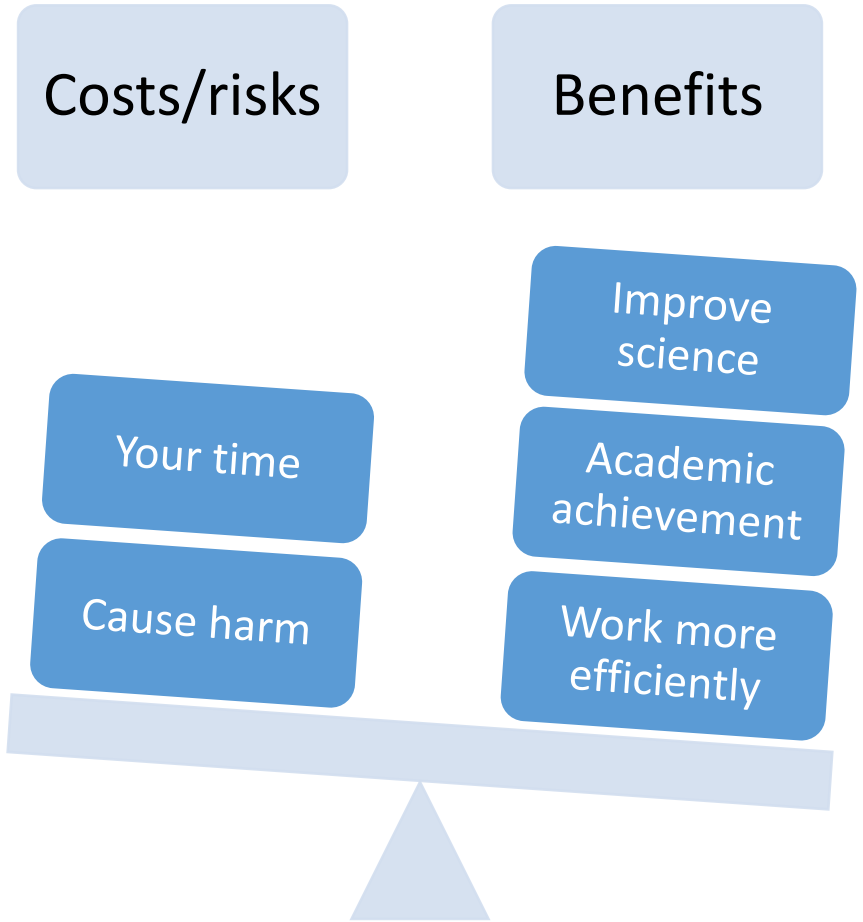
Approaches to software development

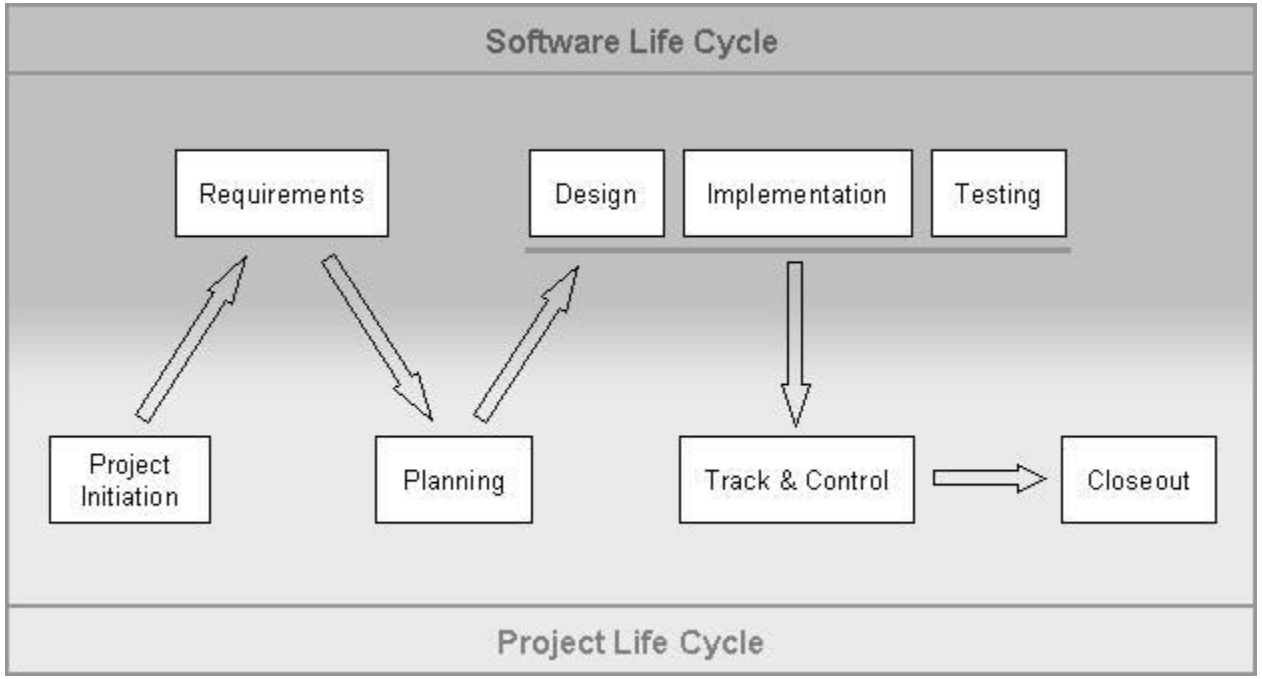
Balance is key



Characteristics of good software

- Operational
  - Correct
  - Usable
  - Efficient
- Transitional
  - Portable
  - Interoperable
  - Adaptable
- Maintainable
  - Modular
  - Flexible

The basic lifecycle



Follow a process

- A process specifies when and how to approach the elements of the basic lifecycle
  - Order of activities
  - Entry and exit criteria
- There isn’t one best process
- It is critical when working as part of a team

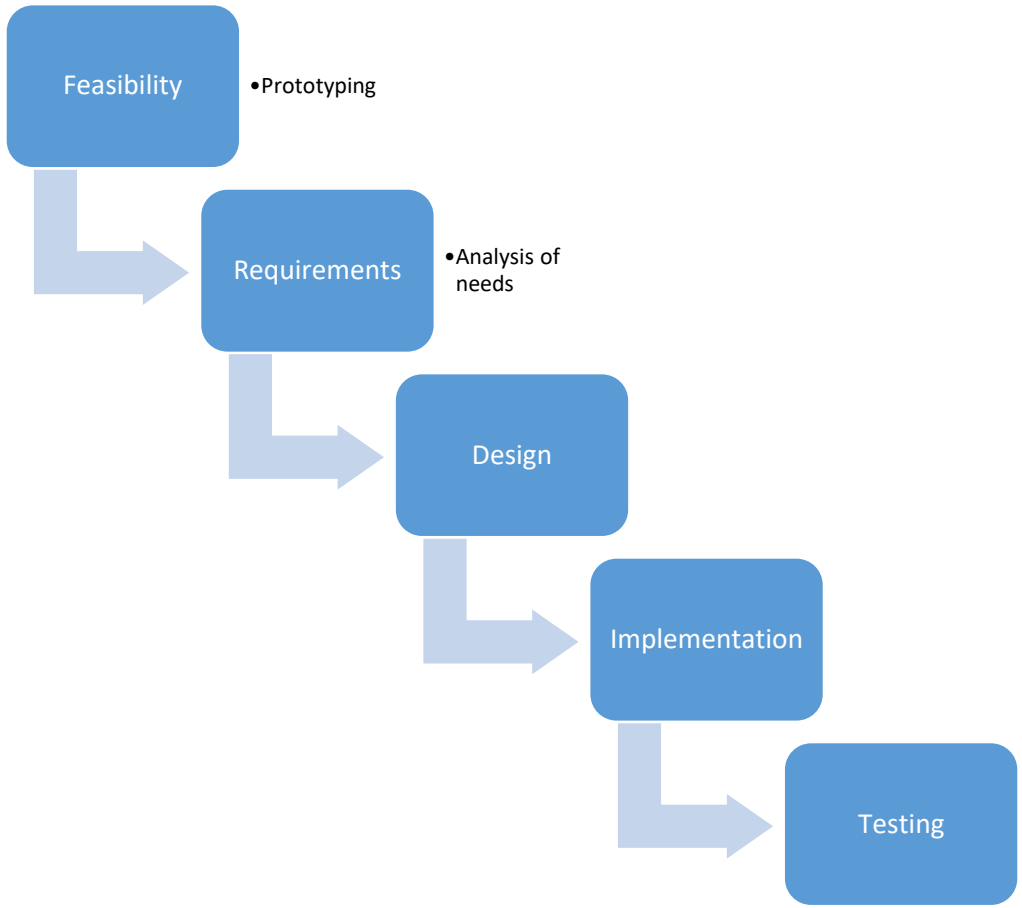
Processes **not** to follow

Code and fix/rushing to code

- As soon as the problem is identified, begin coding
- Once something is working, it is reworked until it meets all requirements

- Probably the most used approach
- Intuitive, akin to dog-paddling, instead of swimming
- **Key problem:** refactoring is difficult, time consuming
  - Risk of incorrectness
  - Reduces interoperability & modularity

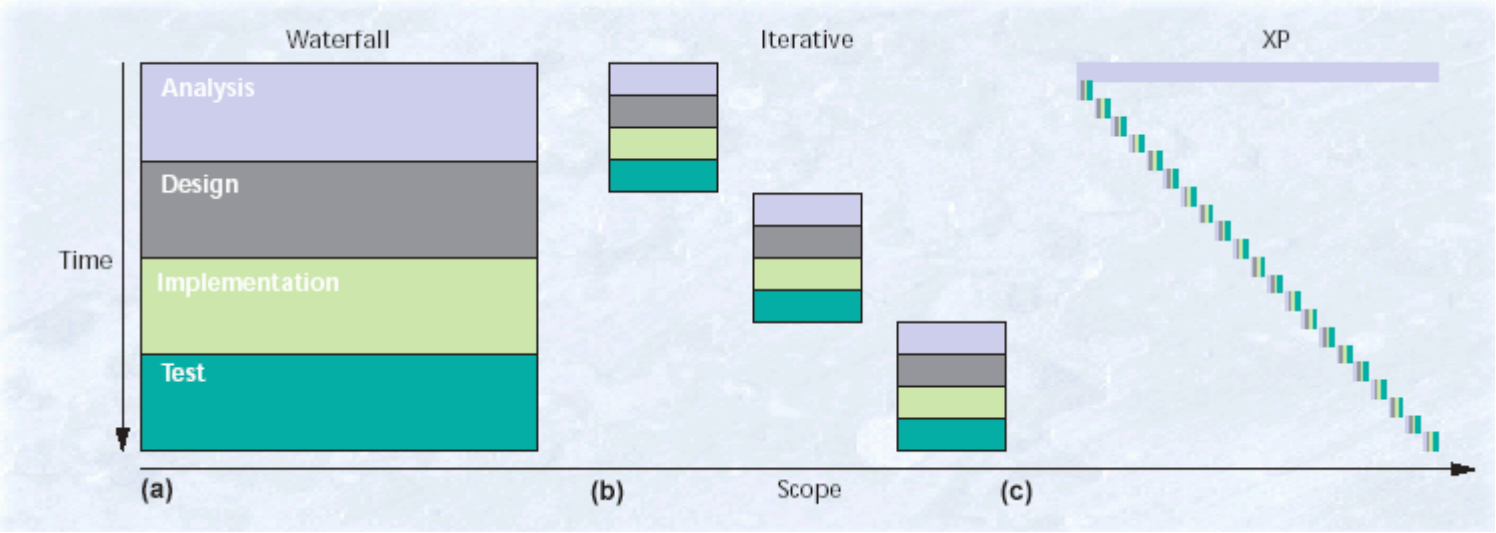
Classic waterfall



Drawbacks

- Code and fix lacks discipline
  - Inefficient
  - Sets you up for problems with maintenance and interoperability
  - May be OK for small projects
- Classic waterfall is too rigid
  - Does not manage risks well
  - Difficult to adapt to changing requirements/new knowledge

Agile methods



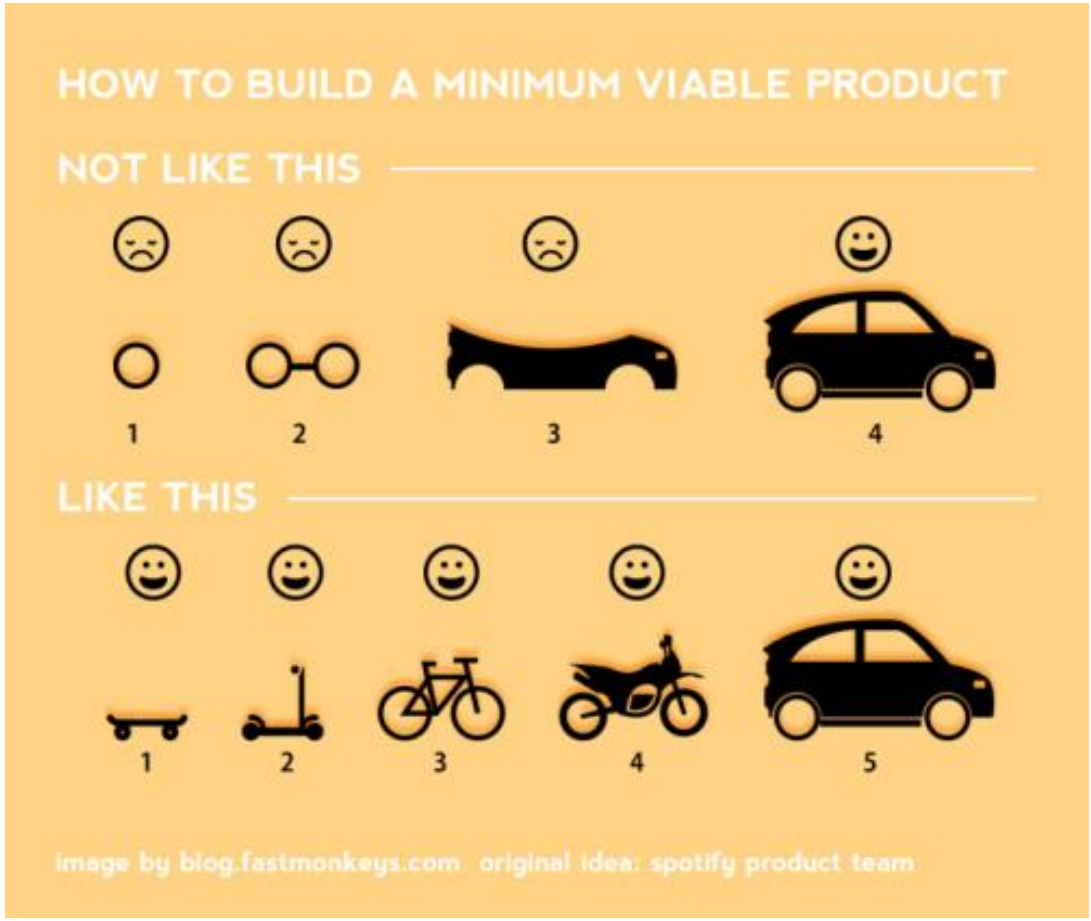
- Iteration is good for managing risks
- Requirements develop and change over time
- Don’t stick to a plan at all costs
- Stick to a plan as long as it is limited in scope

Minimum viable package

What is the goal?
Create a package that does something useful

- Don’t sacrifice functionality for generality
- Make something that works
- Iteratively add features, make more general (if needed)

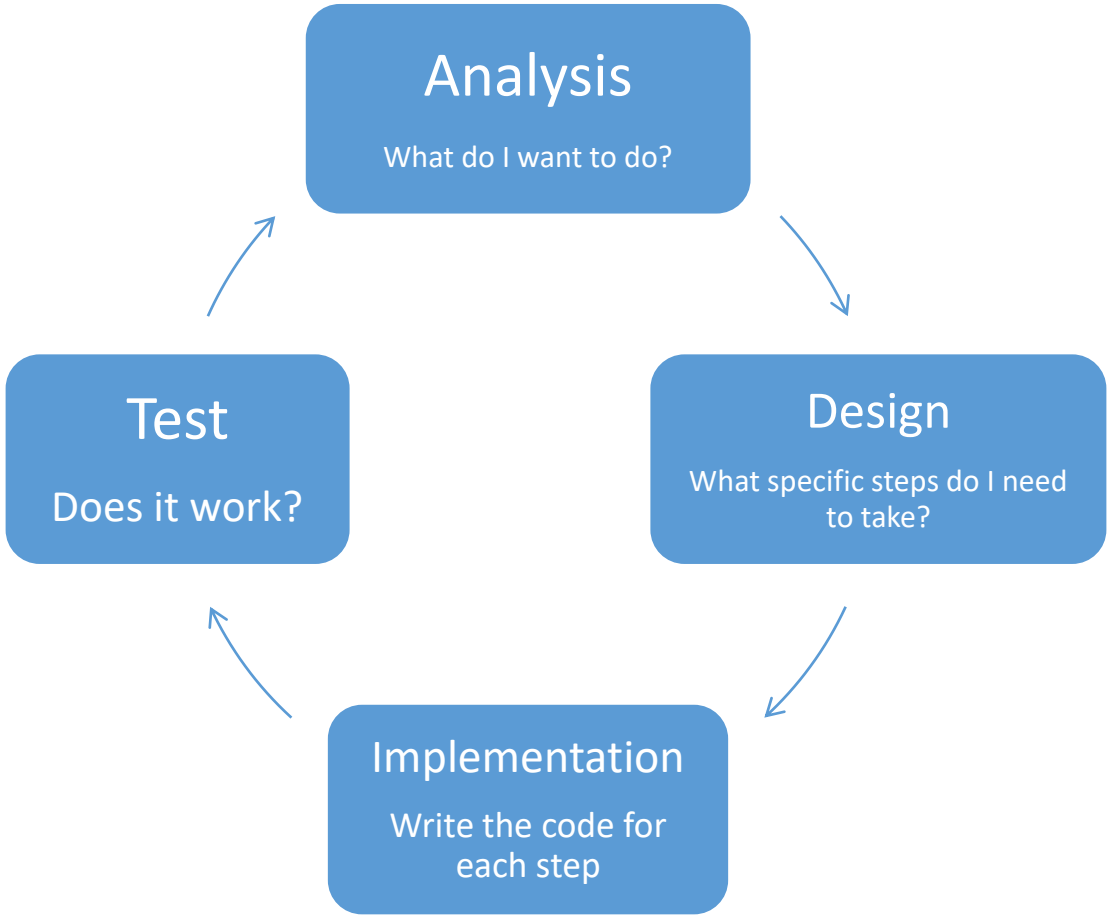
More features
More general → More complexity



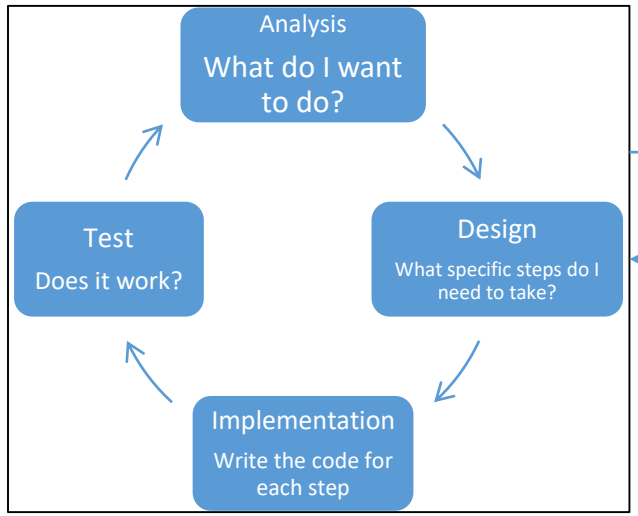
Tools for managing complexity

Organization	Documentation	Code
R package	man pages, code comments	Functions for abstraction
Folder structure	Vignettes (also for developers)	Classes for abstraction
File structure	Github issues, contribution guidelines	Tests

The skateboard



Improving and adding features

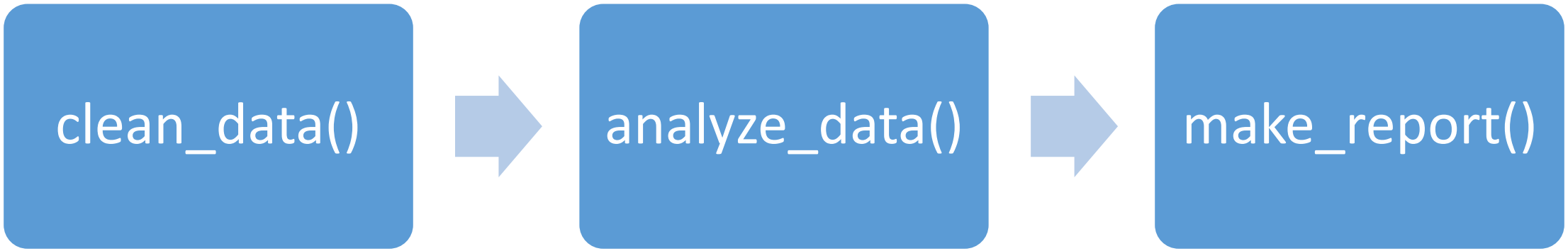


Abstraction

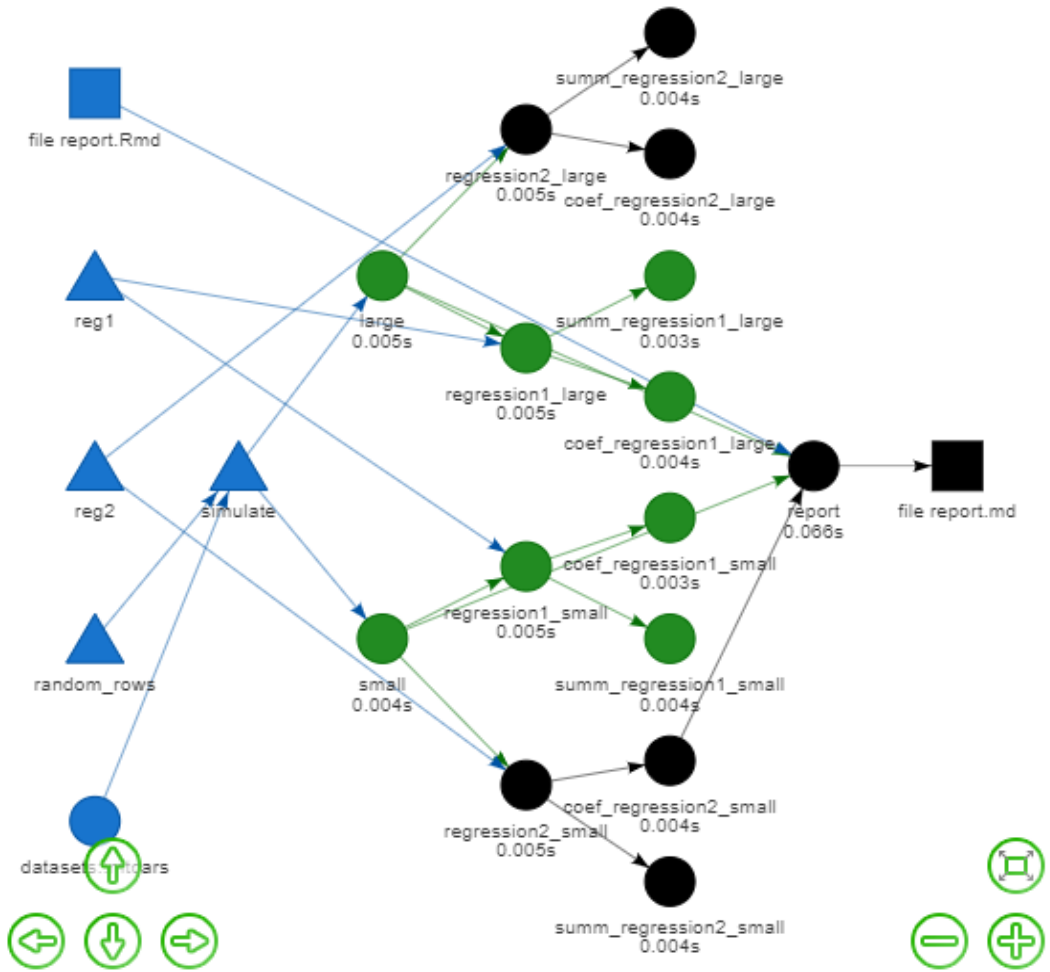
- Moving steps into reusable functions
- Defining classes for data structures

Some examples

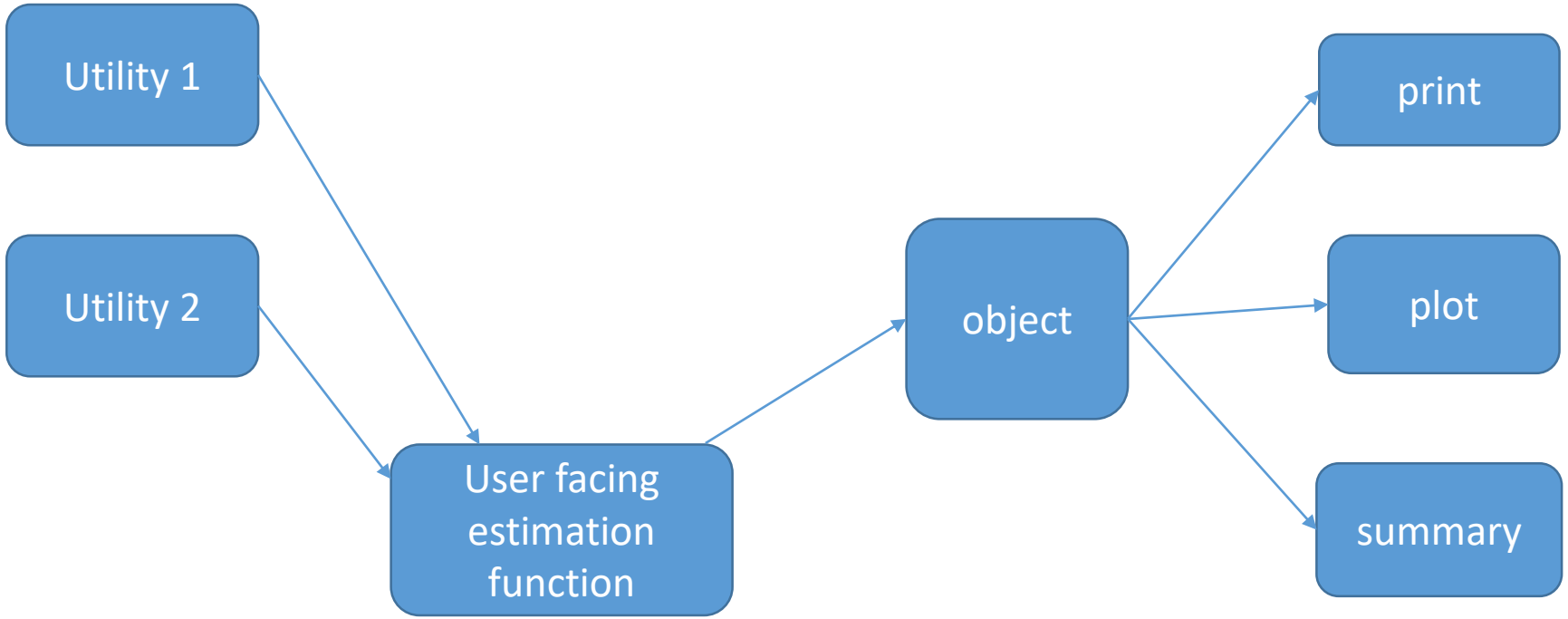
Sequential data analysis



Dependency graph



Statistical method



Summary

- Follow a process, not too rigid, not too flexible
- Use abstraction to help *design* solutions
- R's functional nature is useful for abstraction
- Tools are available to manage complexity at every level