Refactoring and Programming

- Make it run
  - It’s hard to optimize, debug, tune, enhance a design
  - Possible?
- Make it right
  - Make sure the system works before you add all/any enhancements
- Make it fast
  - Fast enough is enough
- Make it small
  - Why might this be important?

Use cases

- An actor interacts with system
  - Might be a person using the system
  - Might be a program or process interacting with the system
- Use cases are descriptions of what happens when an actor uses the system to achieve a goal
  - Collection of possible sequences of interactions between the system and its actors relating to a goal
  - Use cases should define all system behavior relevant to actors achieving goals and should not involve other factors
- Easy to read, scenario/descriptive
- NOT: UI/GUI, implementation based

SLOGO use case

- There is a read-eval-print loop
  - How else might user interact with environment
  - What information should be accessible to user
  - What about display? Part of use case?
- Debugging and error messages, what are issues?
  - AST -> maps to source, what is this useful for?
  - Flag more than the first error? Error correction?
    - Throw exceptions?
- What about design/development/testing
  - How can graphical display be tested independently of parser, lexer?

Parsing

- What’s needed to add a new instruction
  - Where do we add code, what different classes are needed?
  - What improvements are possible?
- Can we think about this without the design/code?
  - Can we recognize flaws before we have them—nascent problems and fix them at design time rather than code/design loop?
- What about reflection?