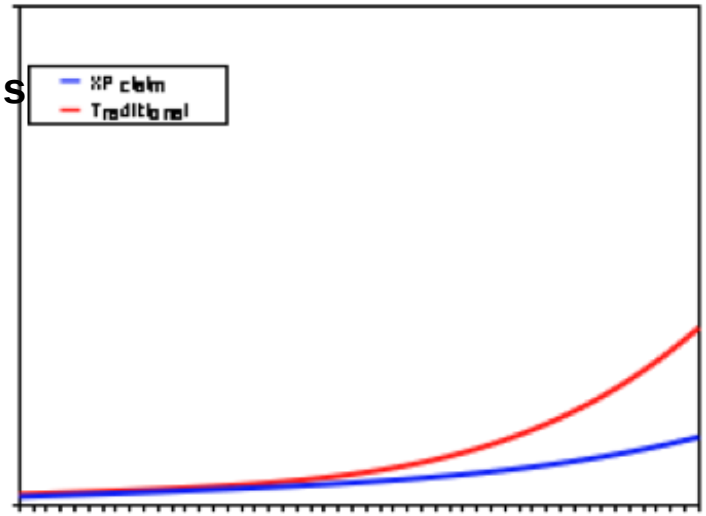
- Have you ever broken something while fixing a bug? Adding a feature?

  Tests remember what the program is supposed to do

- A set of tests is definitive documentation for what the code does
- Alternating between writing tests and code keeps the work incremental

  Keeping the tests running prevents ugly s

- And it's very satisfying!



"Extreme Programming" advocates writing the tests before the code

- Not clear for large projects
- But individuals report good results

The art of testing

What makes a good test?

- Not worth testing something that's too simple to fail
- Some functionality is too complex to test reliably
- Best to test functionality that you understand, but can imagine failing
  - If you're not sure, write a test
  - If you have to debug, write a test
  - If somebody asks what it does, write a test

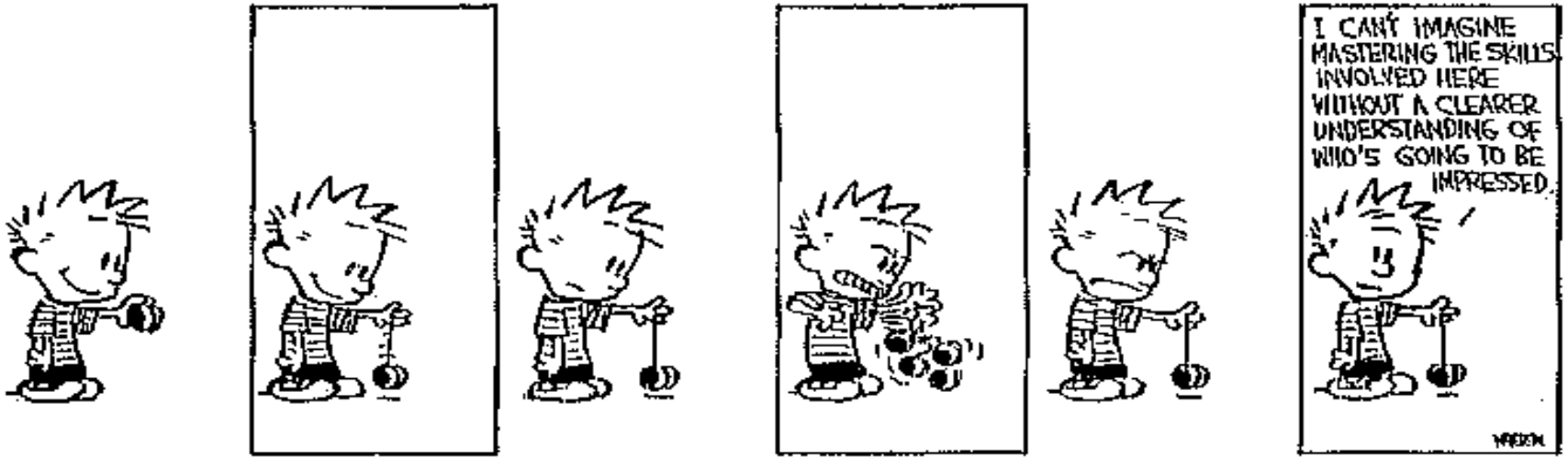
How big should a test be?

- A JUnit test is a unit of failure
  - When a test fails, it stops
  - The pattern of failures can tell you what you broke
- Make lots of small tests so you know what still works

What about existing code?

- Probably not practical to sit down and write a complete set of tests
- But you can write tests for new code, modifications, when you have a question about what it does, when you have to debug it, etc

Summary 1



The principle of 'I think, therefore I am', does not apply to high quality software. - Malcolm Davis

In art, intentions are not enough. What counts is what one does, not what one intends to do. - Pablo Picasso

Excellence is not a single act, but a habit. You are what you repeatedly do. - Aristotle, as quoted by Shaquille O'Neal