

Handling Complexity

Mendel Rosenblum
CS110

Handling complexity

Note: You are going to fail in the end

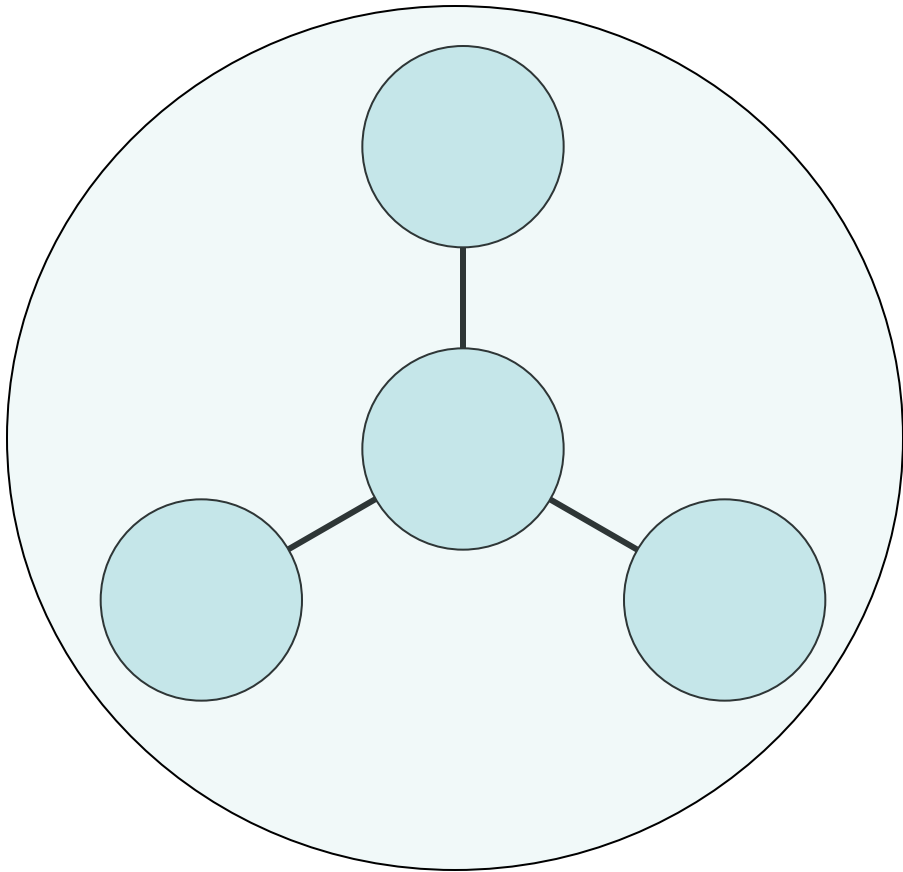
But you will get much further with these techniques

1. Modularity
2. Abstraction
3. Layers
4. Hierarchy
5. Names
6. Iteration
7. Simplicity

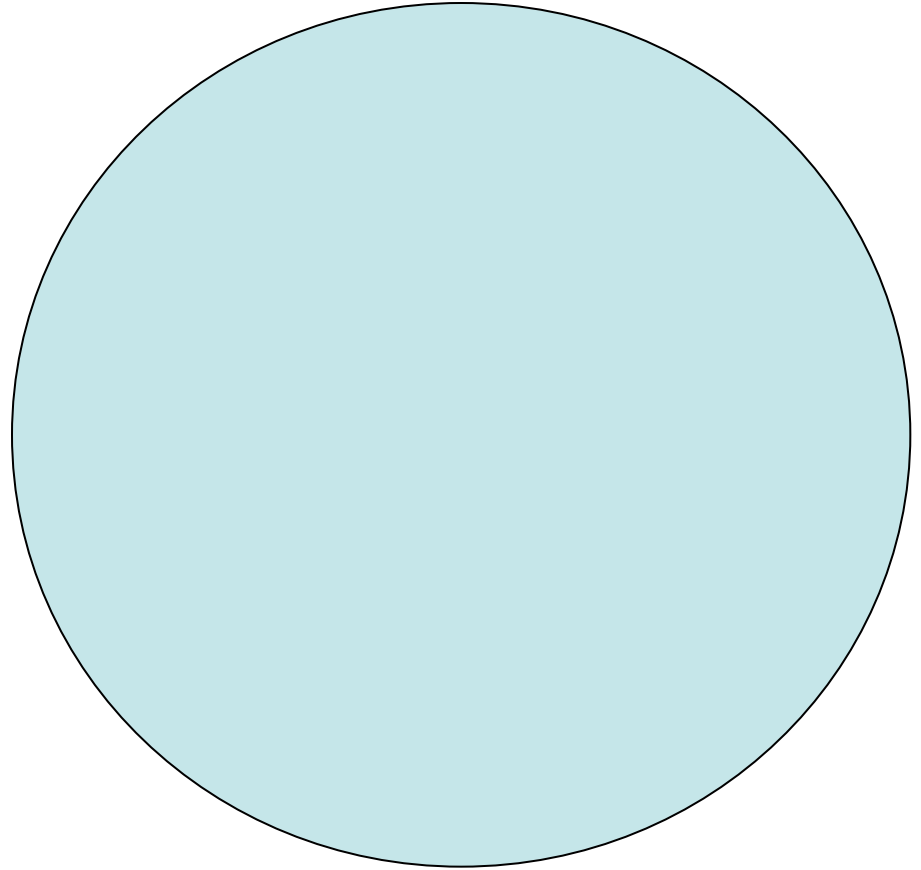
Modularity

- Divide-and-conquer
- Build a large system out a many smaller, simpler ones combined together
 - Make it possible to reason about a much bigger system
 - If done right, individual modules can be modified without effecting the rest of the system

Modularity



With Modularity



Without Modularity

Principle: The unyielding foundations rule

It is easier to change a module than to change the modularity

- Modules are the replacement unit
- Changing the interface to a module requires changing all the modules that talk to it
- Initial modularity choice is key
- For software, if you don't like the current modularity, plan for a total rewrite of the system
 - Makes the current system not seem so bad after all

Abstraction

- Divisions into modules so there is little or no propagation effects between modules
- Natural division points have:
 - Few interactions among modules
 - Modules treat others based on their interface
 - Implementation hiding
- Example:
 - `printf(“Hello World\n”);`
 - An operating System

The robustness principle

Be tolerant of inputs and strict on outputs

- Helps with propagation effects
- The basic principle behind mass production

The safety margin principle

Keep track of the distance to the cliff, or you may fall over the edge

- Note when things go wrong even if you can process them sensibly
- Earlier the warning the better
 - Make root cause analysis easier
- Example:
 - Beta builds in software

Layers

- Minimize the interactions between modules by stacking them in layers
 - Modules only interact between the one above and below
 - Computer systems use this heavily
 - Application
 - Operating system
 - Hardware
 - And internally as well

Hierarchy

- Organize modules in a tree-like structure
 - Reduces interconnections and hence complexity
- Module designer need only worry about interactions with children
- Also widely used

Names

- Name connections into a module
 - Other modules refer by name
 - Allows easy replacement of the module
- Delay binding of modules together
- Examples:
 - All over the place

Principle: Decouple modules with indirection

Indirection supports replaceability

- Widely used:
 - “any problem in a computer system can be solved by adding a layer of indirection”
- Examples:
 - Virtualization

Hard to get modularity right

- Many choices for modularity, abstraction, and different layering and hierarchies
 - It's hard to get it right the first time
- Even if you're close, technology changes can do you in
 - Remember incommensurate scaling

Principle: The incommensurate scaling rule

Changing any system parameter by a factor of ten usually requires a new design

- Design while guessing at future technology is hard
 - Some changes hard to predict
- Bad things happen if you get wrong;
 - OS examples:
 - OS/2
 - Windows Vista

Principle: Design for iteration

You won't get it right the first time, so design it to be easy to change

- Design the system to be redesigned

Principle: Keep digging

Complex systems fail for complex reasons

Simplicity

- KISS – Keep it simple, stupid.
- Be ready to say no to a feature
 - This will make it too complicated
- Hard to push against the “we really need this new feature” trend

Principle: Adopt sweeping simplifications

So you can see what you are doing

- Choosing the right abstraction can be extremely powerful
- The rest of the course is about this

Sample Exam Question

The rapid rate of improving in computer hardware techniques is sometimes blamed for the increased complexity of computer systems but it also can result in a less complexity system doing the same functionality. Explain how this is the case.

For next class

- Read Section 2.1 & 2.2