

Beyond the Mouse – A Short Course on Programming

1. Thinking programs

Ronni Grapenthin

Geophysical Institute, University of Alaska
Fairbanks

September 10, 2009

YOU'LL NEVER FIND A
PROGRAMMING LANGUAGE
THAT FREES YOU FROM
THE BURDEN OF
CLARIFYING
YOUR IDEAS.



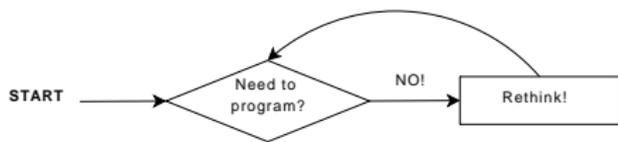
"The Uncomfortable Truths Well",
<http://xkcd.com/568> (April 13, 2009)

Outline

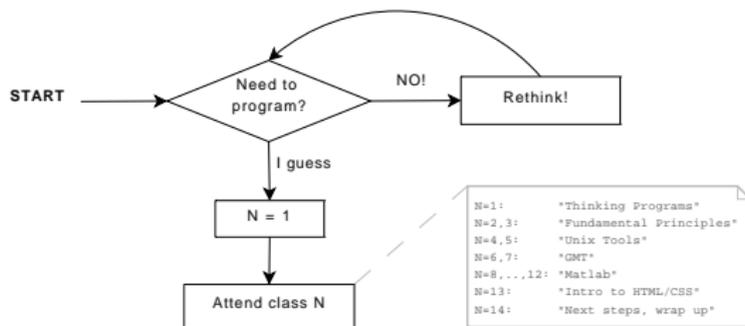
- 1 Overview and Philosophies
- 2 Thinking programs
- 3 Building programs
- 4 Summary

- 1 Overview and Philosophies
- 2 Thinking programs
- 3 Building programs
- 4 Summary

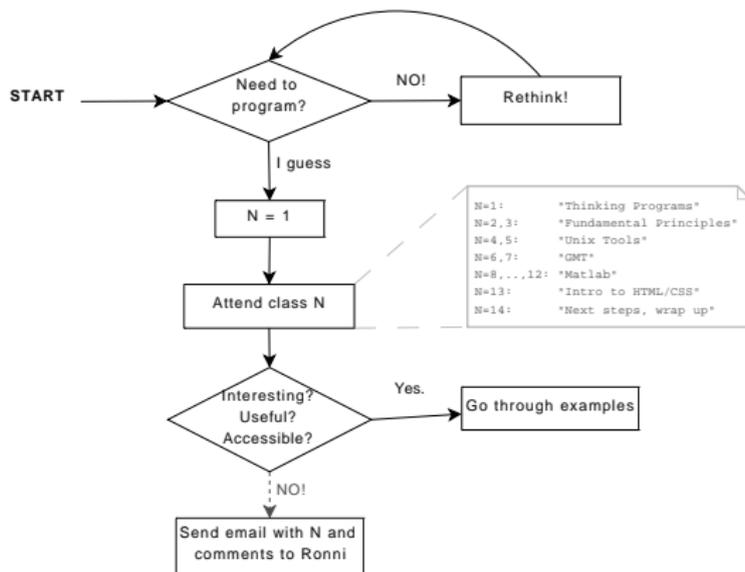
The Program ...



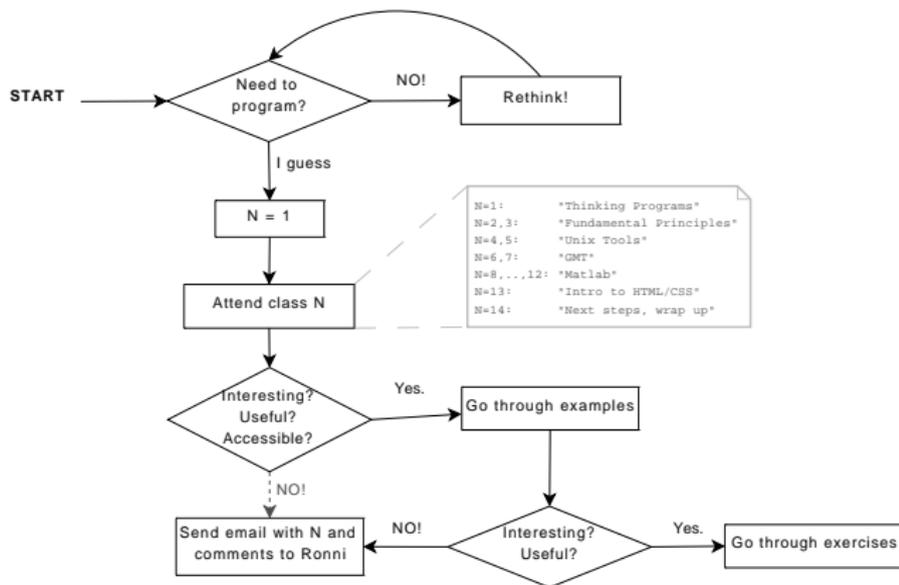
The Program ...



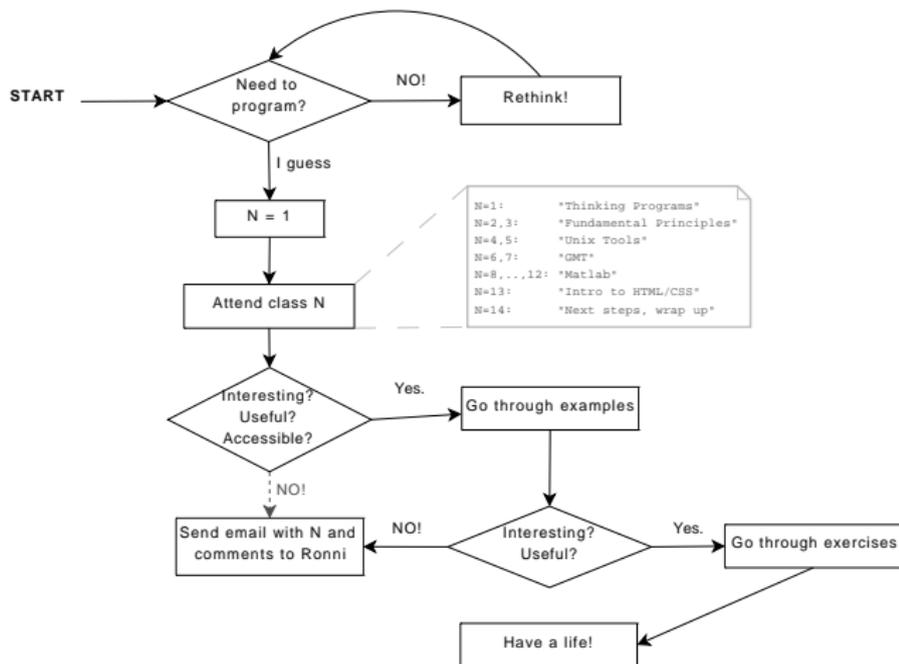
The Program ...



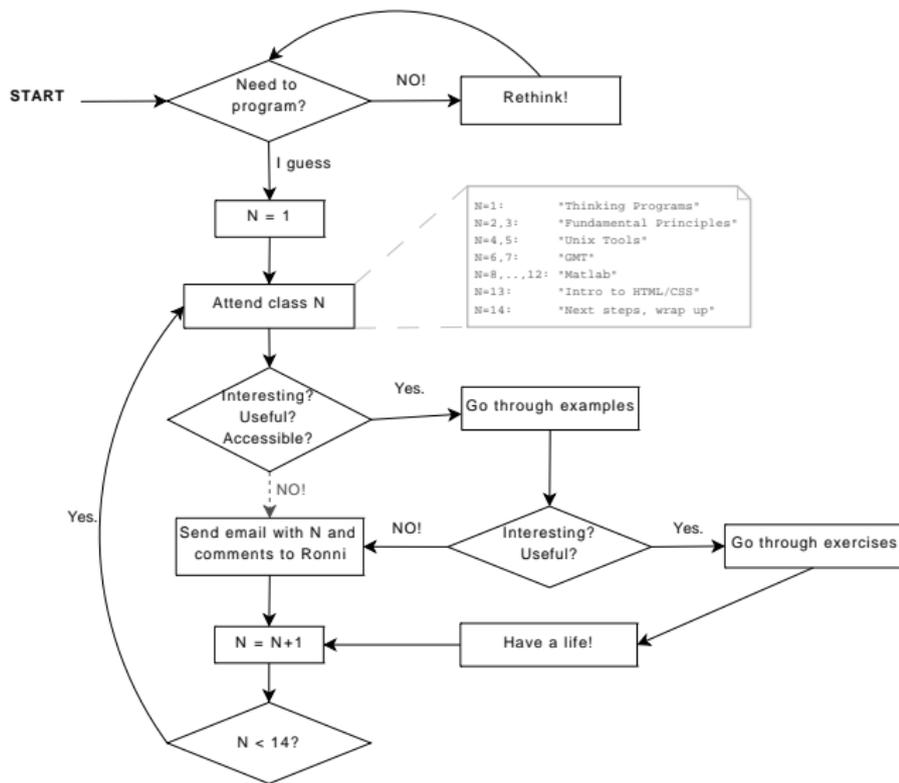
The Program ...



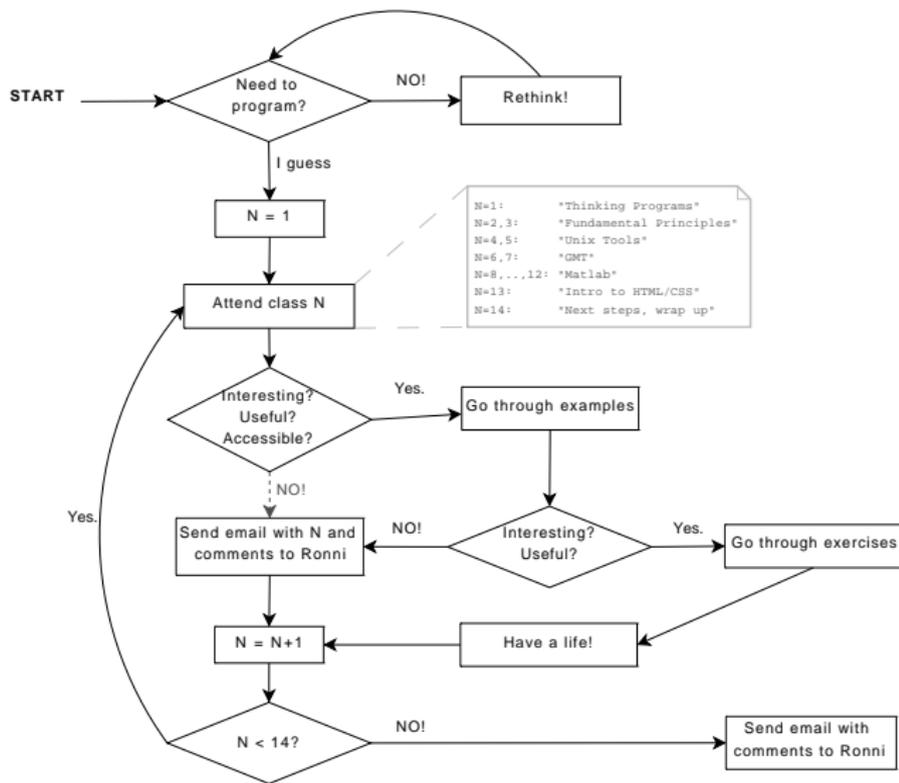
The Program ...



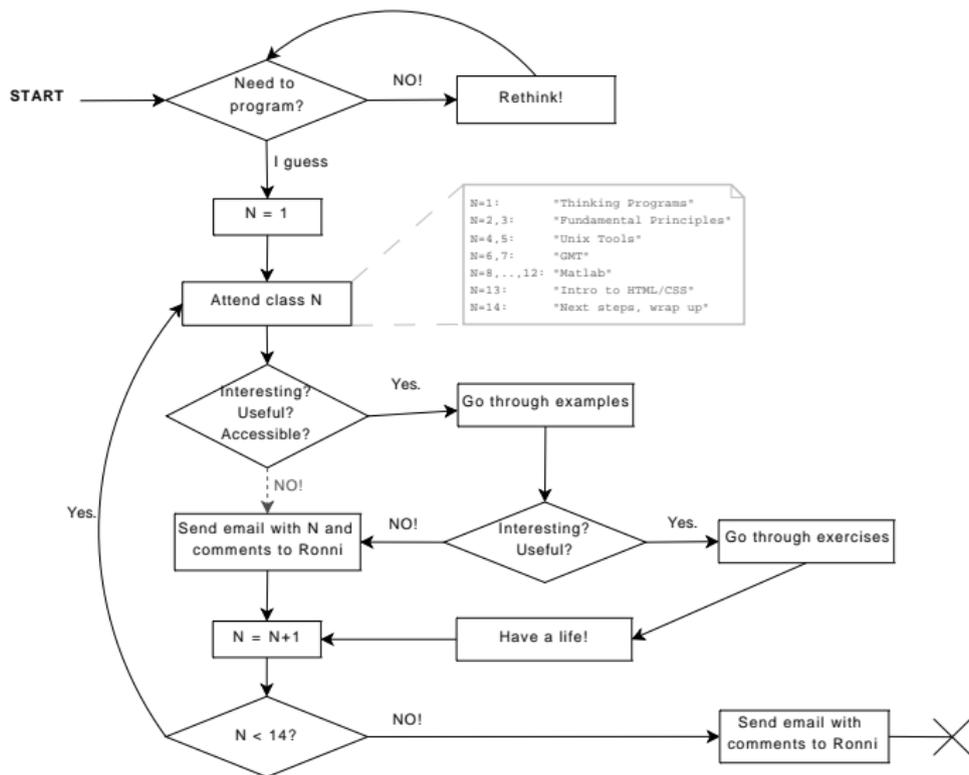
The Program ...



The Program ...

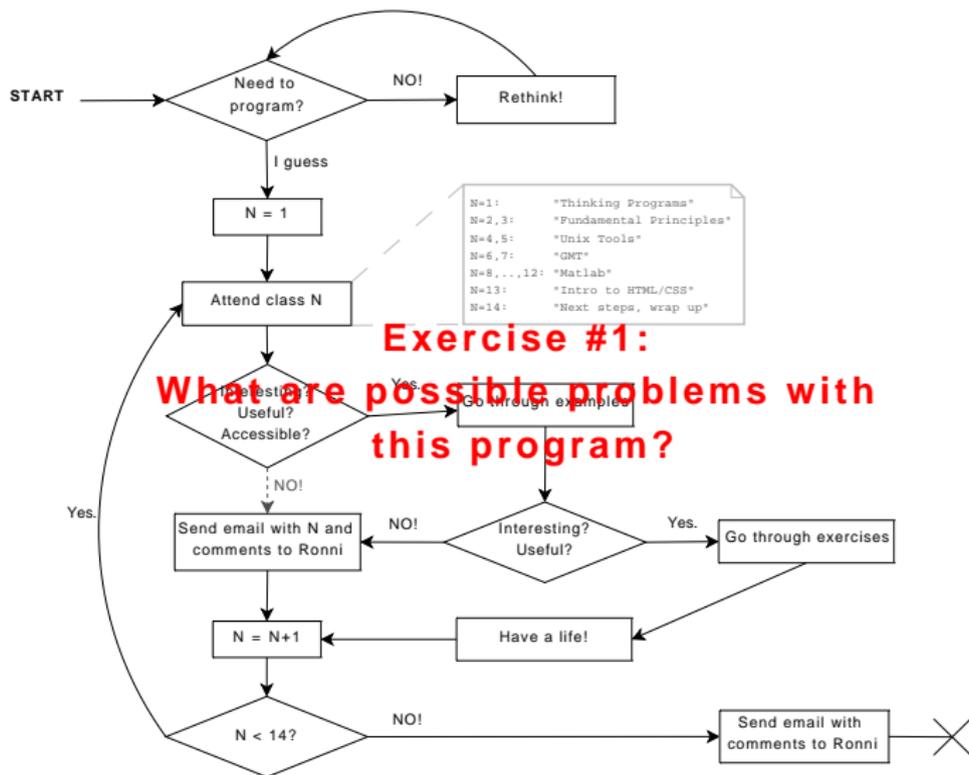


The Program ...



Listing 1: Class flow

The Program ...



Listing 1: Class flow

Modules ... (tentative order)

- `brainWashing()`, `learnMantras()` – **Thinking Programs, Fundamentals of Programming**, me
The esoteric / abstract side of programming.

Modules ... (tentative order)

- `brainWashing()`, `learnMantras()` – **Thinking Programs, Fundamentals of Programming**, me
The esoteric / abstract side of programming.
- `discoverWorld()` – **Unix Tools**, Jeff Freymueller, me
Unix can make your life a lot easier. Intro to tools and scripting.

Modules ... (tentative order)

- `brainWashing()`, `learnMantras()` – **Thinking Programs, Fundamentals of Programming**, me
The esoteric / abstract side of programming.
- `discoverWorld()` – **Unix Tools**, Jeff Freymueller, me
Unix can make your life a lot easier. Intro to tools and scripting.
- `drawMap()` – **Generic Mapping Tools**, Bernie Coakley
GMT can map your data.

Modules ... (tentative order)

- `brainWashing()`, `learnMantras()` – **Thinking Programs, Fundamentals of Programming**, me
The esoteric / abstract side of programming.
- `discoverWorld()` – **Unix Tools**, Jeff Freymueller, me
Unix can make your life a lot easier. Intro to tools and scripting.
- `drawMap()` – **Generic Mapping Tools**, Bernie Coakley
GMT can map your data.
- `newSpeak()`, `messWithData()` – **General Matlab**, Jeff Freymueller, me, TBD
How to do actual programming using Matlab. Things you can do to your data.

Modules ... (tentative order)

- `brainWashing()`, `learnMantras()` – **Thinking Programs, Fundamentals of Programming**, me
The esoteric / abstract side of programming.
- `discoverWorld()` – **Unix Tools**, Jeff Freymueller, me
Unix can make your life a lot easier. Intro to tools and scripting.
- `drawMap()` – **Generic Mapping Tools**, Bernie Coakley
GMT can map your data.
- `newSpeak()`, `messWithData()` – **General Matlab**, Jeff Freymueller, me, TBD
How to do actual programming using Matlab. Things you can do to your data.
- `webPresence()`, `stuffPlusPlus()` – **HTML/CSS, OO Overview**, me
How does that homepage stuff work? Is that all I need to know?

`http://www.gps.alaska.edu/programming`

Things to find:

- slides: the actual presentation
- handouts: write up of lectures that should culminate in a manual type thing
- examples: working samples from each lecture.
- exercises: invitation for you.

Print slides before lecture so you can take necessary notes :)

This course is for you . . . give feedback! (any time, not just on the yellow sheets at the end)

The very basics (1)

From 'The Conscience of a Hacker', The Mentor (1986):

[...] I made a discovery today. I found a computer. Wait a second, this is cool. It does what I want it to. If it makes a mistake, it's because I screwed it up. Not because it doesn't like me ...

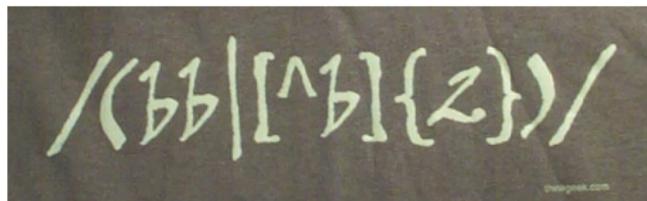
Or feels threatened by me ...

Or thinks I'm a smart ass ...

Or doesn't like teaching and shouldn't be here [...]

The very basics (2)

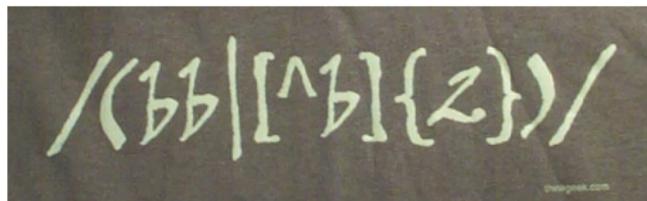
- Programming is beyond language.



<http://thinkgeek.com>

The very basics (2)

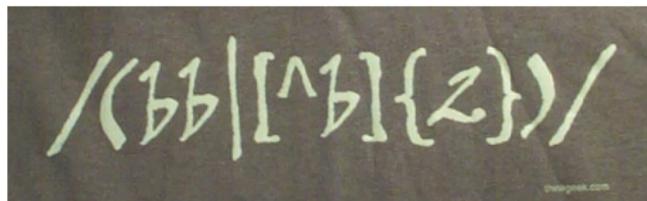
- Programming is beyond language.
- Programming is about writing code that people can read.



<http://thinkgeek.com>

The very basics (2)

- Programming is beyond language.
- Programming is about writing code that people can read.
- Code is poetry.



<http://thinkgeek.com>

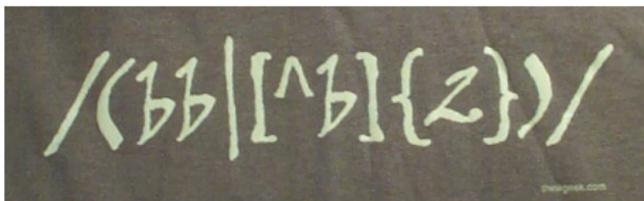
*“When I’m writing poetry,
it feels like the center of
my thinking is in a
particular place, and
when I’m writing code the
center of my thinking feels
in the same kind of place.”*

Richard Gabriel,

Distinguished Engineer at Sun Microsystems

The very basics (2)

- Programming is beyond language.
- Programming is about writing code that people can read.
- Code is poetry.
- RTFM *and/or* the internet



<http://thinkgeek.com>

“When I’m writing poetry, it feels like the center of my thinking is in a particular place, and when I’m writing code the center of my thinking feels in the same kind of place.”

Richard Gabriel,

Distinguished Engineer at Sun Microsystems

Jon Claerbout (a geophysicist), as quoted in “WaveLab and Reproducible Research”:

Jon Claerbout (a geophysicist), as quoted in “WaveLab and Reproducible Research”:

*An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

More Philosophy . . .

Jon Claerbout (a geophysicist), as quoted in “WaveLab and Reproducible Research”:

*An article about computational science in a scientific publication is **not** the scholarship itself, it is merely **advertising** of the scholarship. The actual scholarship is the complete software development environment and the complete set of instructions which generated the figures.*

Implications . . .

- publications should include data and code (example: Okada)
- figures should be reproducible by readers
- write code that others can use!

What does that mean?

Good

```
1 function fp = screw2d(x, xf, d, sdot)
% function fp = screw2d(x, xf, d, sdot)
3 %
% Computes fault-parallel slip rate for 2D screw dislocation
5 % with fault located at xf, with locking depth d and slip rate sdot.
% Will compute at one or many locations x.
7 %
% x    column vector
9 % xf  scalar
% d    scalar
11 % sdot scalar
%
13 if ( d == 0 )
    fp = sdot*0.5*sign(x-xf*ones(size(x)));
15 else
    fp = sdot*atan2((x-xf*ones(size(x))),d)/pi;
17 end
```

What does that mean?

Bad

```
function fp = screw2d(x, xf, d, sdot)
2 if (d==0)fp=sdot*0.5*sign(x-xf*ones(size(x)));else fp=sdot*atan2((x-xf*ones(size(x))),d)/pi;
end
```

Outline

- 1 Overview and Philosophies
- 2 Thinking programs**
- 3 Building programs
- 4 Summary

Example 1:

Getting into Grad School

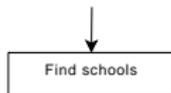
Example 1:

Getting into Grad School

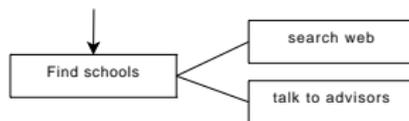
things to do:

apply, figure out where to go, visa stuff, class work, research, thesis . . .

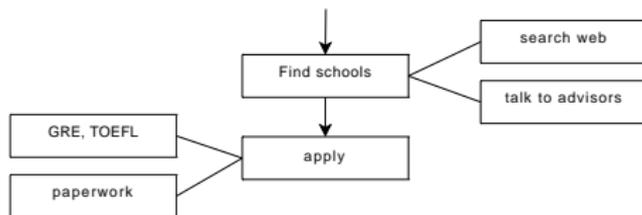
Thinking programs – Breaking down complex tasks



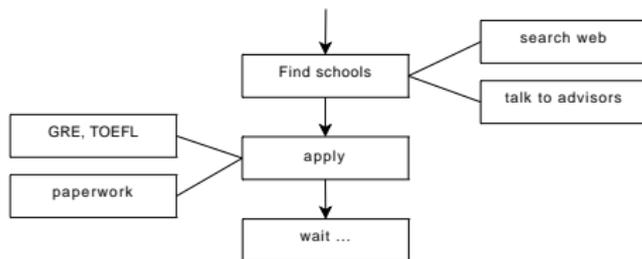
Thinking programs – Breaking down complex tasks



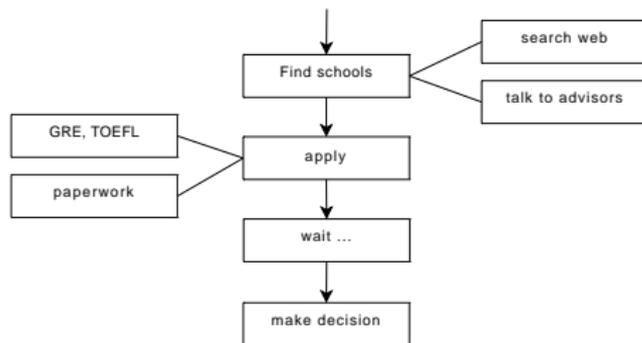
Thinking programs – Breaking down complex tasks



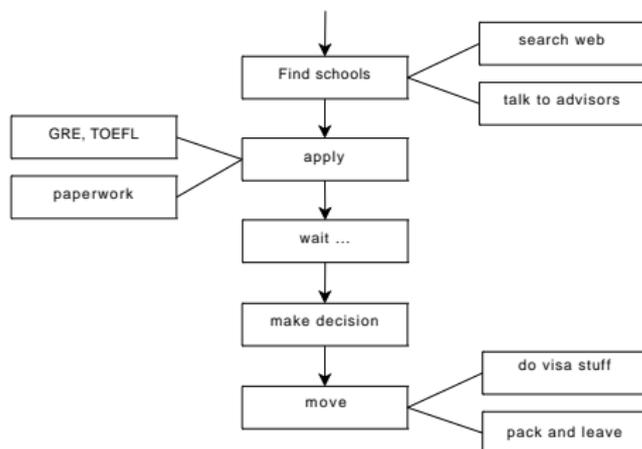
Thinking programs – Breaking down complex tasks



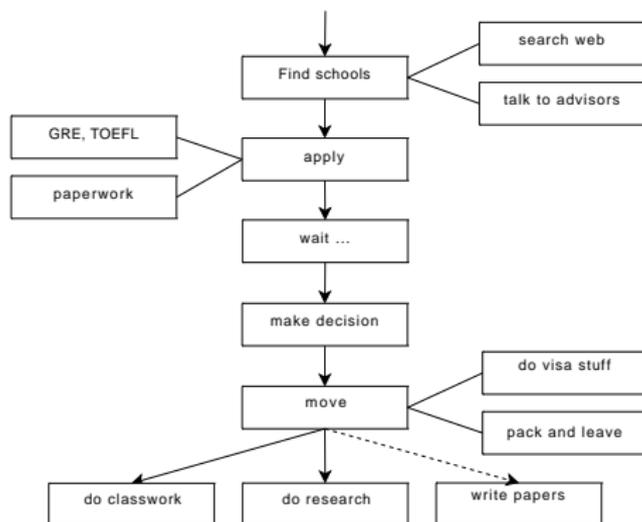
Thinking programs – Breaking down complex tasks



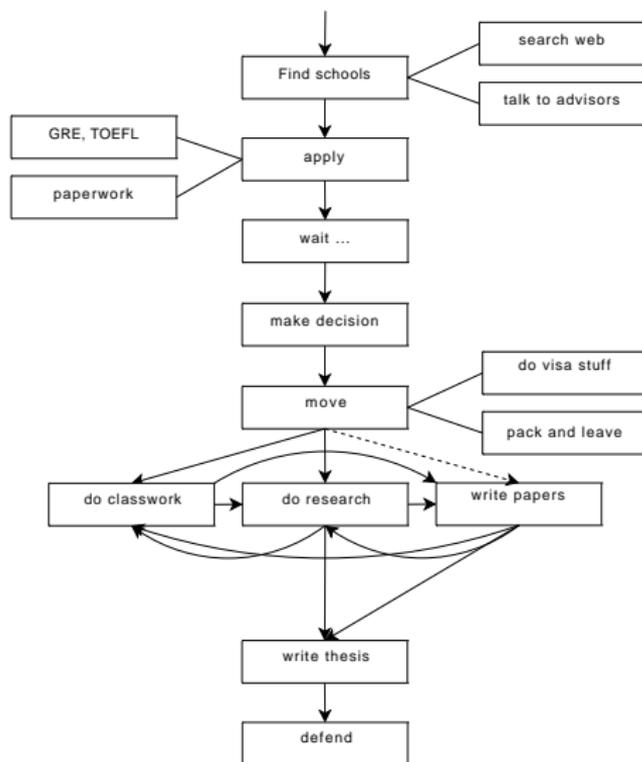
Thinking programs – Breaking down complex tasks



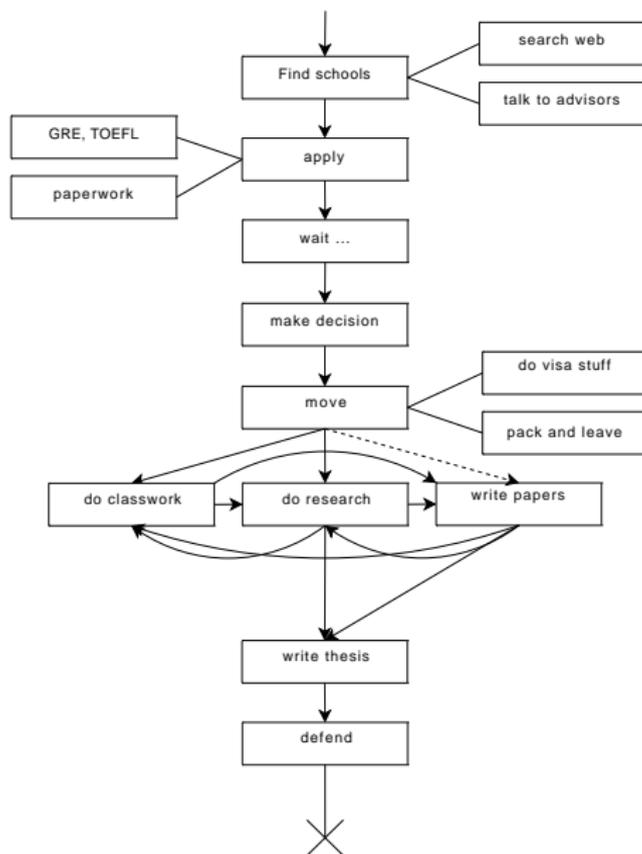
Thinking programs – Breaking down complex tasks



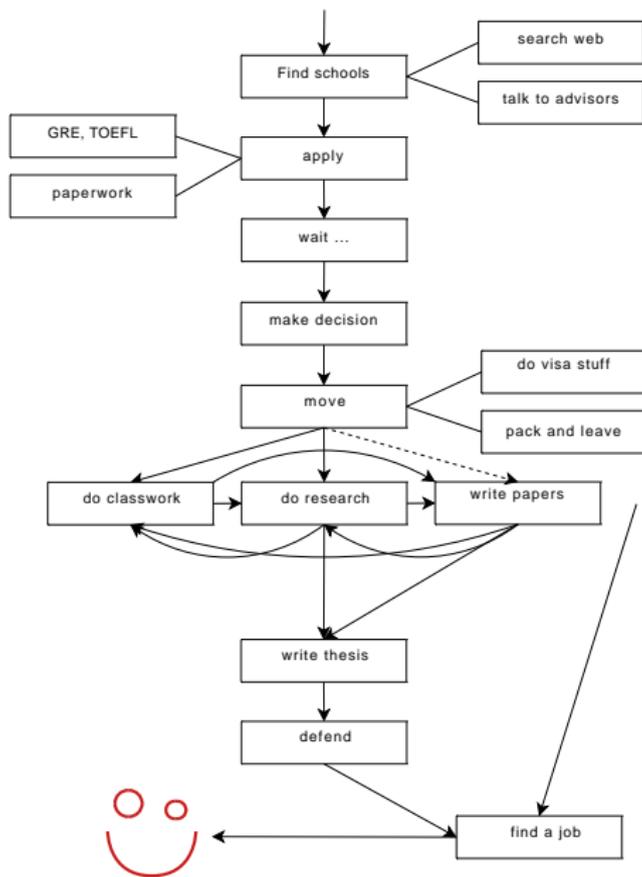
Thinking programs – Breaking down complex tasks



Thinking programs – Breaking down complex tasks



Thinking programs – Breaking down complex tasks



Example 2:

Grad student's Average Day

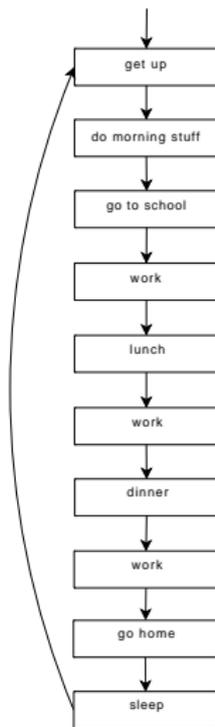
Example 2:

Grad student's Average Day

possible activities:

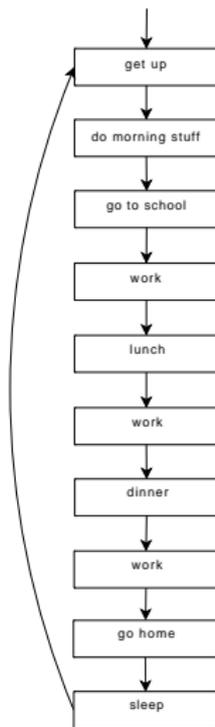
eat, sleep, work, do stuff, . . .

Thinking programs – Breaking down complex tasks



Listing 1: make_my_day

Thinking programs – Breaking down complex tasks



possible implementation

```
% make_my_day.m
2 %-----
% program that shows how much fun
4 % live as a grad student is :)

6 getUp;
  eat('breakfast');
8 walk('school');
  work;
10 eat('lunch');
  work();
12 eat('dinner');
  work();
14 walk('home');
  haveLife;
16 sleep;
```

Listing 1: make_my_day

Outline

- 1 Overview and Philosophies
- 2 Thinking programs
- 3 Building programs**
- 4 Summary

Building programs – One black box at a time

Strategies to implement a program:

Top down

Same as the examples above:

- start with the big picture
- identify reasonable subtasks
- try to divide things to a level of manageable complexity (atoms)
- implement atoms
- implement main routine (flow control)

Building programs – One black box at a time

Strategies to implement a program:

Top down

Same as the examples above:

- start with the big picture
- identify reasonable subtasks
- try to divide things to a level of manageable complexity (atoms)
- implement atoms
- implement main routine (flow control)

Bottom up

- problems accumulate
- implement an atom at the time
- at some point you figure out that things could go together
- revise main routine constantly
- add necessary subroutines

Bottom line

- Try building tools that solve a set of similar problems in a generic way. Use Parameters!
- Build and test each atom individually, test all scenarios (and more) with synthetic input.
- Treat atoms as black boxes that implement desired functionality. Don't care about them once they're working

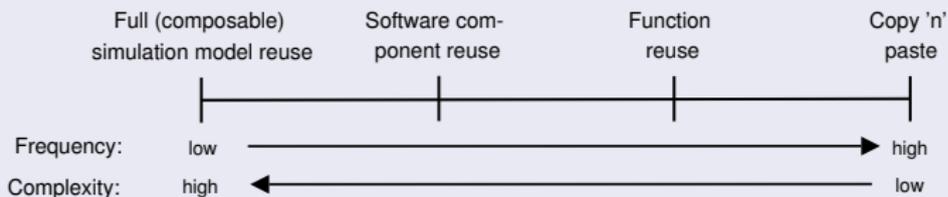
Keys to good programs

- **Modularity:** split problem in manageable tasks, implement and test one at a time

Building programs – One black box at a time

Keys to good programs

- **Modularity:** split problem in manageable tasks, implement and test one at a time
- **Reusability:** write functions, avoid redundance, avoid monolithic code (theoretically one loop would be enough)

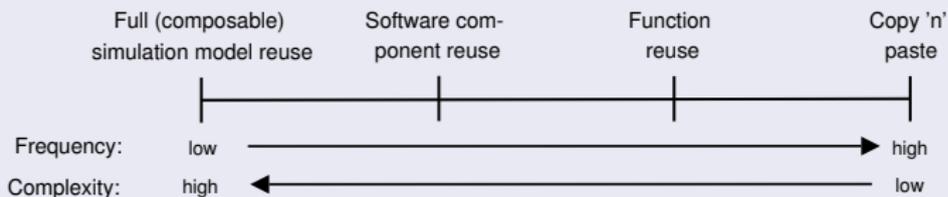


Pidd, 2002

Building programs – One black box at a time

Keys to good programs

- **Modularity:** split problem in manageable tasks, implement and test one at a time
- **Reusability:** write functions, avoid redundance, avoid monolithic code (theoretically one loop would be enough)



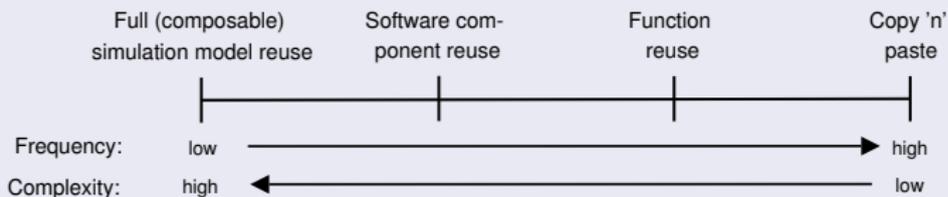
Pidd, 2002

- **Generalize:** use variables instead of hard coded values, hand parameters to functions

Building programs – One black box at a time

Keys to good programs

- **Modularity:** split problem in manageable tasks, implement and test one at a time
- **Reusability:** write functions, avoid redundance, avoid monolithic code (theoretically one loop would be enough)



Pidd, 2002

- **Generalize:** use variables instead of hard coded values, hand parameters to functions
- Functionality, then efficiency

The Control Routine

```
% make_my_day.m
2 %-----
% program that shows how much fun
4 % live as a grad student is :)

6 getUp;
  eat('breakfast');
8 walk('school');
  work;
10 eat('lunch');
  work();
12 eat('dinner');
  work();
14 walk('home');
  haveLife;
16 sleep;
```

Using Parameters

```
% eat.m
2 %-----
function eat(what)
4     disp(sprintf('yummy_..._%s', what));
     pause(2);
6 end
```

Outline

- 1 Overview and Philosophies
- 2 Thinking programs
- 3 Building programs
- 4 Summary**

Summary – Take home messages

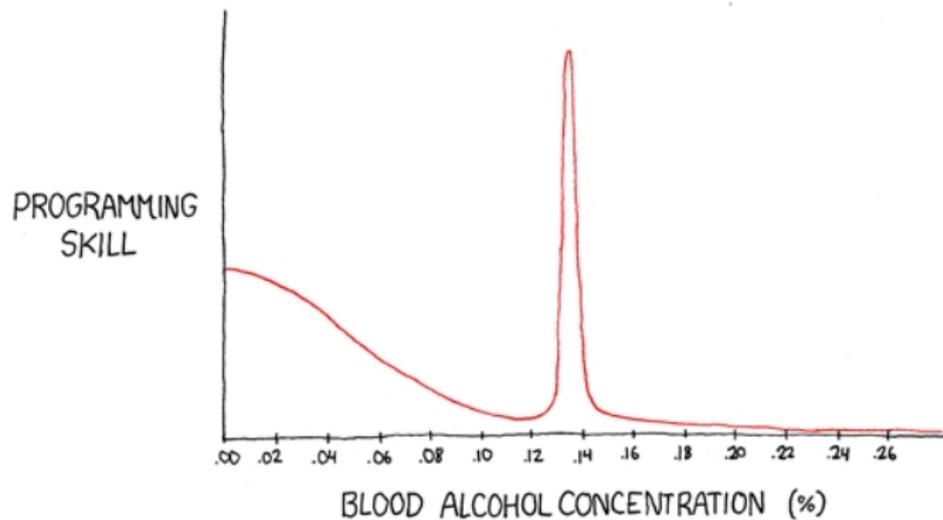
Thinking ...

- Think modular
- Think in general cases
- Think non-redundant
- Think about reuse
- Think about reproducibility

Exercising ...

- Read other peoples' code ... critically
- The first version is for the trash bin (unintentionally)

If all fails . . .



"The Ballmer Peak"

<http://www.xkcd.com/323/>