SWE 265P
Reverse Engineering and Modeling

Lecture 2: Basic strategies

Reality

"Reading code, like writing it, is an iterative process. Don't expect to understand the whole system all at once. If you understand one function or object, you can build on that to gain an understanding of the code that uses it or is used by it."
– Sara Triplett [Lead development actuary, FIS]

Today

- Last week's material
- Basic strategies
- In-class practice

Last week's material

- Many reasons exist for developers to read code
- Despite there being a variety of high-level approaches and useful tools that can assist, no recipe exists and each person has to find their own best way of reading code
- Reflecting on one's activities is key to finding this best way
- Opportunity to build an important resume item in terms of real-world experience, a GitHub profile, stories to tell during an interview, etc.

Last week's homework

- Download, build, and run some large systems
- Teams & diaries

Basic strategies

- Top-down comprehension know what software supposed to do
  - reconstructing knowledge of the domain of the program and mapping this knowledge to the source code

small part of code

- Bottom-up comprehension
  - reading code statements and mentally chunking or grouping these statements into higher level abstractions, and aggregate these abstractions further until a high-level understanding of the program is attained

Basic strategies

- Systematic comprehension relation between different parts
  - reading the code in detail, tracing through the control-flow and data-flow abstractions in the program to gain a global understanding of it

- Opportunistic comprehension
  - as-needed focusing only on the code relating to the task at hand

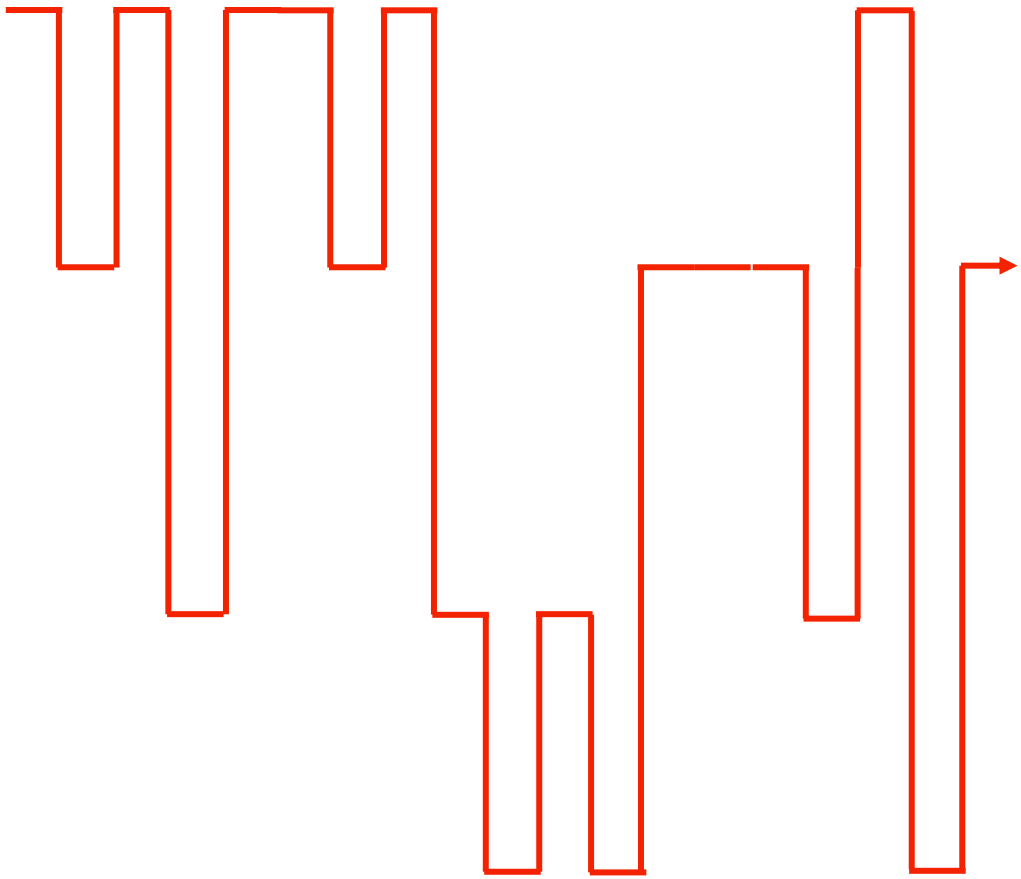
Reality

top-down

bottom-up

systematic

opportunistic



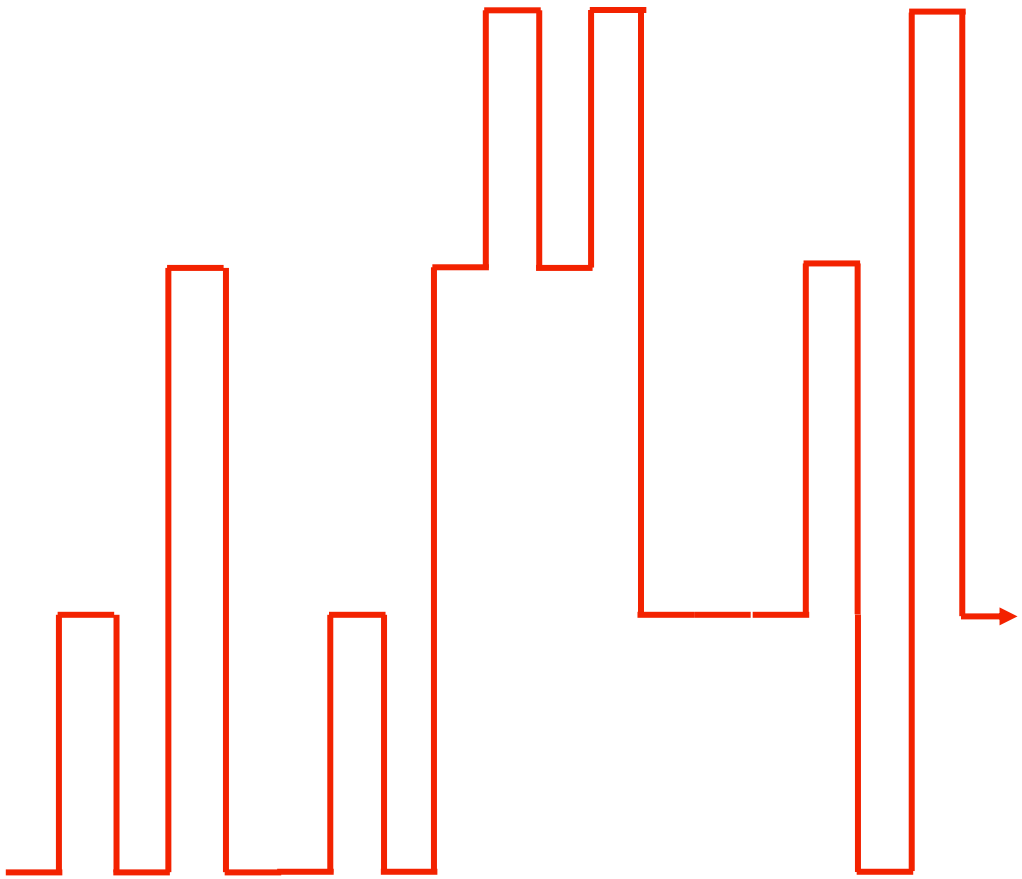
Reality

top-down

bottom-up

systematic

opportunistic



Information foraging theory

- The theory of optimal foraging stipulates that foraging animals attempt to maximize their energy intake (by finding food) over the time required to find that food

- The theory of information foraging stipulates that information seeking humans attempt to maximize the value of processing information and minimize the cost of 'traveling' to find that information

Impact of familiarity

Familiar	Non-familiar
Hypothesis-driven	Inference-driven
Beacons	Chunks
Programming plans	Abstractions
Iteration	Iteration

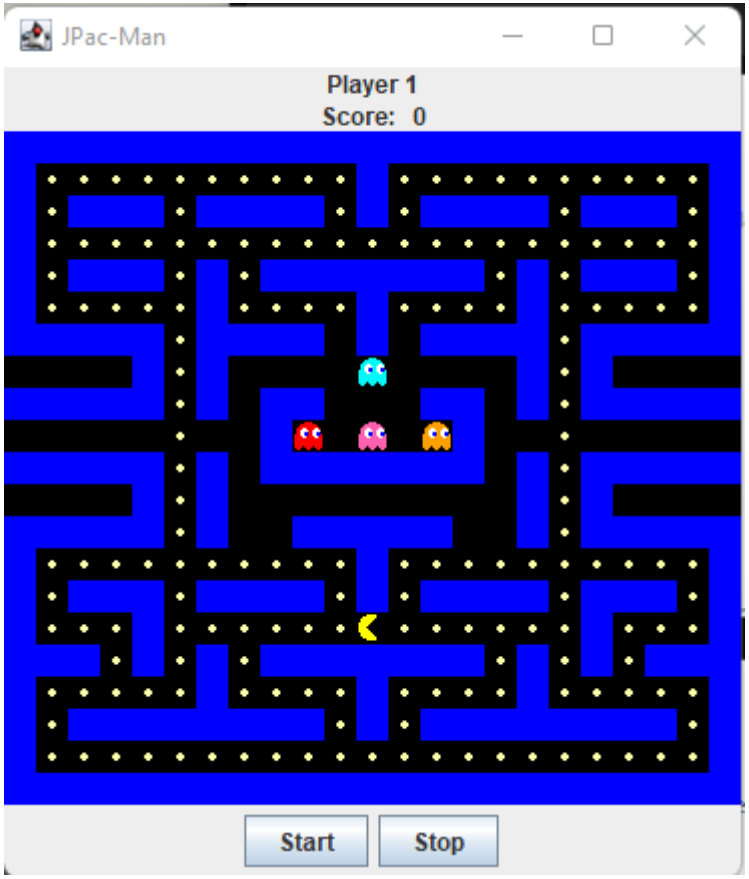
Reality

- Most often, programmers use both
  - use domain knowledge to form hypotheses
  - react to counter evidence with more detailed reading
- Professionals focus on getting a task done, rather than on understanding the code

Goal-driven

- Ultimately, how developers approach the problem of reverse engineering a software system depends on the goal that they have
  - learning
  - fixing a bug
  - reviewing a pull request
  - adding/changing some functionality
  - assessing a component before adopting it
  - vulnerability analysis
  - rearchitecting
  - …

JPacMan



https://www.google.com/logos/2010/pacman10-i.html

Let’s practice: JPacMan1

- Use IntelliJ to clone https://github.com/SWE-265P/jpacman1
- Open the project

JPacMan goal #1 (fix a bug)

- Fix the direction that PacMan moves, because right now it moves in the direction opposite of the key that the player presses

JPacMan goal #2 (change to the code)

- Change the amount earned per dot to 25 points

Break

Let’s practice: JPacMan2

- Use IntelliJ to clone https://github.com/SWE-265P/jpacman2
- Open the project

JPacMan goal #3 (learn)

- How does JPacMan animate the PacMan character?

JPacMan goal #4 (learn)

- Are there fruits in this particular implementation of PacMan?

Let’s practice: JPacMan3

- Use IntelliJ to clone https://github.com/SWE-265P/jpacman3
- Open the project

JPacMan goal #5 (code review)

- By what rules does the cyan ghost move?

JPacMan goal #6 (change functionality)

- Each keystroke makes PacMan move 2 spots

Homework (team)

- With your team, decide upon a large open source system that you want to use as the basis for your course project
- Each open source system can only be claimed by a single team
- Must claim your system by Friday, April 12, 9am
  - e-mail to Yirui, Yu, and Andre
  - name of system, motivation for choice, size in LOC and how you measured it, and what system each team member studied in Prof. Jones' class and in Prof. Malek's class
- System must be approved before you can work on it

Homework (team, continued)

- Your large open source system should:
  - consist of at least 100,000 lines of code
  - employ an issue tracking system and have at least 100 open issues
  - have at least 10 new pull requests over the past month, at least some of which have been accepted
  - be written in Java (though other languages are acceptable; contact Yirui, Yu, and Andre)
  - *not* be a stand-alone library, but a system with actual end-user functionality (exceptions may potentially be granted)
  - be different from the system any of the students on your team studied in Prof. Jones' class and in Prof. Malek's class
  - be of interest to you and your team
  - serve as a resume builder

Homework (individual)

- How to read code without ripping your hair out
  - https://medium.com/launch-school/how-to-read-source-code-without-ripping-your-hair-out-e066472bbe8d
- 7 Ways to improve your code reading skills
  - https://dzone.com/articles/7-ways-to-improve-your-code-reading-skill
- How to read source code
  - https://github.com/aredridel/how-to-read-code/blob/master/how-to-read-code.md

Homework (individual)

- Make sure to regularly update your personal diary

Homework (optional)

- Continue to explore JPacMan3 by answering the following questions:
  - what is the role of EmptySprite?
  - what is the role of MOVE_INTERVAL and INTERVAL_VARIATION?
  - if you wanted to add a fruit, which files would you need to change?