Tools and Methods

Track introduction

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

OK, HOBBES, OUR TIME MACHINE IS ALL SET. PUT ON YOUR GOGGLES AND WE'LL BE OFF TO THE FUTURE!
TIME MACHINE
©1987 Universal Press Syndicate
WHY DO WE HAVE TO WEAR GOGGLES?
GEEZ, DO YOU THINK TRAVELING YEARS INTO THE FUTURE IS LIKE DRIVING DOWN THE STREET?!
WATERSON
WE'VE GOT TO CONTEND WITH VORTEXES AND LIGHT SPEEDS! ANYTHING COULD GO WRONG! OF COURSE WE NEED TO WEAR GOGGLES!
GOSH, I THINK MY GOGGLES ARE IN THE BEDROOM. IF I'M NOT BACK IN A COUPLE MINUTES, YOU CAN GO WITHOUT ME.
SIT DOWN, SISSY. I ALREADY GOT YOUR GOGGLES.
TIME MACHINE
9-2

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**

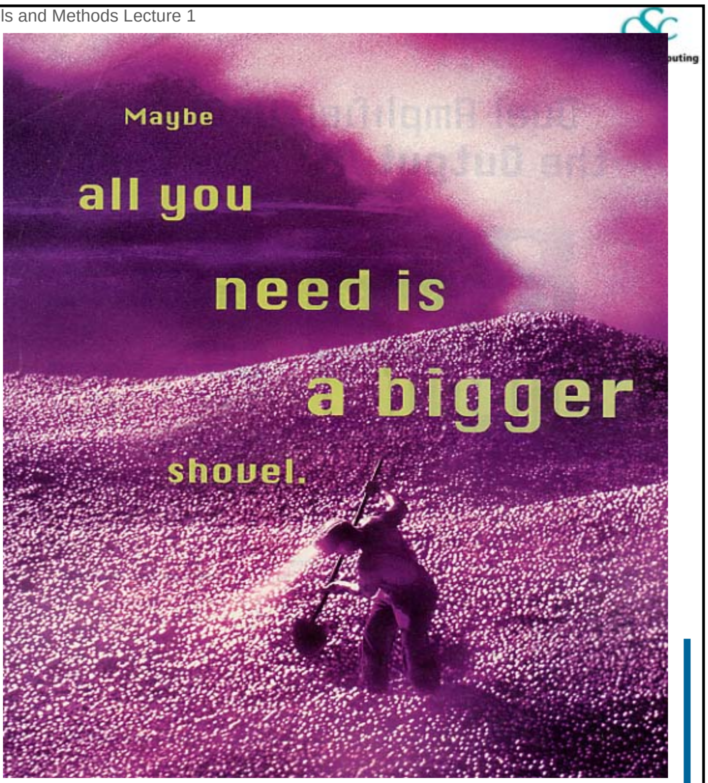
Already, I need an editor, compiler, linker, and probably a debugger

"Don't worry, I'll remember what I changed."

"The answer looks OK, lets move on."

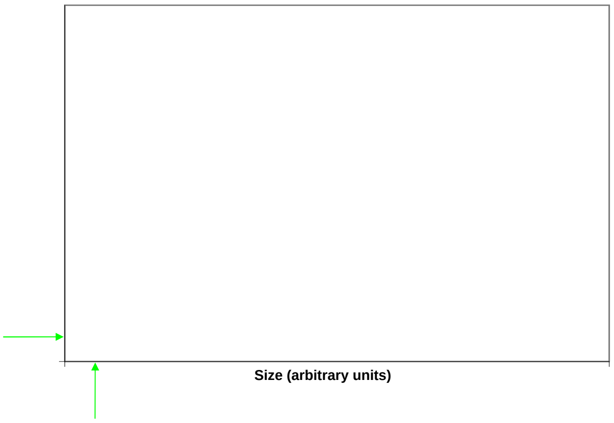
"Does anybody know where this value came from?"

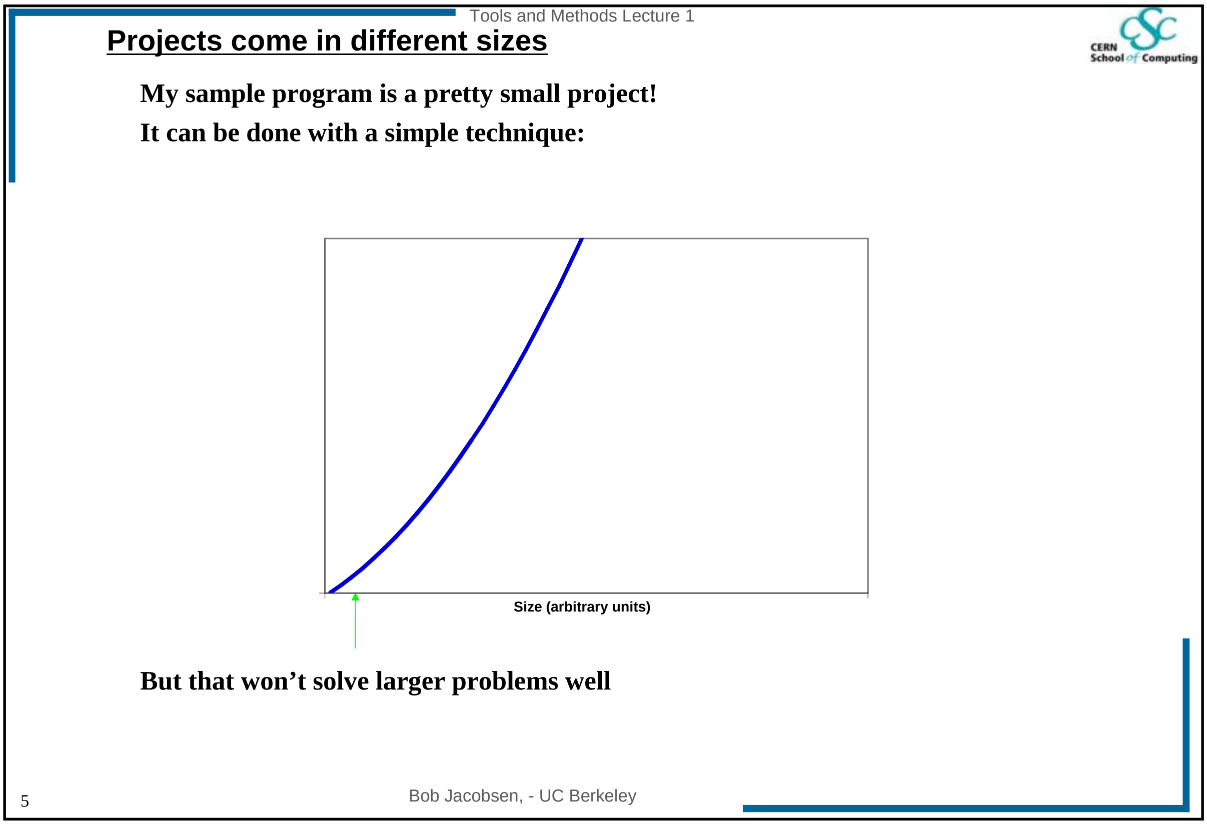
"Your #%@!& code broke again!"



Projects come in different sizes

My sample program is a pretty small project!



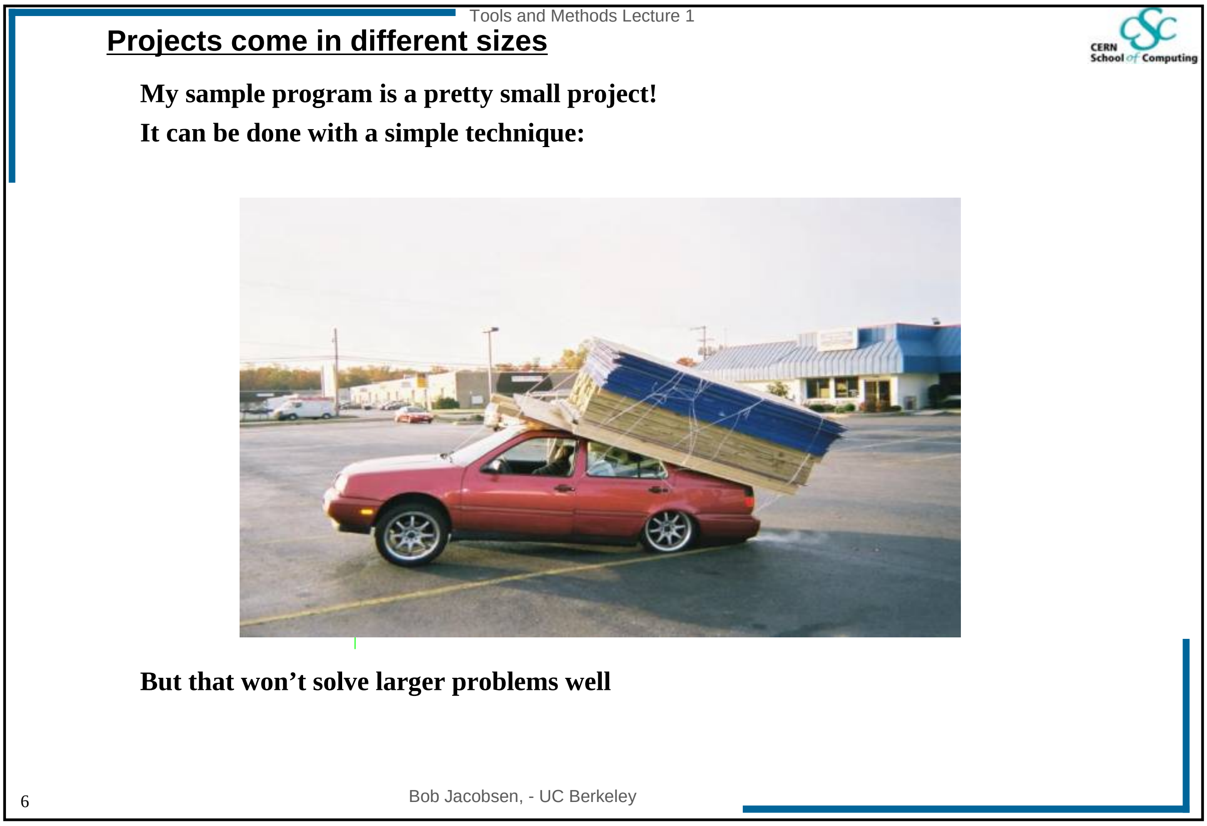


Projects come in different sizes

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!

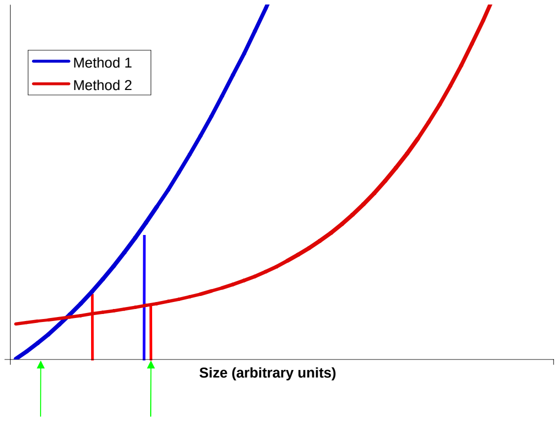
It can be done with a simple technique:

But that won't solve larger problems well

Projects come in different sizes

A larger project may need a different approach

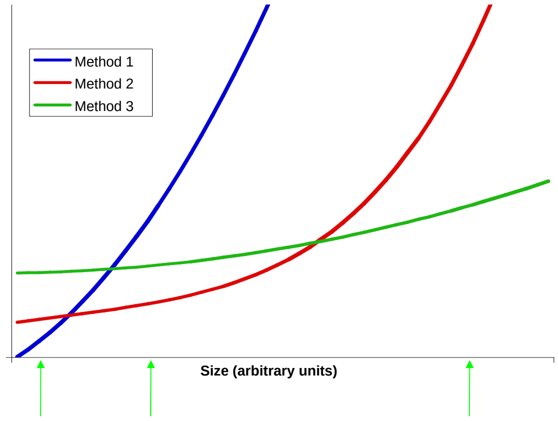
- Those tend to require more effort up front



What do you do when your project grows?

Projects come in different sizes

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



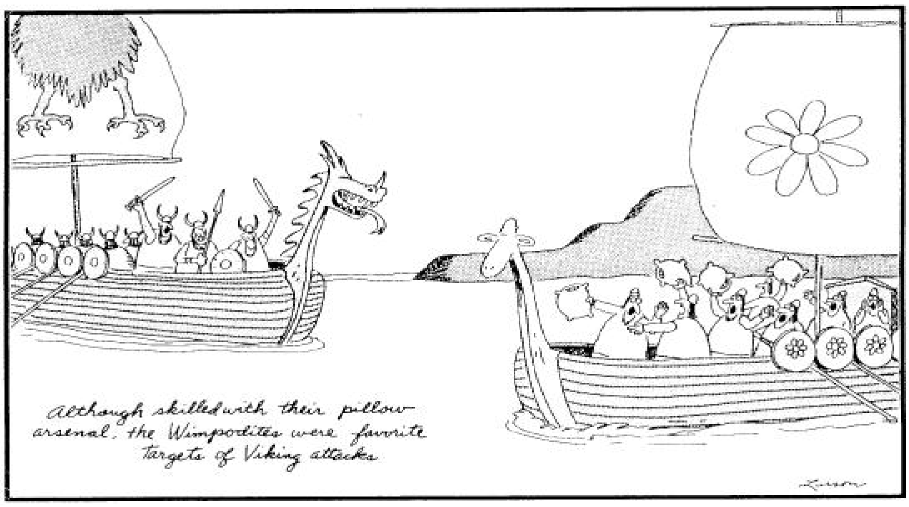
What has all this to do with us?

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


Although skilled with their pillow arsenal, the Wimpodites were favorite targets of Viking attacks

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.

I DON'T WANNA TAKE A BATH! I HATE TAKING BATHS!
AAAAAAAAAAAAA
NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO NO
THEY CAN MAKE ME DO IT, BUT THEY CAN'T MAKE ME DO IT WITH DIGNITY.

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 Tools & Method Track

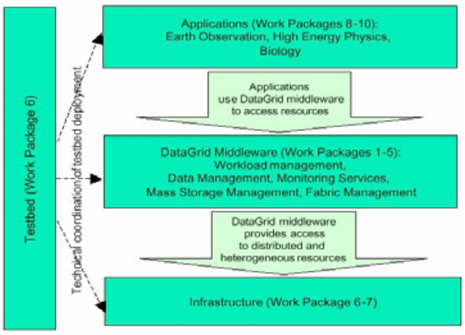
A spectrum of places to improve:

- **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**
- **Building new systems**


Testbed (Work Package 6)
Technical coordination of testbed deployment
Applications (Work Packages 8-10): Earth Observation, High Energy Physics, Biology
Applications use DataGrid middleware to access resources
DataGrid Middleware (Work Packages 1-5): Workload management, Data Management, Monitoring Services, Mass Storage Management, Fabric Management
DataGrid middleware provides access to distributed and heterogeneous resources
Infrastructure (Work Package 6-7)

Organisation of the technical work packages in the DataGrid project

Plan fo this we

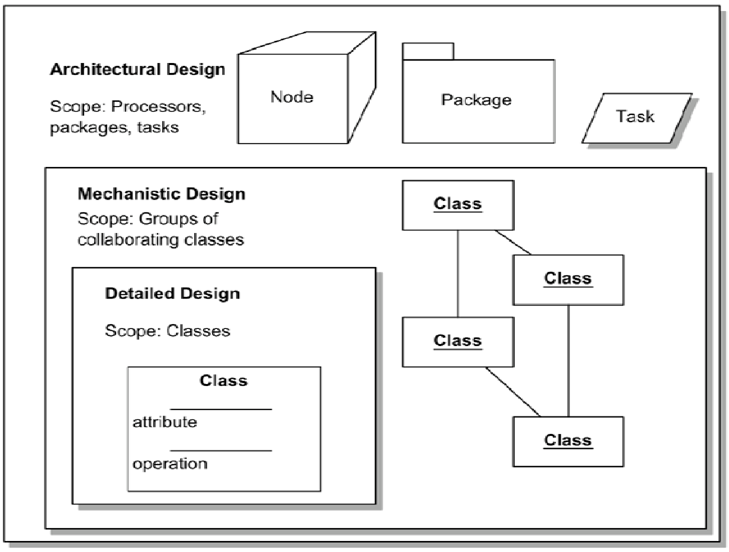
Sun 19 Aug		Mon 20 Aug	Tue 21 Aug	Wed 22 Aug	Thu 23 Aug	Fri 24 Aug	Sat 25 Aug
Arrival	09.00 - 09.55	Opening Session Part 1	L Computer Security 1 A.Pace	L Computer Security 2 A.Pace	L Computer Security 3 A.Pace	L Introduction to Physics Computing 2 R.Frühwirth	L ROOT Technologies 4 A.Naumann B.Bellenot
	10.05 - 11.00	Opening Session Part 2	L Tools and Techniques 3 B.Jacobsen	L Secure Software 1 S.Lopienski	L Secure Software 2 S.Lopienski	L ROOT Technologies 1 A.Naumann B.Bellenot	E ROOT Technologies 3 A.Naumann B.Bellenot
	11.05	Coffee	Coffee	Coffee	Coffee	Coffee	Coffee
	11.30 - 12.25	L Tools and Techniques 1 B.Jacobsen	L Web Services 1 A.Pace	L Web Services 2 A.Pace	L Introduction to Physics Computing 1 R.Frühwirth	E ROOT Technologies 1 A.Naumann B.Bellenot	E ROOT Technologies 4 A.Naumann B.Bellenot
	12.30	Lunch	Lunch	Lunch	Lunch	Lunch	Lunch
	13:30 - 14:30	Free Time	Free Time Sport Programme	Excursion (Details TBC)	Free Time Sport Programme	Free Time Sport Programme	Free Time Sport Programme (Details TBC)
	14:30 - 15:30	Presentation Sport/Social activities - TBC	Free Time Sport Programme Study Time*		Free Time Sport Programme Study Time*	Free Time Sport Programme Study Time*	
	15.30	Coffee	Coffee		Coffee	Coffee	
	16.00 - 16:55	L Tools and Techniques 2 B.Jacobsen	E Tools and Techniques 3 B.Jacobsen		Reserve	L ROOT Technologies 2 A.Naumann B.Bellenot	
	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 ROOT Technologies 2 A.Naumann B.Bellenot	
	18.05 - 19:00	E Tools and Techniques 2 B.Jacobsen	E Tools and Techniques 5 B.Jacobsen		E Secure Software 2 S.Lopienski	L ROOT Technologies 3 A.Naumann B.Bellenot	

Design

System architecture

Individual project

Specific task



Architectural Design

Scope: Processors, packages, tasks

Mechanistic Design

Scope: Groups of collaborating classes

Detailed Design

Scope: Classes

"Design" is how you think about what you're doing

Design Levels: an analogy

Imagine the project is not to build software but to go on an inter-planetary journey...

Architectural design

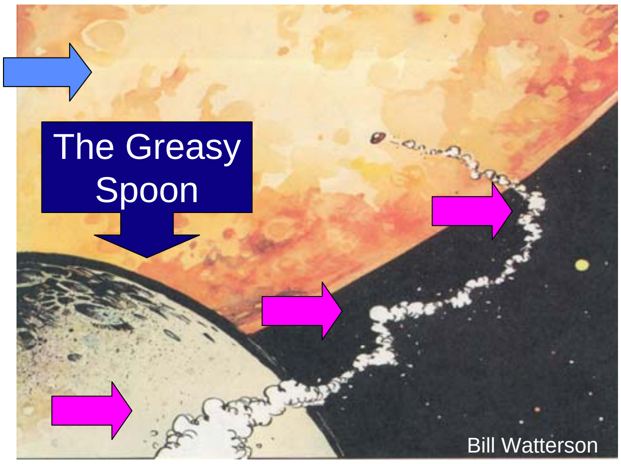
decide which planet to fly to

Mechanistic design

select the flight path

Detailed design

choose where to have lunch

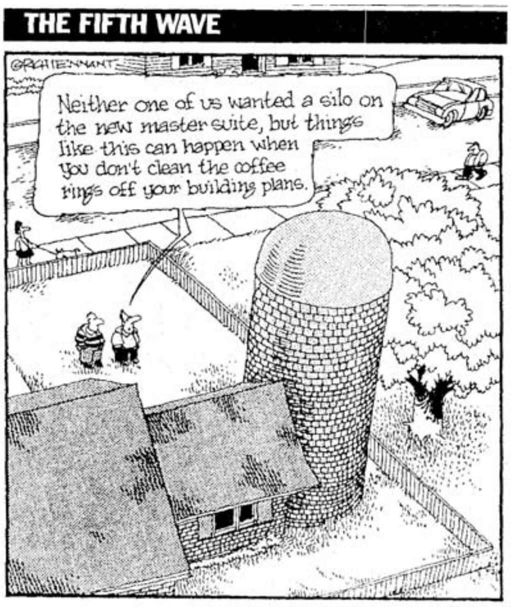


Bill Watterson

Architectural design

Goals

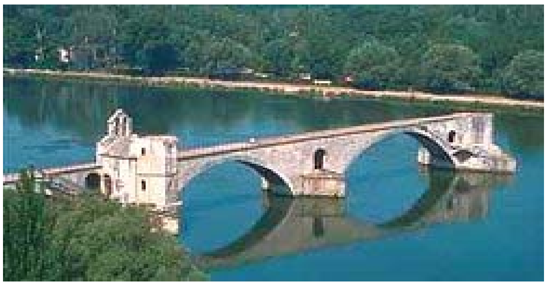
- **Capture major interfaces between subsystems and packages early**
- **Be able to visualize and reason about the design in a common notation**
- **Be able to break work into smaller pieces that can be developed by different teams (concurrently)**
- **Acquire an understanding of non-functional constraints**
  - **programming languages and operating systems**
  - **technologies: distribution, concurrency, database, GUIs**
  - **component reuse**


THE FIFTH WAVE
©RICHTENNANT
Neither one of us wanted a silo on the new master suite, but things like this can happen when you don't clean the coffee rings off your building plans.

Architectural Design Qualities

A well designed architecture has certain qualities:

- layered subsystems
- low inter-subsystem coupling
- robust, resilient and scalable
- high degree of reusable components
- clear interfaces
- driven by the most important and risky use cases
- EASY TO UNDERSTAND



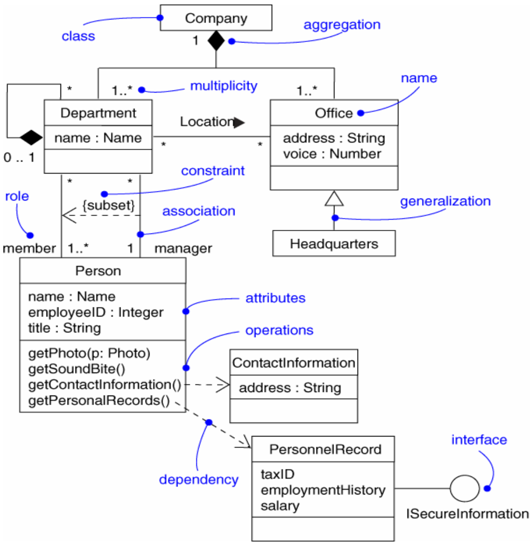
Mechanistic Design

Specify the details of inter-object collaboration *mechanisms*

- Determine the *structure* of classes and their associations
  - Class diagram
- Determine the *behavior* of classes
  - Interaction diagrams
    - Collaboration
    - Sequence
- Target: The people working together
  - Over time & space
  - You can’t do everything!

Class Diagram

Describes the types of objects in the system and the various kinds of static relationships that exist between them



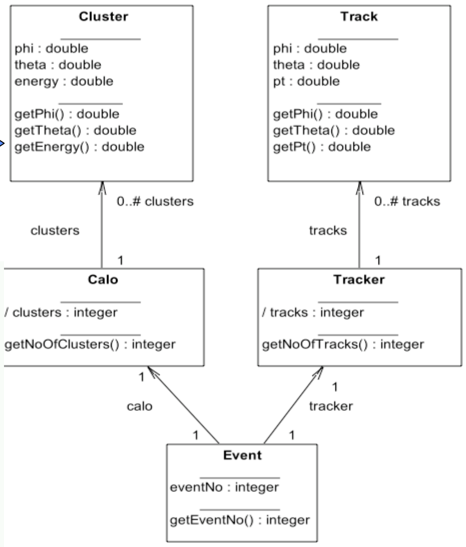
Rational Software Corporation

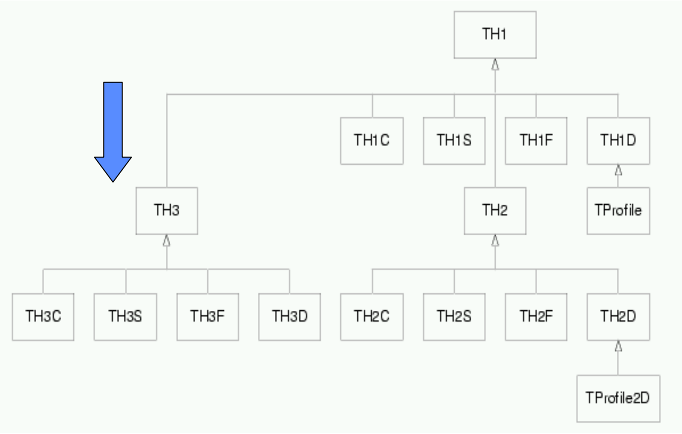
Example Class Diagrams

There are many possible designs

Goal: Allow you to reason about the strengths and weaknesses of a particular choice

Communicate through time and space





Building software is difficult

It cannot be learned from a book

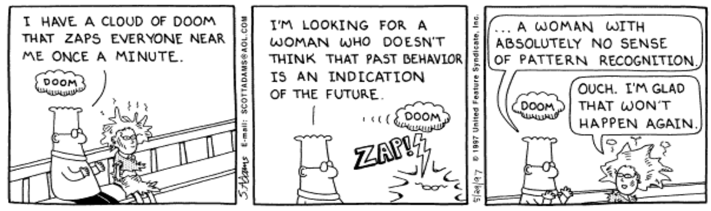
- You have got to do it and make mistakes
- Only time will tell if the result is “good”

It is a creative activity

- And hence enjoyable
- Not always clear when you should stop

It requires experience

- After a while you will tend to be more cautious and less ambitious
- Try to keep it simple
  - You will remember past-project horror stories
  - Or am I just getting old?


I HAVE A CLOUD OF DOOM THAT ZAPS EVERYONE NEAR ME ONCE A MINUTE.
DOOM
I'M LOOKING FOR A WOMAN WHO DOESN'T THINK THAT PAST BEHAVIOR IS AN INDICATION OF THE FUTURE.
DOOM
ZAP!
... A WOMAN WITH ABSOLUTELY NO SENSE OF PATTERN RECOGNITION.
DOOM
OUCH. I'M GLAD THAT WON'T HAPPEN AGAIN.

Copyright © 1997 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited

Tools you can use

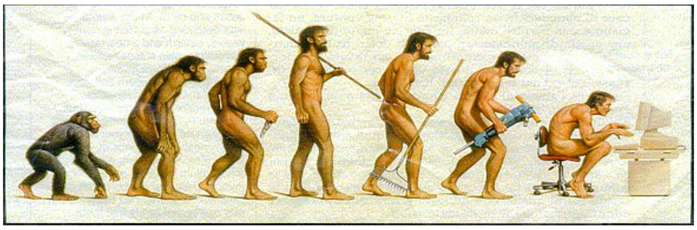
Knowing whether it works - JUnit

Toward an informed way of experimental working

These techniques remove the cost 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

© 1994 by Sidney Harris

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 its 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

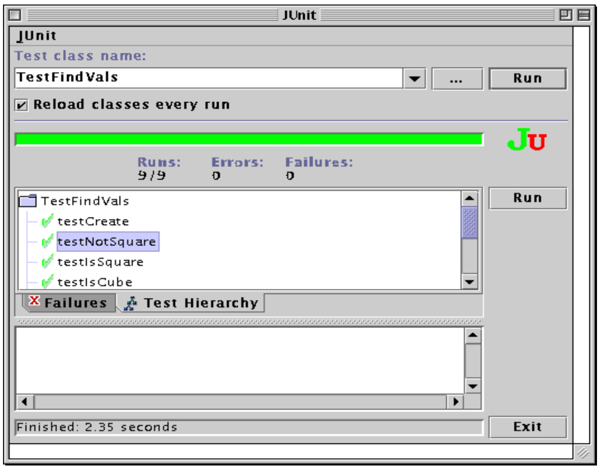
- **Gets boring fast!**
- **How often are you willing to do this?**

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

- **Can become ungainly**

Most useful: A test framework

- **Great feedback**
- **Better control over testing**



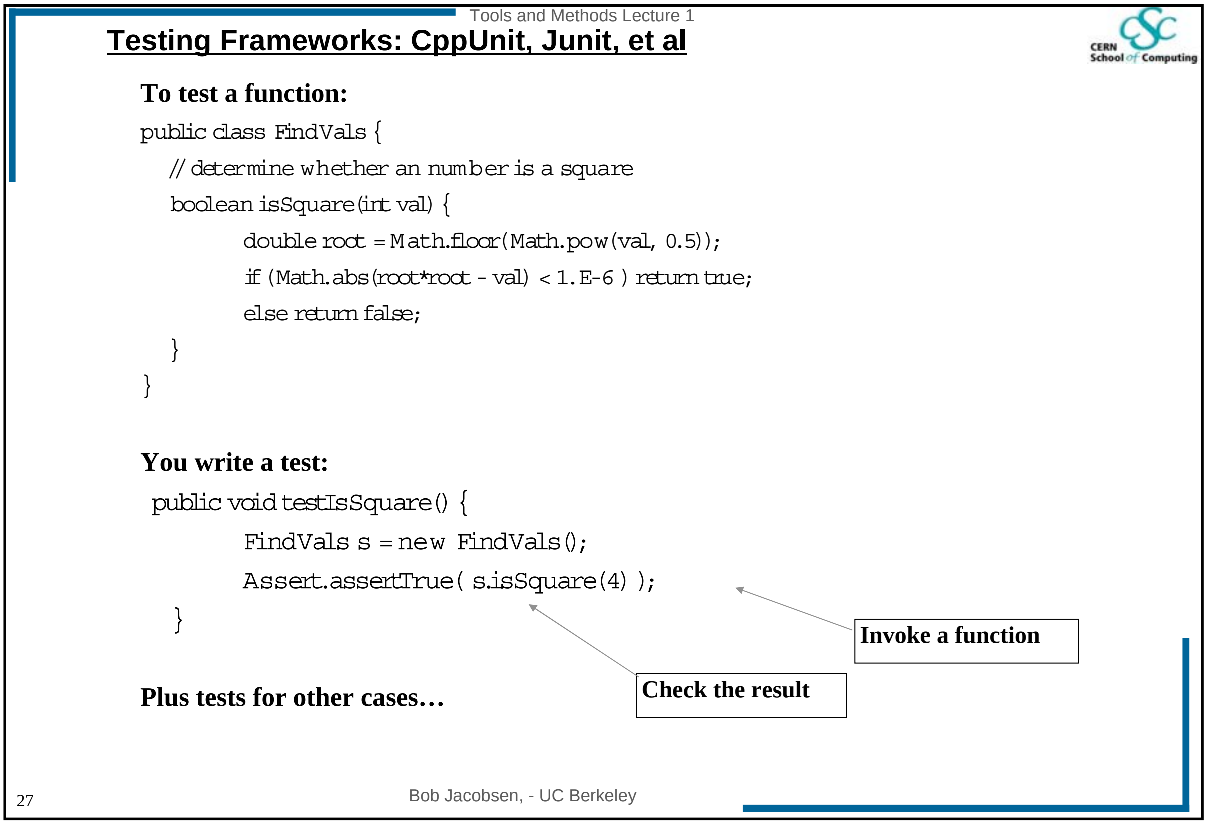
Test class name: TestFindVals

☑ Reload classes every run

Runs: 9/9   Errors: 0   Failures: 0

TestFindVals
- ✓ testCreate
- ✓ testNotSquare
- ✓ testIsSquare
- ✓ testIsCube

Finished: 2.35 seconds



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;
   }
}
```

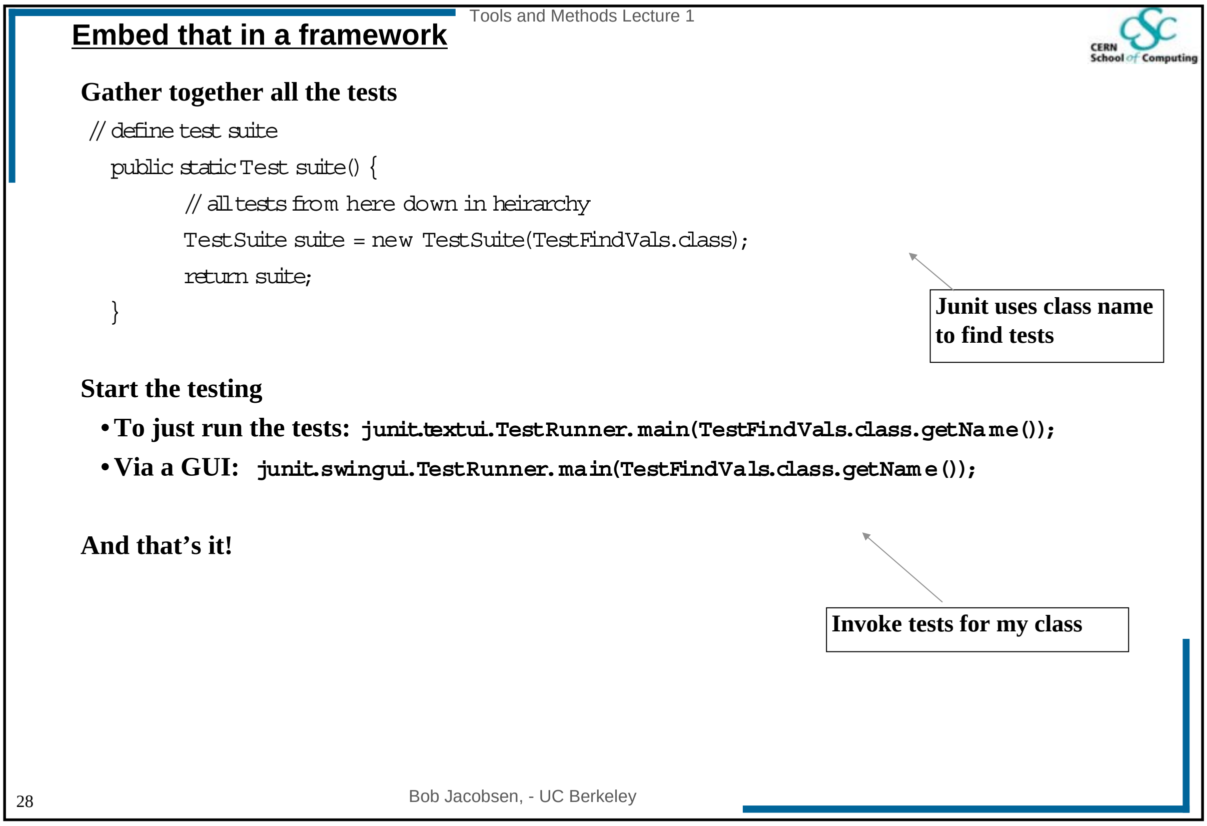
You write a test:

```
 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

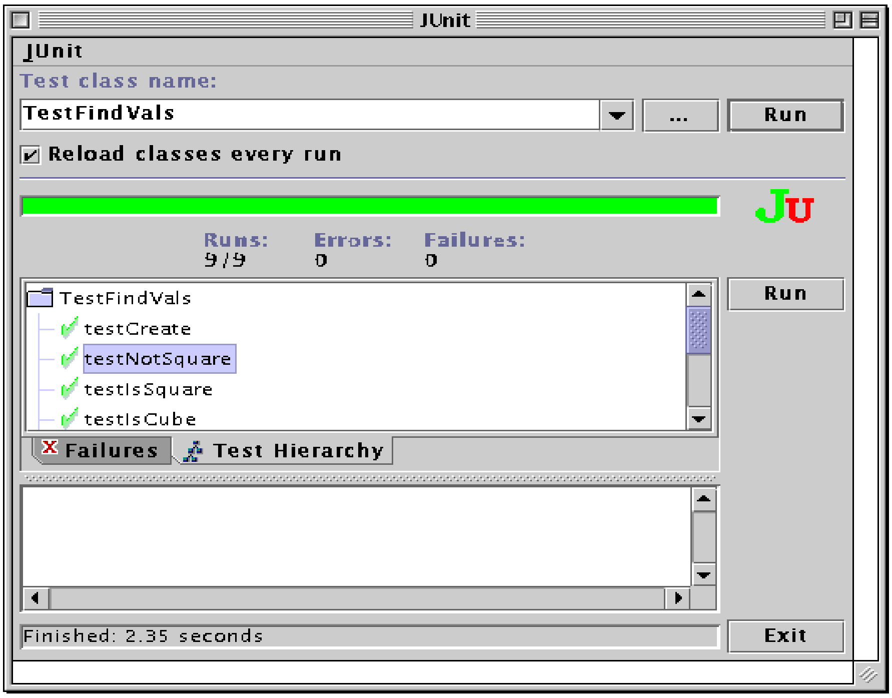
Start the testing

- **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



JUnit

Test class name:

TestFindVals

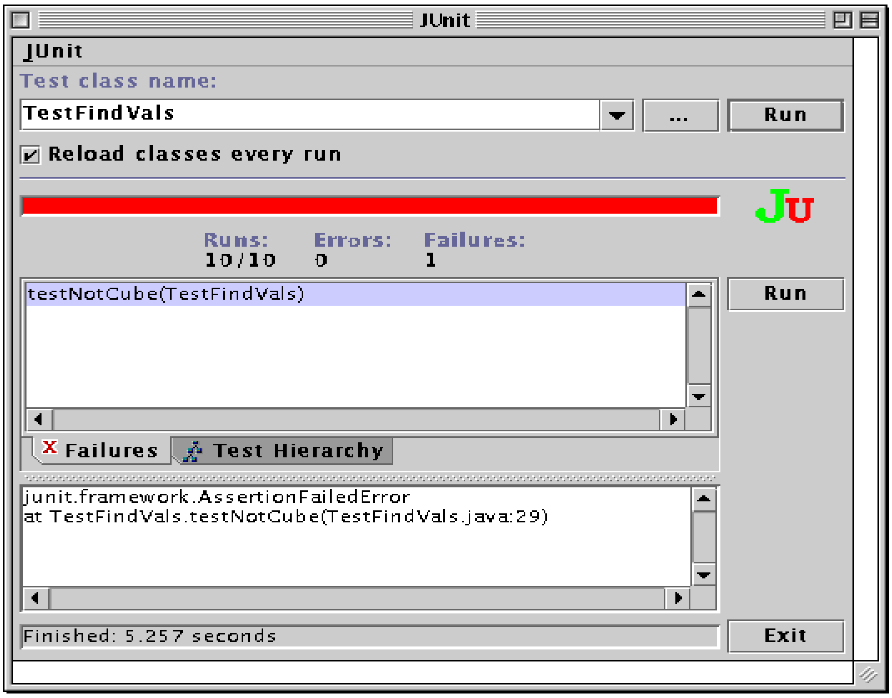
☑ Reload classes every run

Runs:	Errors:	Failures:
9/9	0	0

```
TestFindVals
  ✓ testCreate
  ✓ testNotSquare
  ✓ testIsSquare
  ✓ testIsCube
```

Finished: 2.35 seconds

Running the tests



JUnit

Test class name:

TestFindVals

☑ Reload classes every run

Runs:	Errors:	Failures:
10/10	0	1

testNotCube(TestFindVals)

```
junit.framework.AssertionFailedError
at TestFindVals.testNotCube(TestFindVals.java:29)
```

Finished: 5.257 seconds

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

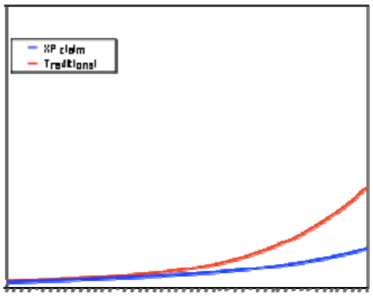
- Maybe saves you a couple seconds once or twice

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 surprises
- And its 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**

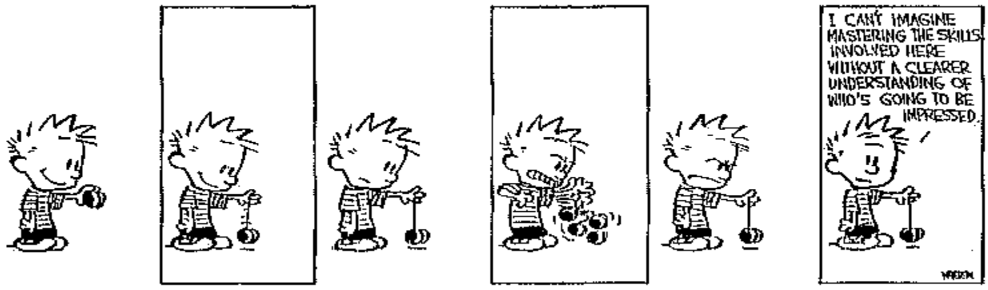
How big should a test be?

- **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


I CAN'T IMAGINE MASTERING THE SKILLS INVOLVED HERE WITHOUT A CLEARER UNDERSTANDING OF WHO'S GOING TO BE IMPRESSED.

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