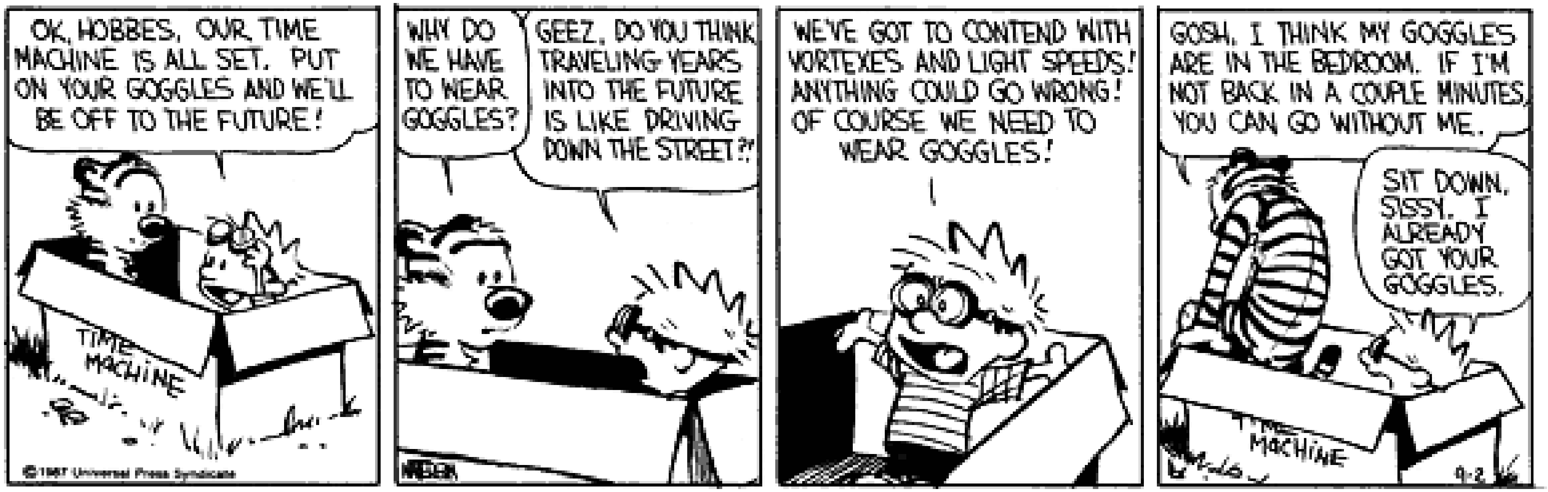
Tools & Methods


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?!
WATTERSON
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

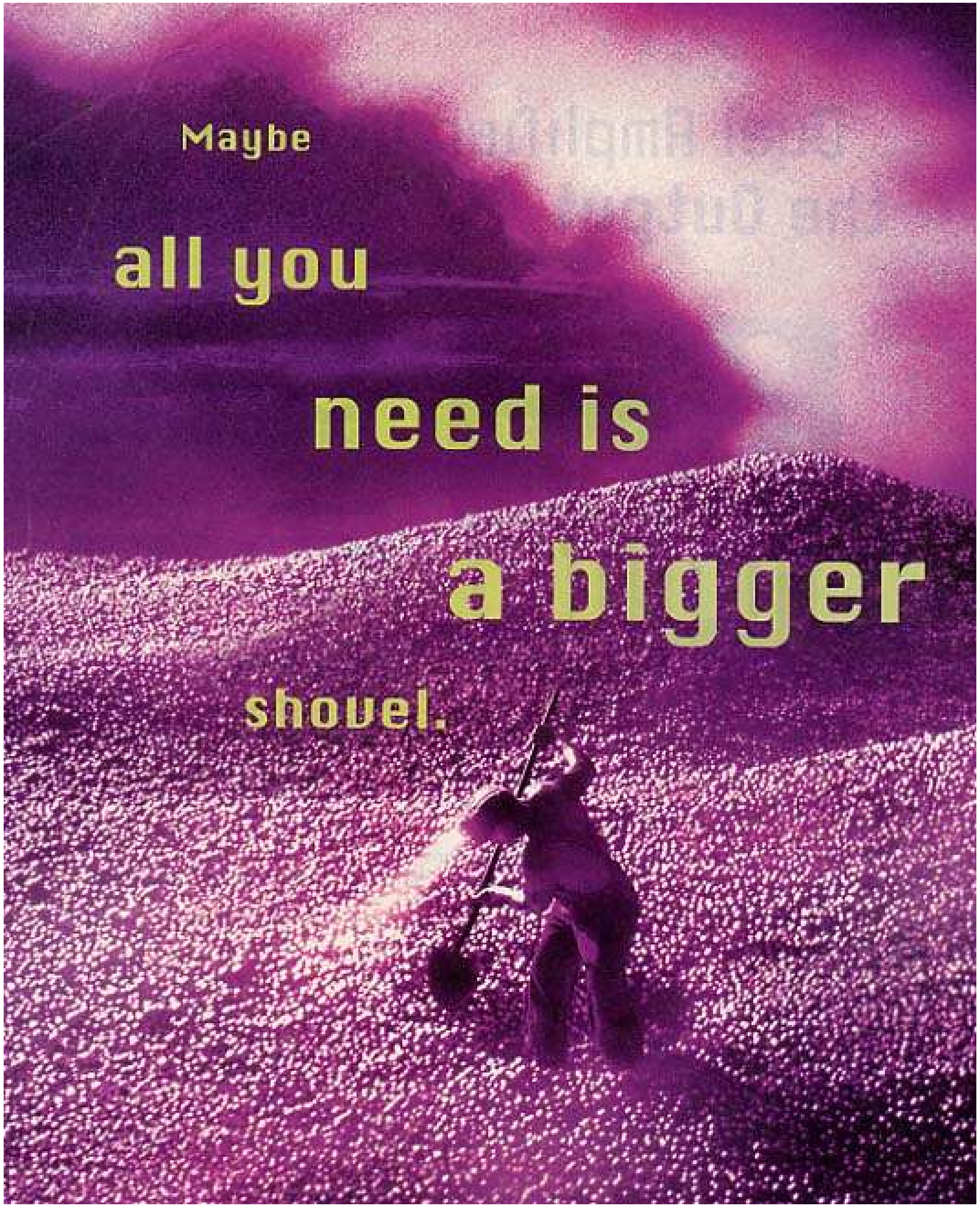
Already, I need an editor, compiler, linker, and probably a debugger

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

“The answer looks OK, lets move on.”

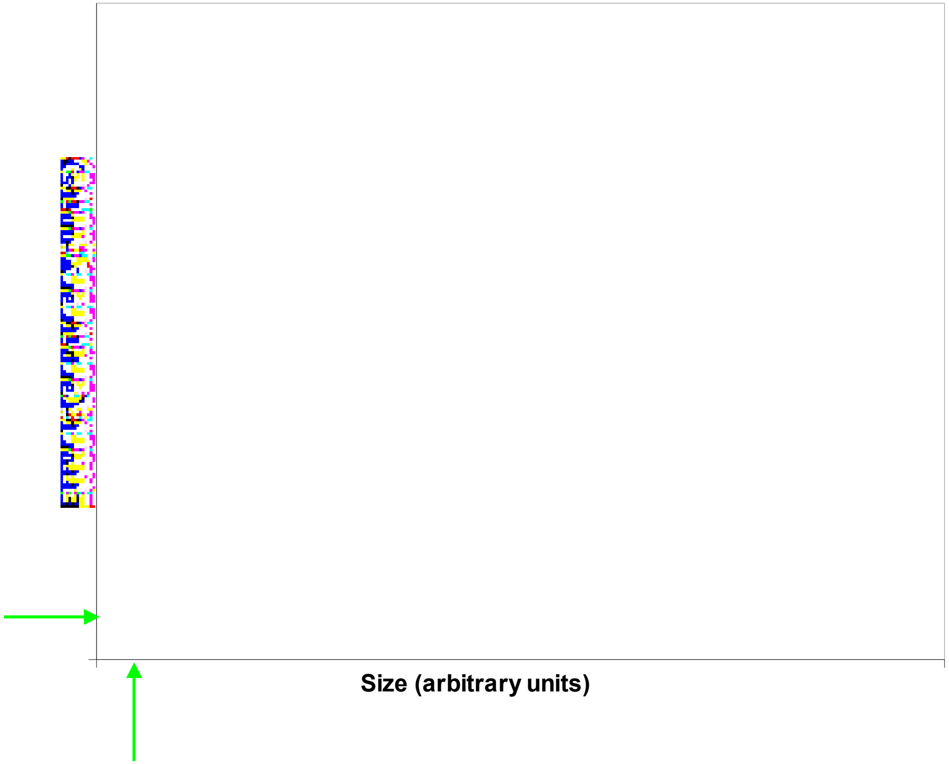
“Does anybody know where this value came from?”

“Your #%@!& code broke again!”



Projects come in different sizes

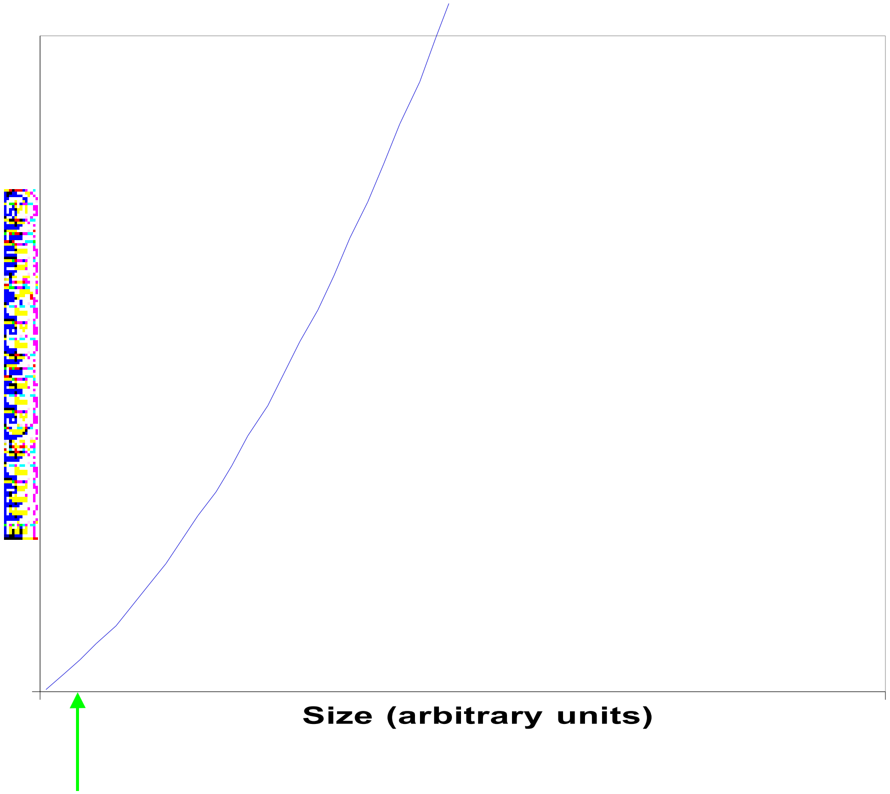
My sample program is a pretty small project!



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!

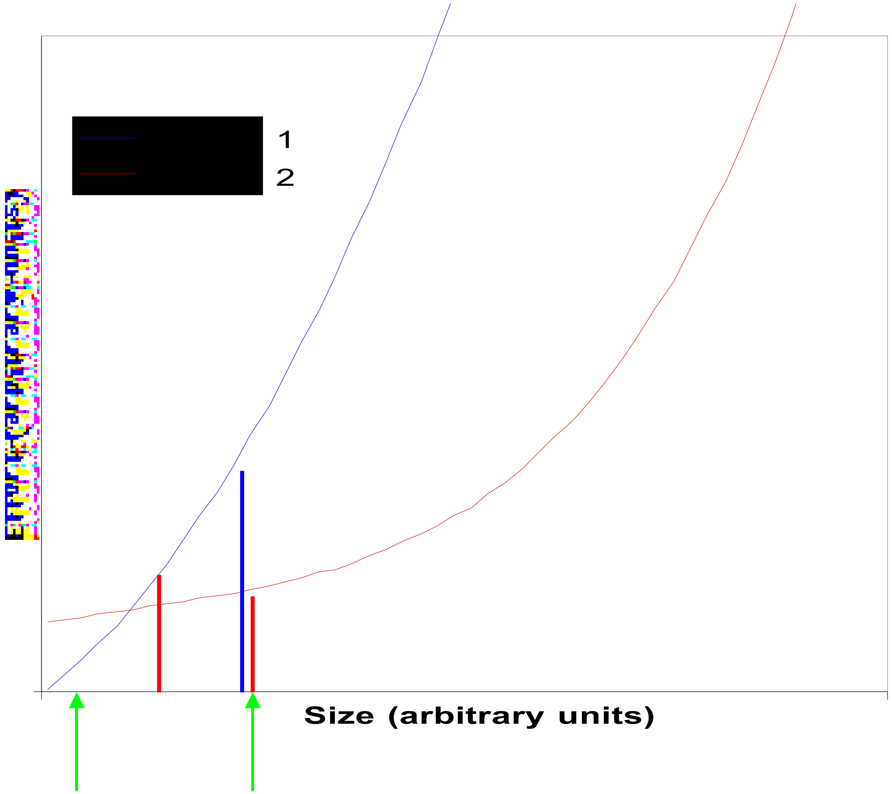
It can be done with a simple technique:

But that won’t solve larger problems well

Projects come in different sizes

A larger project may need a different approach

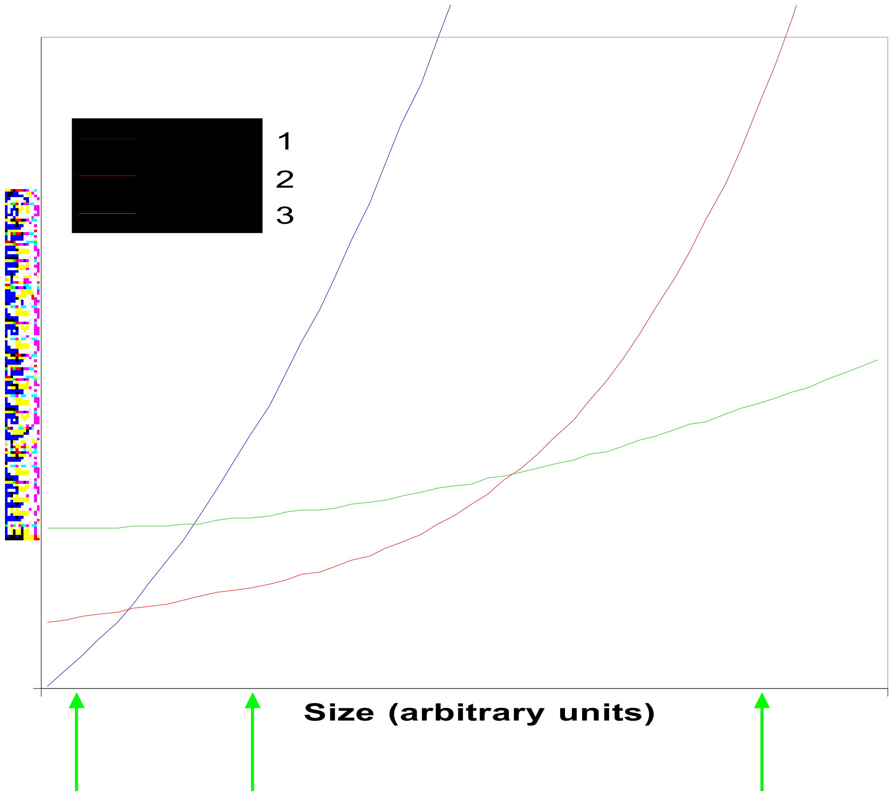
- Those tend to require more effort up front



What do you do when your project grows?

Projects come in different sizes

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



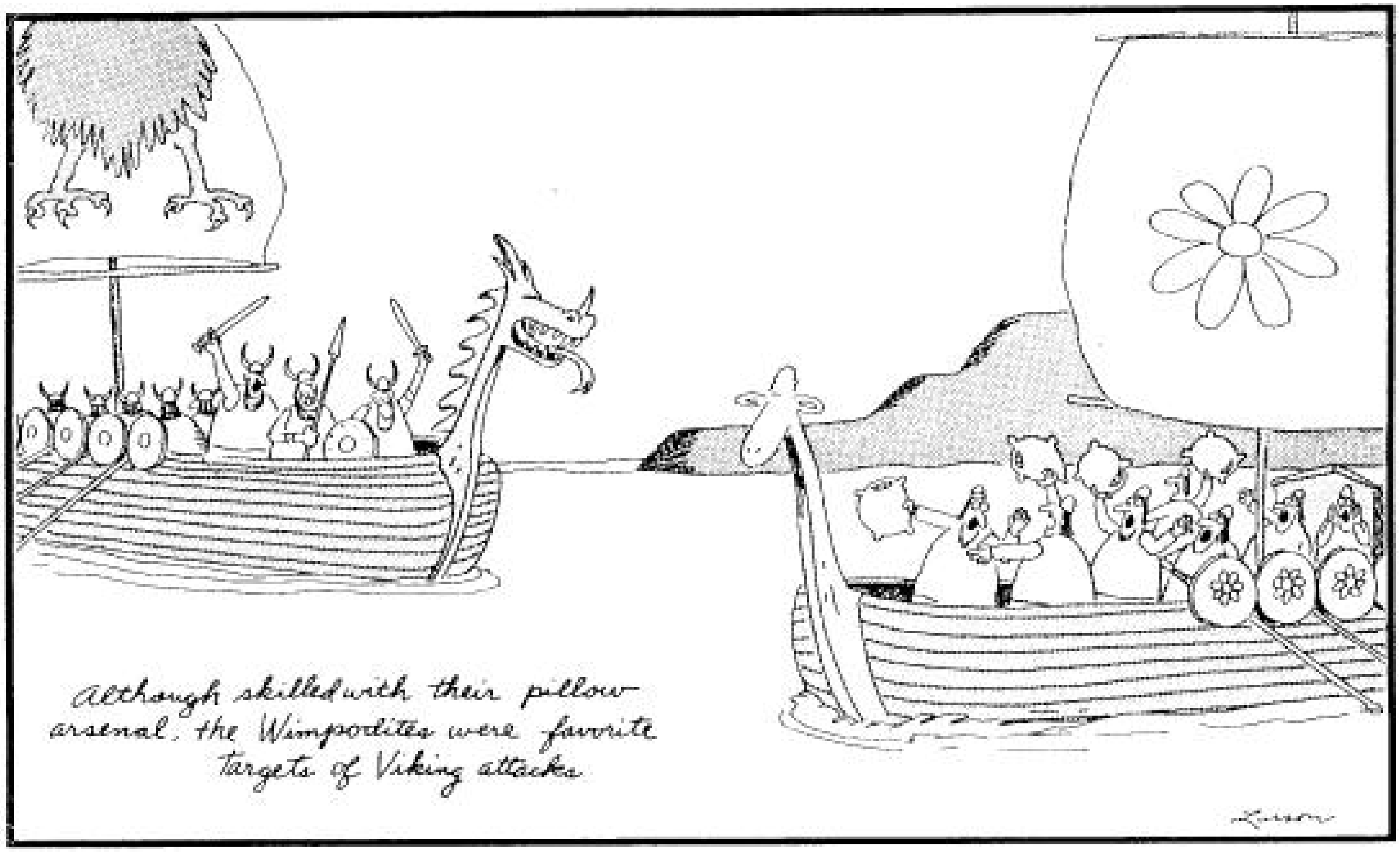
What has all this to do with us?

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

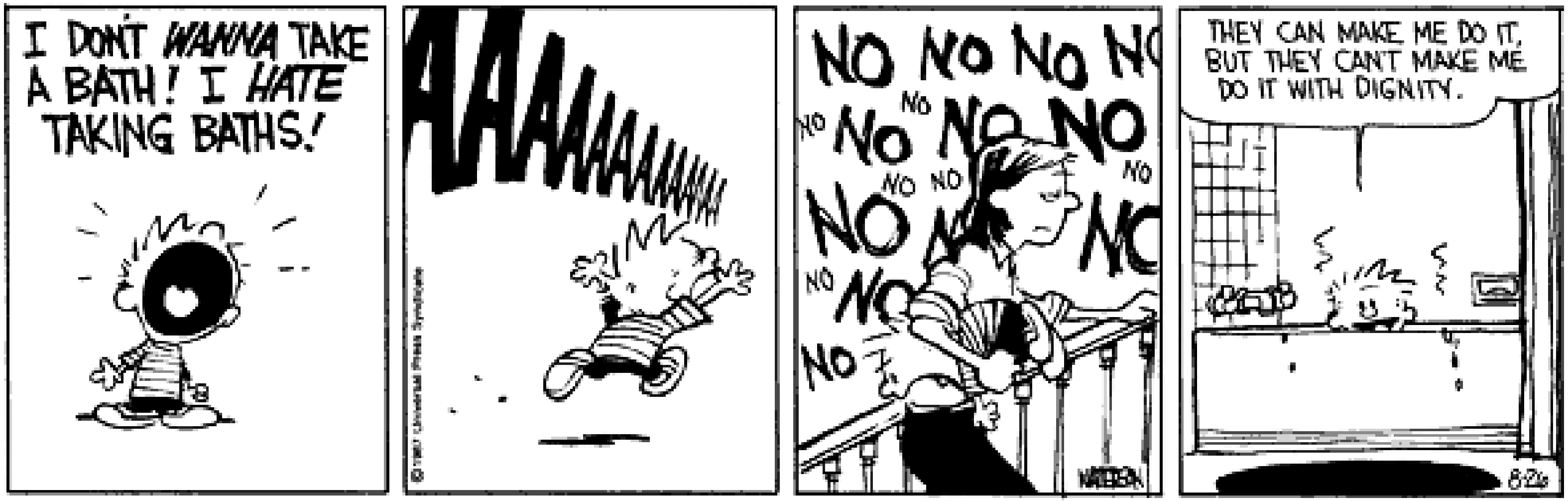
- Standards for languages, data storage, etc.

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 NO
THEY CAN MAKE ME DO IT, BUT THEY CAN'T MAKE ME DO IT WITH DIGNITY.
WATTERSON
8-26

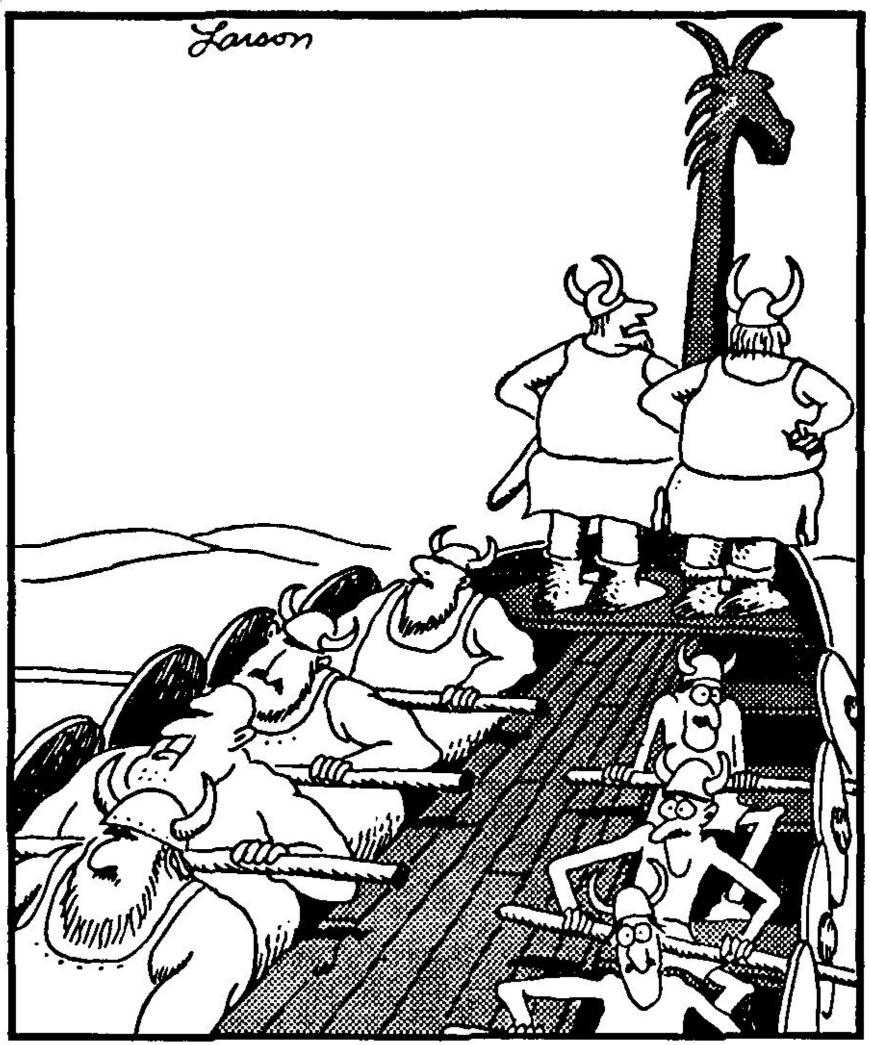
But individual effort is still important!

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

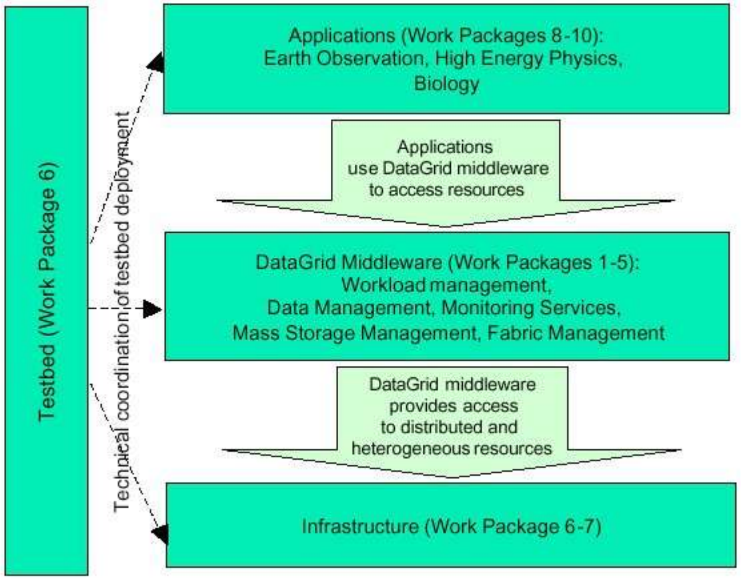
A spectrum of places to improve:

- 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 prim
        bool prime = true;
        for (int j=1; j < 10; j++) { // loop over possible fact
            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

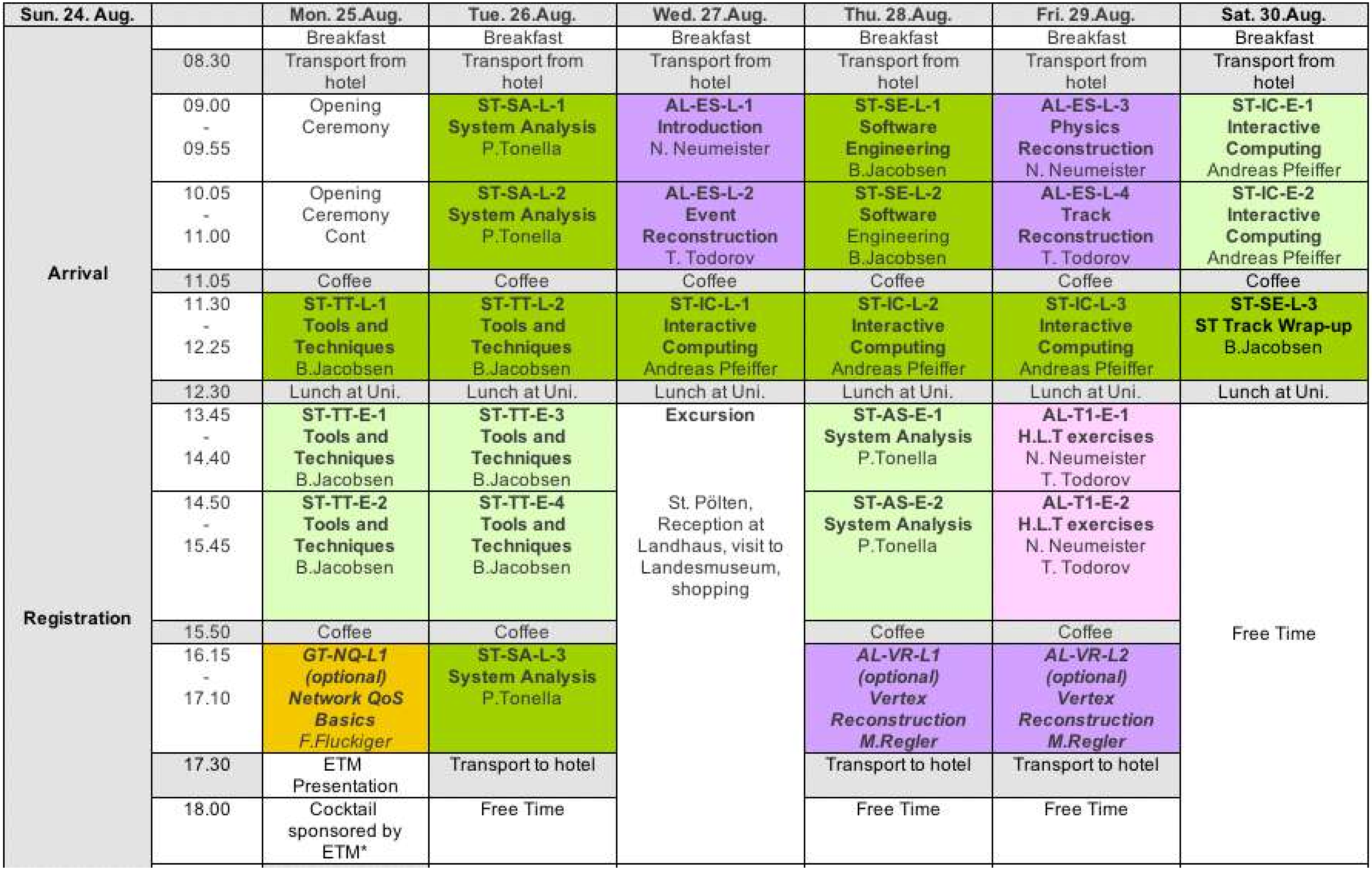
DataGrid Middleware (Work Packages 1-5):
Workload management,
Data Management, Monitoring Services,
Mass Storage Management, Fabric Management

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 for this week:



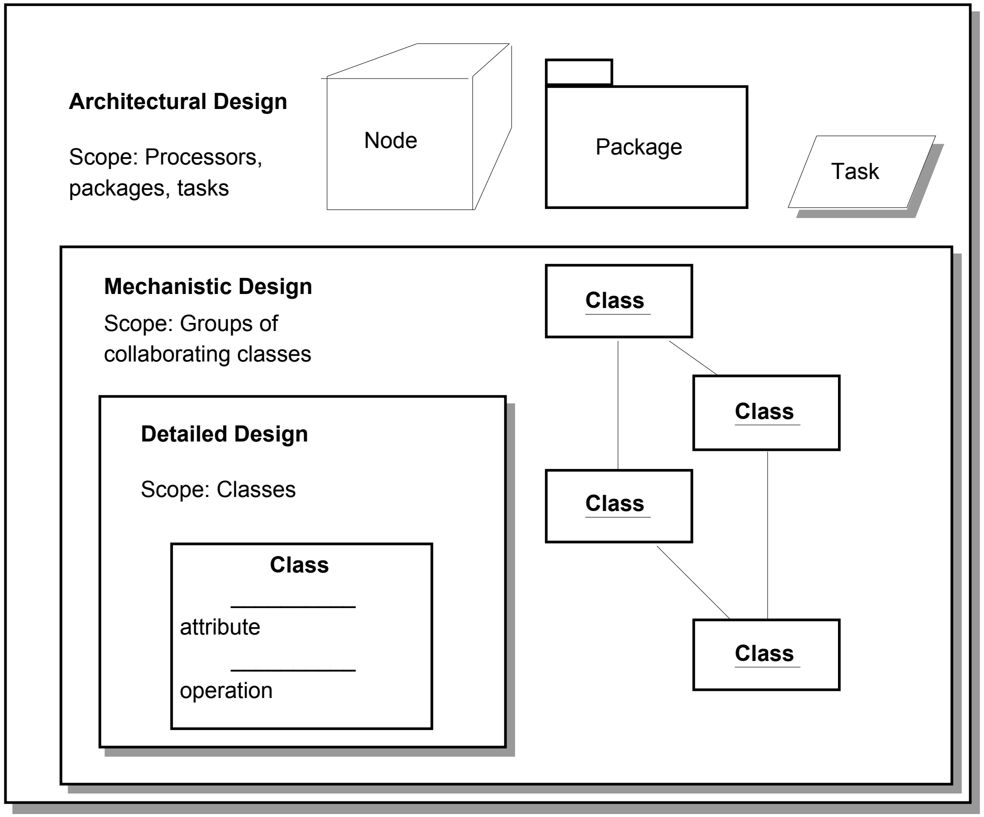
Sun. 24. Aug.		Mon. 25.Aug.	Tue. 26.Aug.	Wed. 27.Aug.	Thu. 28.Aug.	Fri. 29.Aug.	Sat. 30.Aug.
		Breakfast	Breakfast	Breakfast	Breakfast	Breakfast	Breakfast
	08.30	Transport from hotel	Transport from hotel	Transport from hotel	Transport from hotel	Transport from hotel	Transport from hotel
	09.00 - 09.55	Opening Ceremony	**ST-SA-L-1** **System Analysis** P.Tonella	**AL-ES-L-1** **Introduction** N. Neumeister	**ST-SE-L-1** **Software Engineering** B.Jacobsen	**AL-ES-L-3** **Physics Reconstruction** N. Neumeister	**ST-IC-E-1** **Interactive Computing** Andreas Pfeiffer
	10.05 - 11.00	Opening Ceremony Cont	**ST-SA-L-2** **System Analysis** P.Tonella	**AL-ES-L-2** **Event Reconstruction** T. Todorov	**ST-SE-L-2** **Software** Engineering B.Jacobsen	**AL-ES-L-4** **Track Reconstruction** T. Todorov	**ST-IC-E-2** **Interactive Computing** Andreas Pfeiffer
Arrival	11.05	Coffee	Coffee	Coffee	Coffee	Coffee	Coffee
	11.30 - 12.25	**ST-TT-L-1** **Tools and Techniques** B.Jacobsen	**ST-TT-L-2** **Tools and Techniques** B.Jacobsen	**ST-IC-L-1** **Interactive Computing** Andreas Pfeiffer	**ST-IC-L-2** **Interactive Computing** Andreas Pfeiffer	**ST-IC-L-3** **Interactive Computing** Andreas Pfeiffer	**ST-SE-L-3** **ST Track Wrap-up** B.Jacobsen
	12.30	Lunch at Uni.	Lunch at Uni.	Lunch at Uni.	Lunch at Uni.	Lunch at Uni.	Lunch at Uni.
	13.45 - 14.40	**ST-TT-E-1** **Tools and Techniques** B.Jacobsen	**ST-TT-E-3** **Tools and Techniques** B.Jacobsen	**Excursion**	**ST-AS-E-1** **System Analysis** P.Tonella	**AL-T1-E-1** **H.L.T exercises** N. Neumeister T. Todorov	
	14.50 - 15.45	**ST-TT-E-2** **Tools and Techniques** B.Jacobsen	**ST-TT-E-4** **Tools and Techniques** B.Jacobsen	St. Pölten, Reception at Landhaus, visit to Landesmuseum, shopping	**ST-AS-E-2** **System Analysis** P.Tonella	**AL-T1-E-2** **H.L.T exercises** N. Neumeister T. Todorov	
Registration	15.50	Coffee	Coffee		Coffee	Coffee	Free Time
	16.15 - 17.10	***GT-NQ-L1*** ***(optional)*** ***Network QoS Basics*** *F.Fluckiger*	**ST-SA-L-3** **System Analysis** P.Tonella		***AL-VR-L1*** ***(optional)*** ***Vertex Reconstruction*** ***M.Regler***	***AL-VR-L2*** ***(optional)*** ***Vertex Reconstruction*** ***M.Regler***	
	17.30	ETM Presentation	Transport to hotel		Transport to hotel	Transport to hotel	
	18.00	Cocktail sponsored by ETM*	Free Time		Free Time	Free Time	

Design

System architecture

Individual project

Specific task



“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

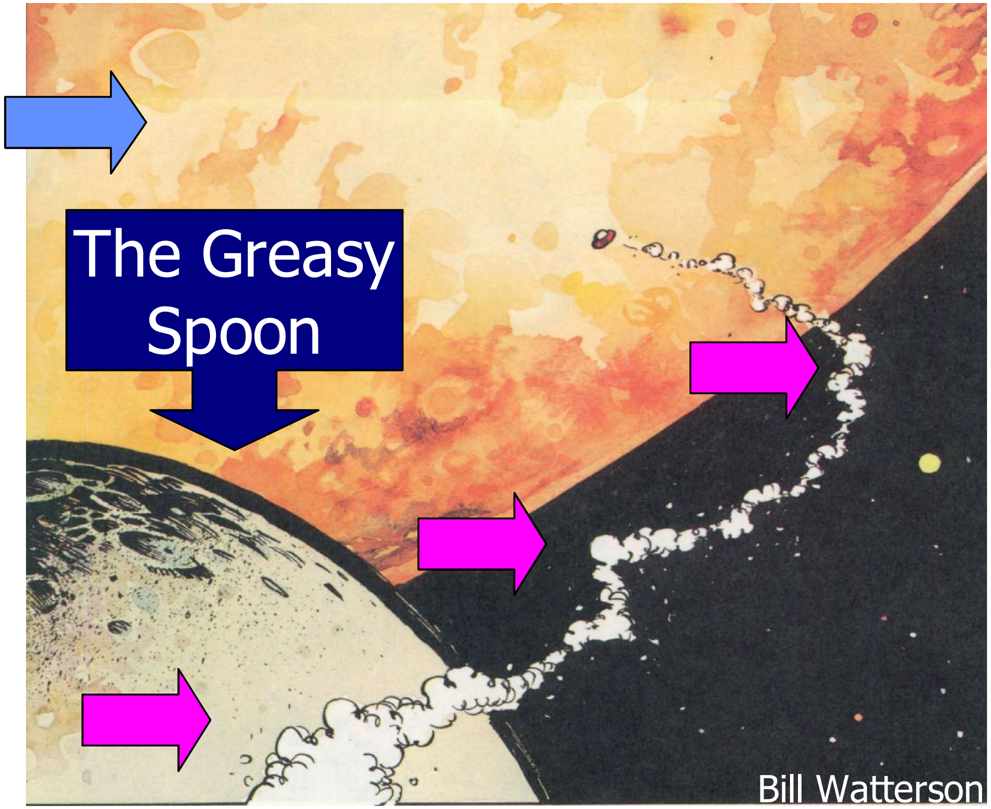
decide which planet to fly to

Mechanistic design

select the flight path

Detailed design

choose where to have lunch



Bill Watterson

Architectural design

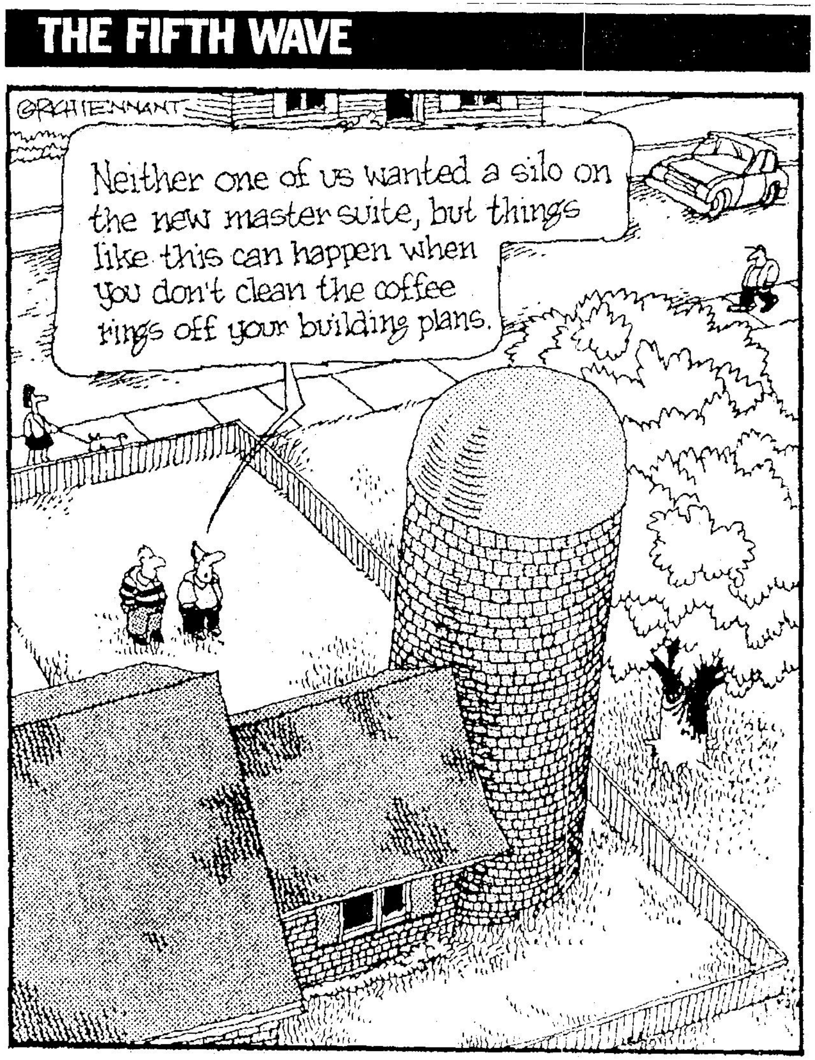
Goals

- 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

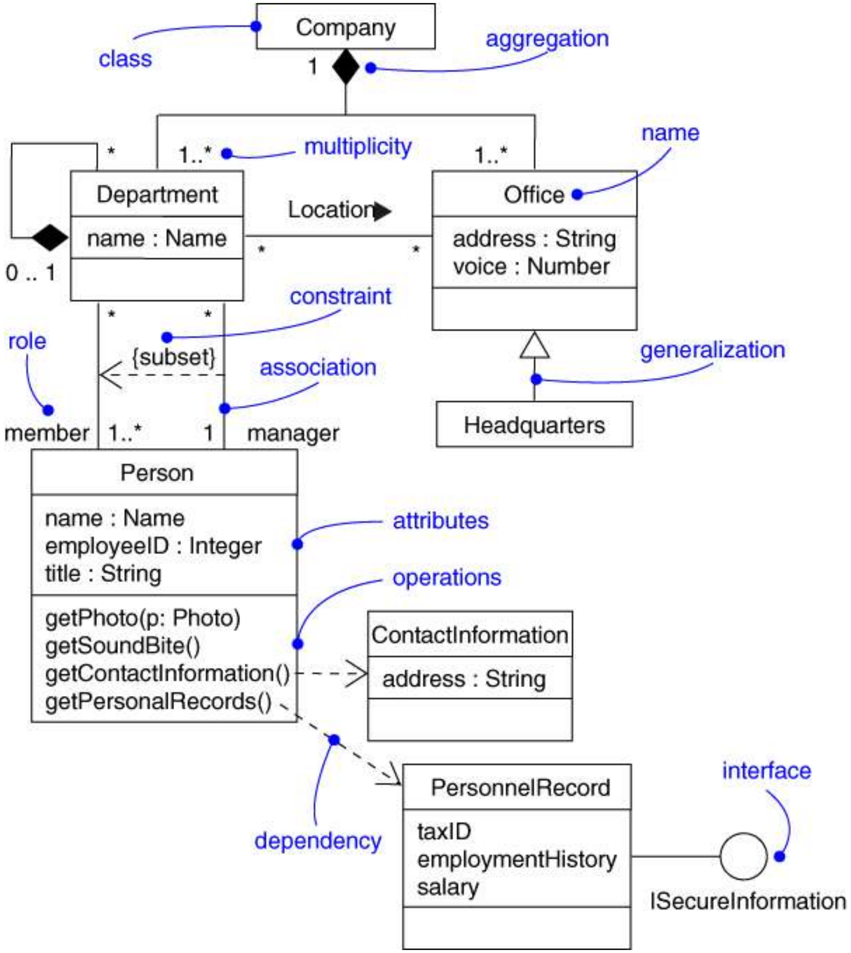
Mechanistic Design

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

Class Diagram

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



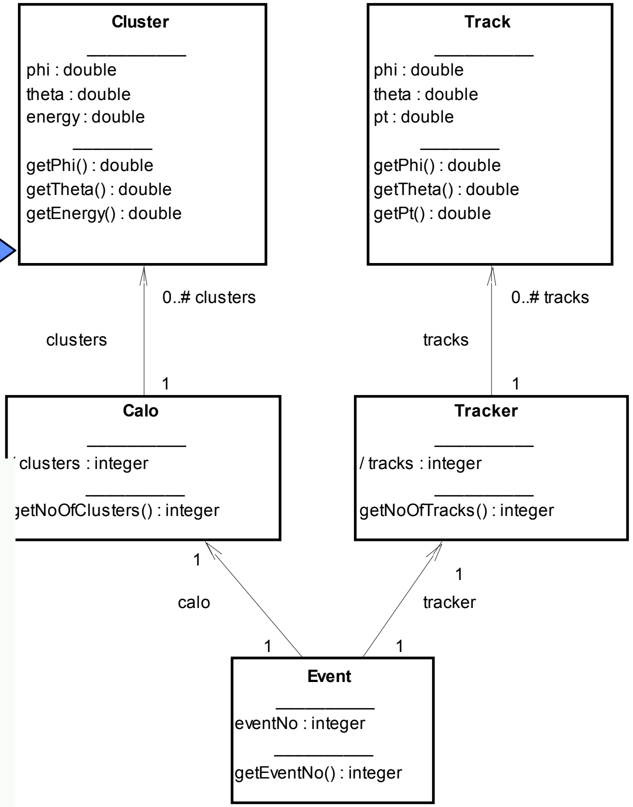
Rational Software Corporation

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


Cluster

phi : double
theta : double
energy : double

getPhi() : double
getTheta() : double
getEnergy() : double

Track

phi : double
theta : double
pt : double

getPhi() : double
getTheta() : double
getPt() : double

0..# clusters
clusters
1

0..# tracks
tracks
1

Calo

' clusters : integer

jetNoOfClusters() : integer

Tracker

/ tracks : integer

getNoOfTracks() : integer

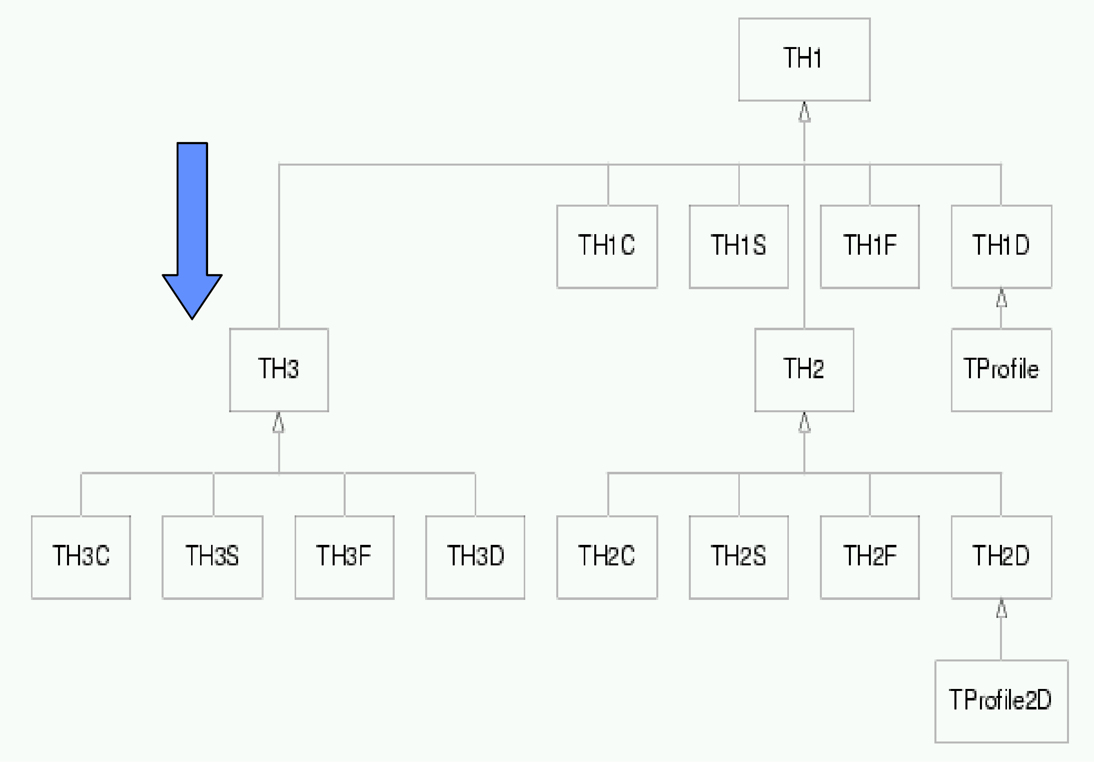
1
calo
1

1
tracker
1

Event

eventNo : integer

getEventNo() : integer



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

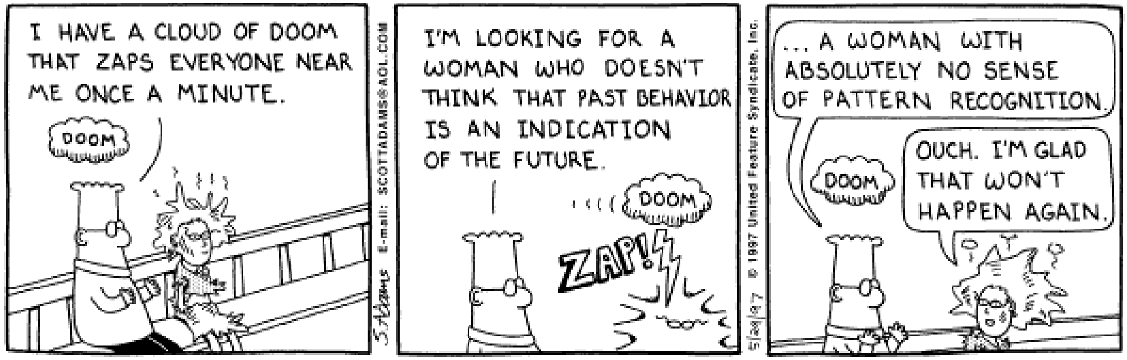
- And hence enjoyable
- Not always clear when you should stop

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
S.Adams
E-mail: SCOTTADAMS@AOL.COM
I'M LOOKING FOR A WOMAN WHO DOESN'T THINK THAT PAST BEHAVIOR IS AN INDICATION OF THE FUTURE.
DOOM
ZAP!
5/20/97
© 1997 United Feature Syndicate, Inc.
... A WOMAN WITH ABSOLUTELY NO SENSE OF PATTERN RECOGNITION.
DOOM
OUCH. I'M GLAD THAT WON'T HAPPEN AGAIN.

Addressing these themes:

Tools you can use

Knowing what you’ve done - CVS

Knowing whether it works - JUnit

CVS Source Code Management

Maintains a repository of text files

- Allows users to check in and check out changed text
- Old code remains available
  - Each checked-in change defines a new revision
  - You can retrieve, ask for differences with any of them
- Revisions can be tagged for easy reference

Similar in concept to RCS, CMS, other products

Big advantage: checkout is not exclusive

- More than one developer can have the same file checked out
- Developers can control their own use of the code for read, write
- Changes can come from multiple sources
- CVS handles (most) of the conflict resolution

Key tool for large collaborations!

- But also an important tool for individuals

Simple usage: checkout and update

Getting a copy of the most recent contents of a package Foo:

```
cvs checkout Foo
```

Getting a copy of version (tag) V00-02-23 of a package Foo:

```
cvs checkout -r V00-02-23 Foo
```

These produce fully editable Foo directories, etc

To update a directory to the most recent contents:

```
cvs update -A
```

To see what an update will change, without actually changing

```
cvs -n update -A
```

Update flags:

- U update   M modified   A added
- C conflict   ?   unknown   D deleted

Committing changes back to the repository

To put your changes back into the repository:

- Merge in any changes since your checkout

```
cvs update -A
```

- commit:

```
cvs commit
```

Many options:

- Specify comment for logs from command line
- Commit only one file
- Control processing of subdirectories

Possible failures

- Can’t get a temporary lock on the repository
- Conflict during update

Adding and removing files

To tell CVS a new file exists:

- First create the file, then

```
cvs add <name>
cvs commit
```

- Nothing changes in the repository until the commit

To tell CVS a file is no longer needed

- First delete the file, then

```
cvs rm <name>
cvs commit
```

- Nothing changes in the repository until the commit

Labeling particular contents for later

To add a particular label to certain contents:

- Make sure that everything is in the repository

```
cvs update
cvs commit
```

- Tell CVS to add a tag to the current contents

```
cvs tag <string>
```

Tags are an easy way to communicate with your colleagues

- “I just fixed that in jake20010924a, give it a try”
- This bug is back in V00-03-04, I thought it was fixed in V00-03-02”

Web based tools exist for seeing what changed, who changed it, etc

Conflict resolution & parallel development

Its rare for developers to really conflict by changing the same line

- Usually only one person working on a particular piece of functionality
- And people working on the same thing should talk to each other!
- Conflicts happen most often during migrations of the code

When it happens, CVS can’t figure out how to cope

- Marks both sets of changed lines with markers

```
<<<<<<<<<<<<<<<<<
One content
=================
Other content
>>>>>>>>>>>>>>>>>
```

- User has to edit this to select one or other, or combine

Really not a significant problem

- Though we will provoke it during exercises

Behind the curtain

The repository contains *,v files

- Each contains some version info at the front,
- followed by the most recent contents
- followed by enough patch information to recreate old contents

Deleted files are stored in the “Attic” directory

Each CVS-controlled directory has a CVS subdirectory

- Contains various files used by CVS
- Don’t touch!

How CVS find changes

Triple compare

- The contents you have now
- The contents you checked out
- The current contents of the repository

CVS calculates two sets of changes:

- From second and third, it finds changes to the repository
- From first and second, it finds your changes

So long as these don't overlap, there's no problem merging them in

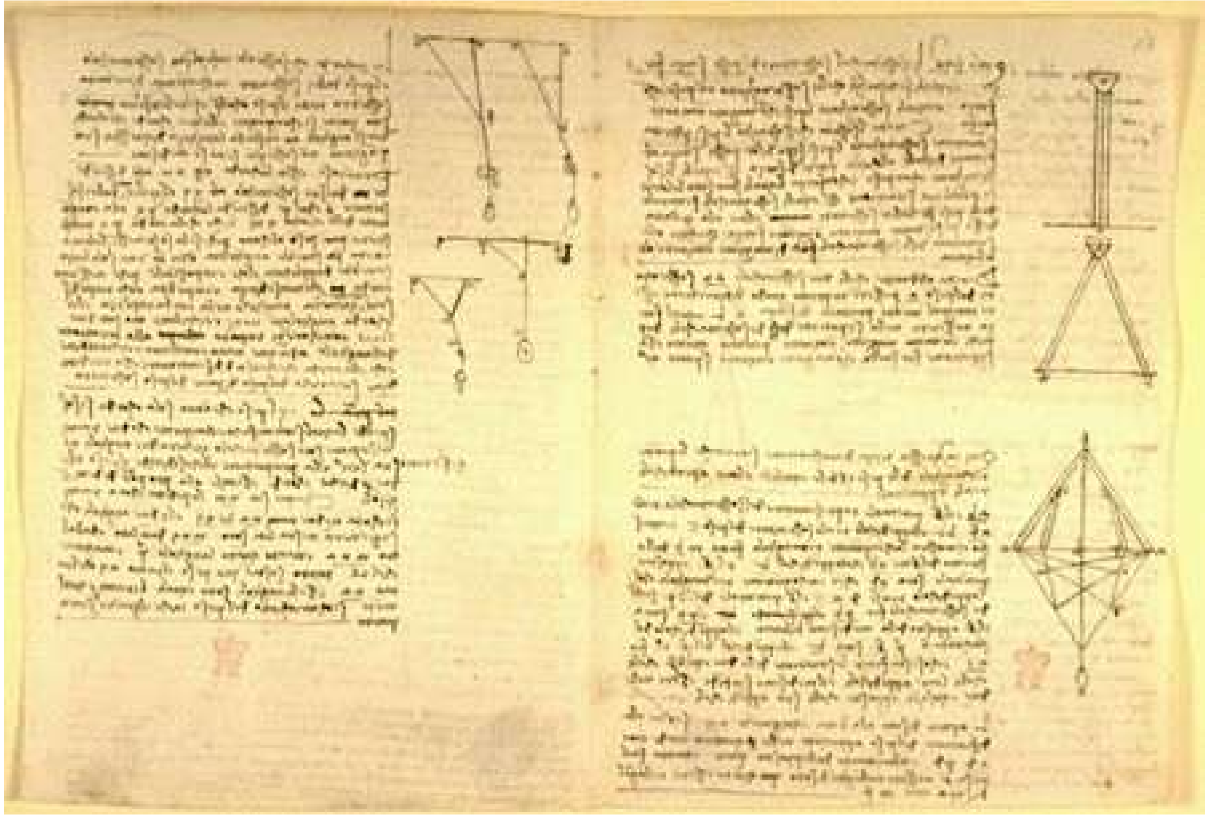
CVS thinks that any change that it detects is deliberate

- If you edit a file to remove changes, it will let you check it in
- If you copy an old version into a directory, it will let you check it in

Since CVS does not tag the file contents, copying files from one directory to another is a time-bomb

- CVS thinks it sees deliberate changes on commit, and the old version become the "current contents"

What's in it for you?



Science, medicine, even football use a notebook as a basic tool

- What you did when
- Why you did it
- What happened then

CVS can provide that

Commit, tag, update operations are cheap, logged, carry comments

Use that as your record of progress

- Commit each piece as you do it
- Spend a couple seconds on a useful comment

  "Added undo tool, next will use it from Frabitzoid"

  "Now conserves momentum"

  "Now ready for energy test cases"

Use tags to capture important states

- Tag each time it's basically working

  ```
  cvs tag jake-copy-works
  ```

- Tag to share with a coworker

  ```
  cvs tag jake20030828a
  ```

Not a heavyweight action!

Now what have I done?

It worked just minutes ago…

```
cvs diff Foo.java
```

- Can also do entire directories, etc.

How did I do that last time?

```
cvs diff -D 6-Jun-2003 -D 12-Jun-2003
cvs diff -r 1.2 -r 1.3
cvs diff -r jake-copy-works  -r jake-added-mass
```

OK, that was a bad idea

Everybody makes mistakes

- Key question: how hard to fix them?

This time I won't screw up! I won't, I won't, I won't, I won't...

Roger screws up.

Can remove changes:

- cvs update -j jake-copy-works -j jake-added-mass

Even if there are more recent changes!

- CVS uses its three-file diff method to do this
- If there are conflicts, you’ll have some hand edits to do

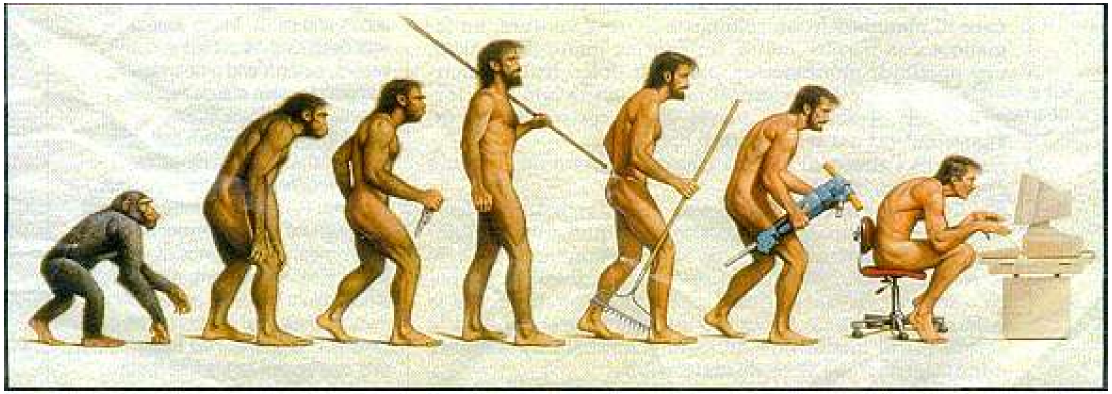
Don’t forget to commit the resulting changes back!

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

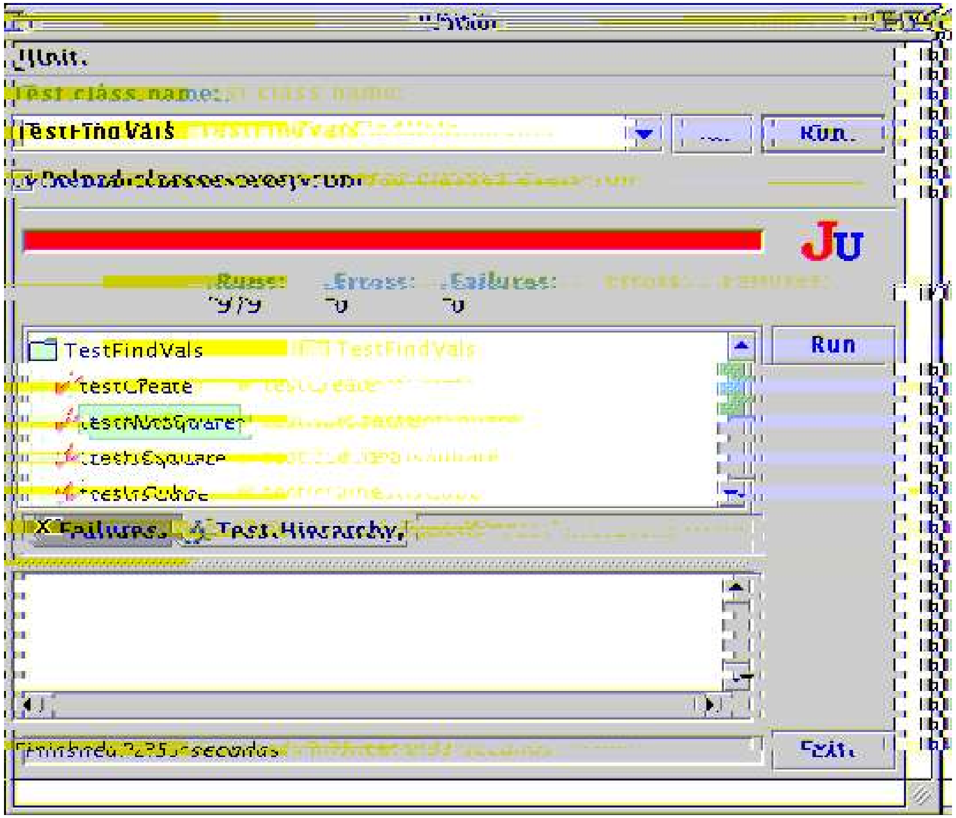
- Gets boring fast!
- How often are you willing to do this?

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

- Can become ungainly

Most useful: A test framework

- Great feedback
- Better control over testing



TestFindVals

Runs: 9/9 Errors: 0 Failures: 0

TestFindVals
testCreate
testNotSquare
testIsSquare
testIsCube

Finished: 2.253 seconds

Testing Frameworks: CppUnit, Junit, et al

To test a function:

```
public class FindVals {
// Test 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

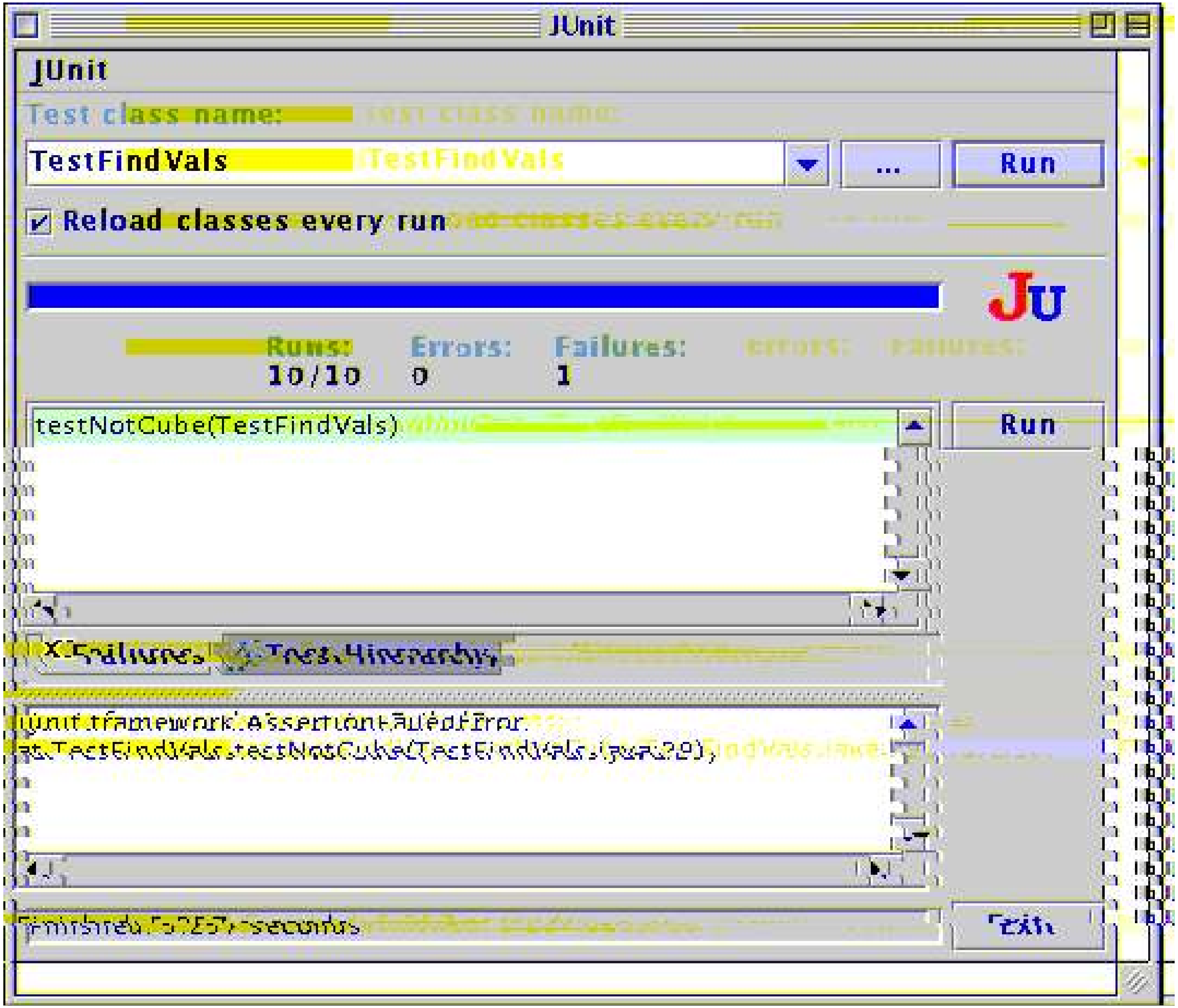
Start the testing

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

Invoke tests for my class

And that’s it!

Running the tests



Test class name: TestFindVals

☑ Reload classes every run

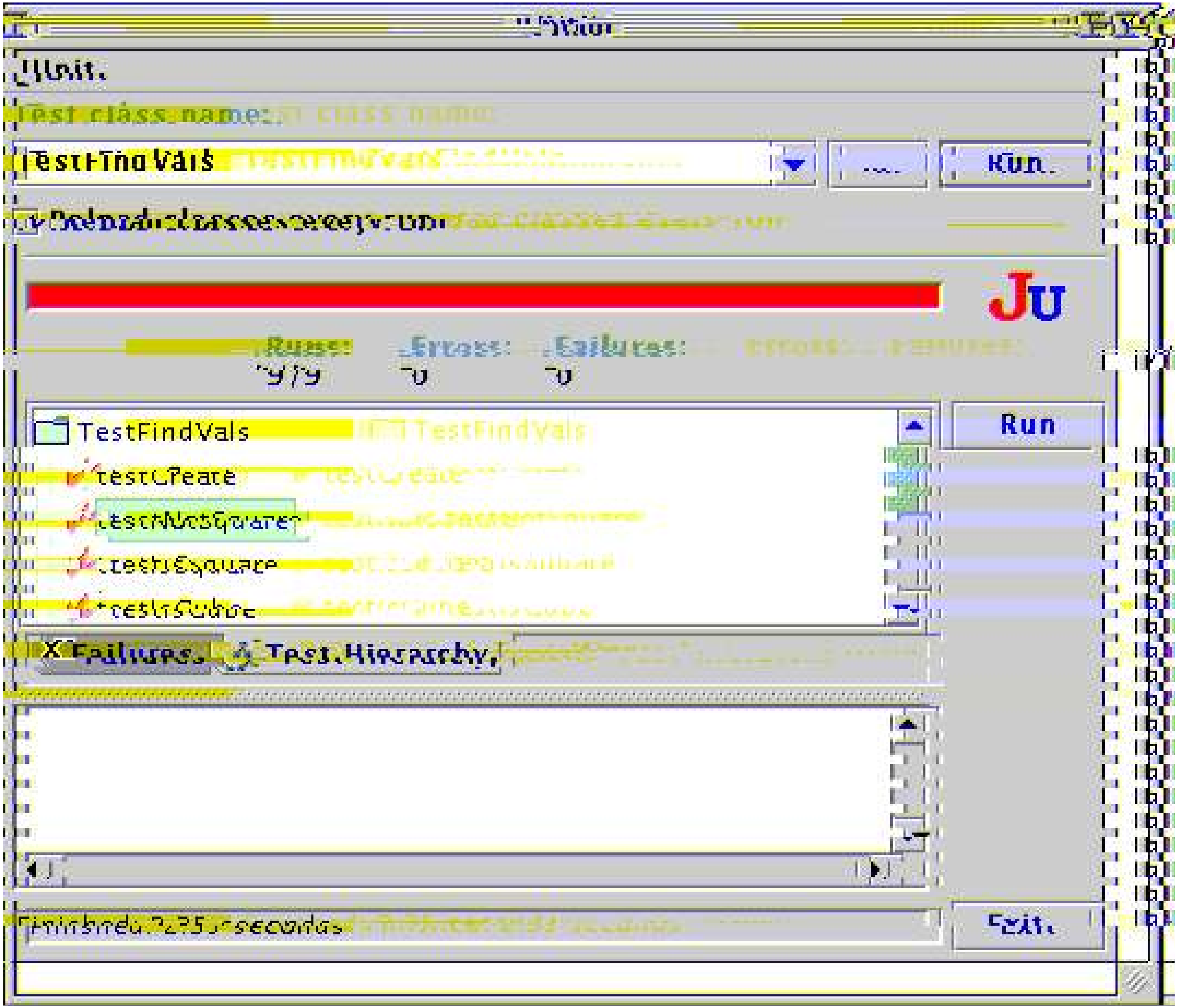
Runs:	Errors:	Failures:
10/10	0	1

testNotCube(TestFindVals)

```
junit.framework.AssertionFailedError:
at TestFindVals.testNotCube(TestFindVals.java:20)
```

Finished: 0.257 seconds

Running the tests



Test class name: TestFindVals

☑ Reload classes every run

Runs: 9/9 Errors: 0 Failures: 0

TestFindVals
- testCreate
- testNotSquare
- testIsSquare
- testIsCube

Finished: 2.35 seconds

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

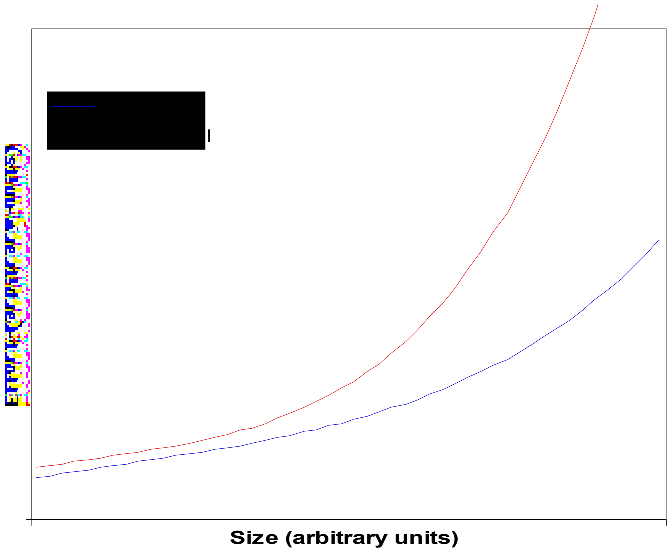
- A set of tests is definitive documentation for what the code does
- Alternating between writing tests and code keeps the work incremental

  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

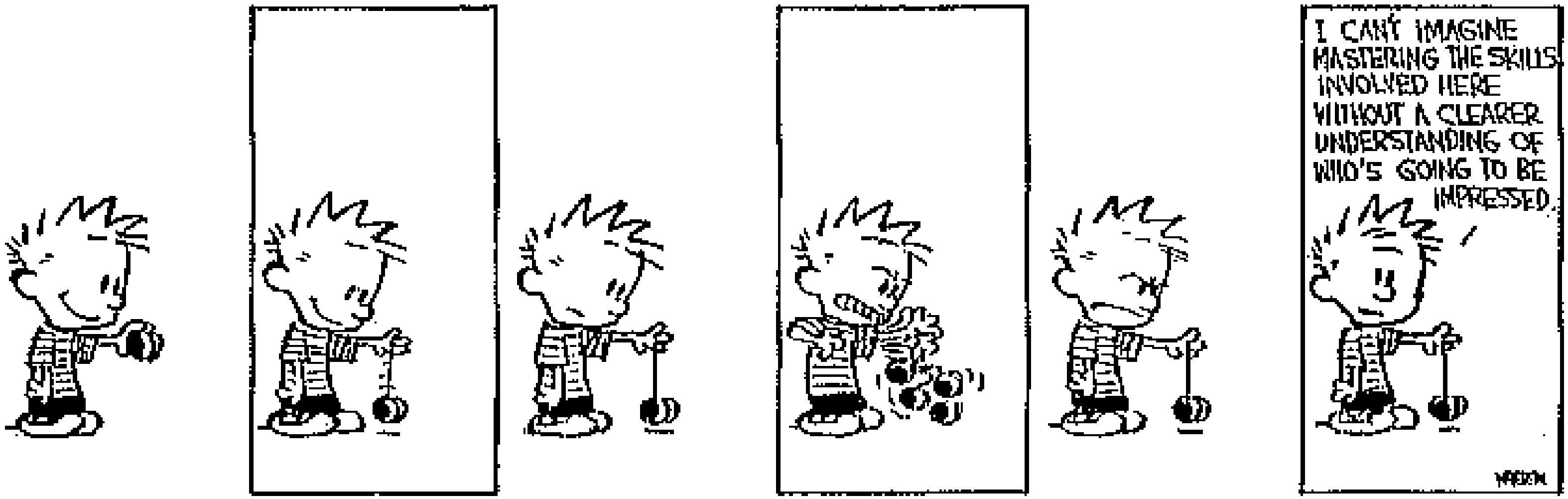
How big should a test be?

- 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.
WATTERSON

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

Today's Exercises

1) Simple use of CVS

2) More advanced CVS, showing how conflicts are handled

3) Demonstrate that *everybody* can edit the same file successfully!

5) Demonstration of a test framework

6) Practice debugging using a test framework

Instruction sheets are available via web browser at

file:~jake/index.html

If you get past these, feel free to move on to tomorrow's exercises (see the instructions page)