S05 Software Design

Software Complexity
Software Design

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

Software Complexity

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+
[<]<-]>>.>---.+++++++..+++.>>.<-.<.++
+.------.--------.>>+.>++.
```

- “Hello World” in the BrainF#@k language (apparently: source wikipedia)
- Syntax only 8 characters, *Turing complete*
- Simple or complex?

Software Complexity

- The International **Obfuscated** C Code Contest
- Yusuke Endoh one of the 2020 winners: Minesweeper Solver

What is Software Complexity?

- ***Accidental* Complexity**
  - Software that is designed or presented in a way that is more difficult for a **human to understand, use and modify** *than it needs to be*.
  - It is difficult to write elegant, clear, reusable code.
- ***Essential* Complexity**
  - Inherent to the problem being solved. Irreducible.
- **Not to be confused with** computational complexity.

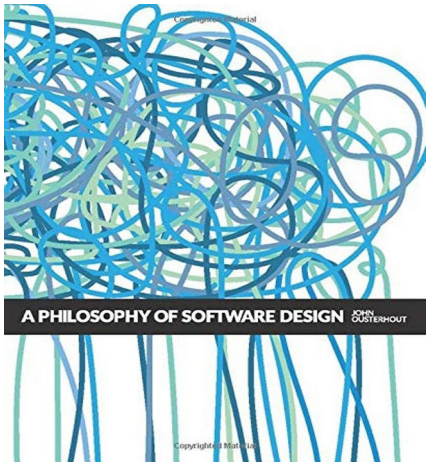
Software Complexity

- Some **contributing factors**:
  - Interlinking many components
  - Unstated assumptions
  - Non-local changes, unintuitive side-effects
  - Duplication / lack of encapsulation / exposure to details
  - Poor naming
  - Not following conventions / inconsistency
- Often **incrementally** works its way into a project, e.g., *feature creep*, dealing with *legacy*.

Good Software Design

- Many opinions. Conventions / preferences vary between communities.
- Recommendation:

  A Philosophy of Software Design, John Ousterhout



- Design principles
- Red flags

Some Principles (Ousterhout)

- **Deep “modules”** (method, class, package, or module)
  - Simple interfaces* (narrow)
  - Encapsulate lots of complexity (depth)
  - General-purpose
- Prefer **simple interface** over simple implementation
- Design **errors out of existence**
- Design for **ease of reading**, not ease of writing
- Extra: Don’t Repeat Yourself (**DRY**) and **SOLID** principles

* Interfaces in the broad sense, not just the Java keyword

Some Red Flags (Ousterhout)

- **Shallow module**: interface not much simpler than implementation
- **Overexposure**: user needs to be aware of rarely-used features
- **Repetition**: non-trivial code is repeated
- **Conjoined methods**: methods are so co-dependent that you have to understand implementation of both
- **Comment repeats code**
- **Hard to name entity**
- Extra: **Deeply nested control-flow blocks**