Advanced Programming Techniques

Software Engineering

Christopher Moretti

Software Engineering

Computer Science

Software Development Processes

Abstract Problem Solving

Intermediate-level Design

Theory

Algorithms

Practical Problem Solving

Refactoring

Hardware Systems

Client Services

Human Organization

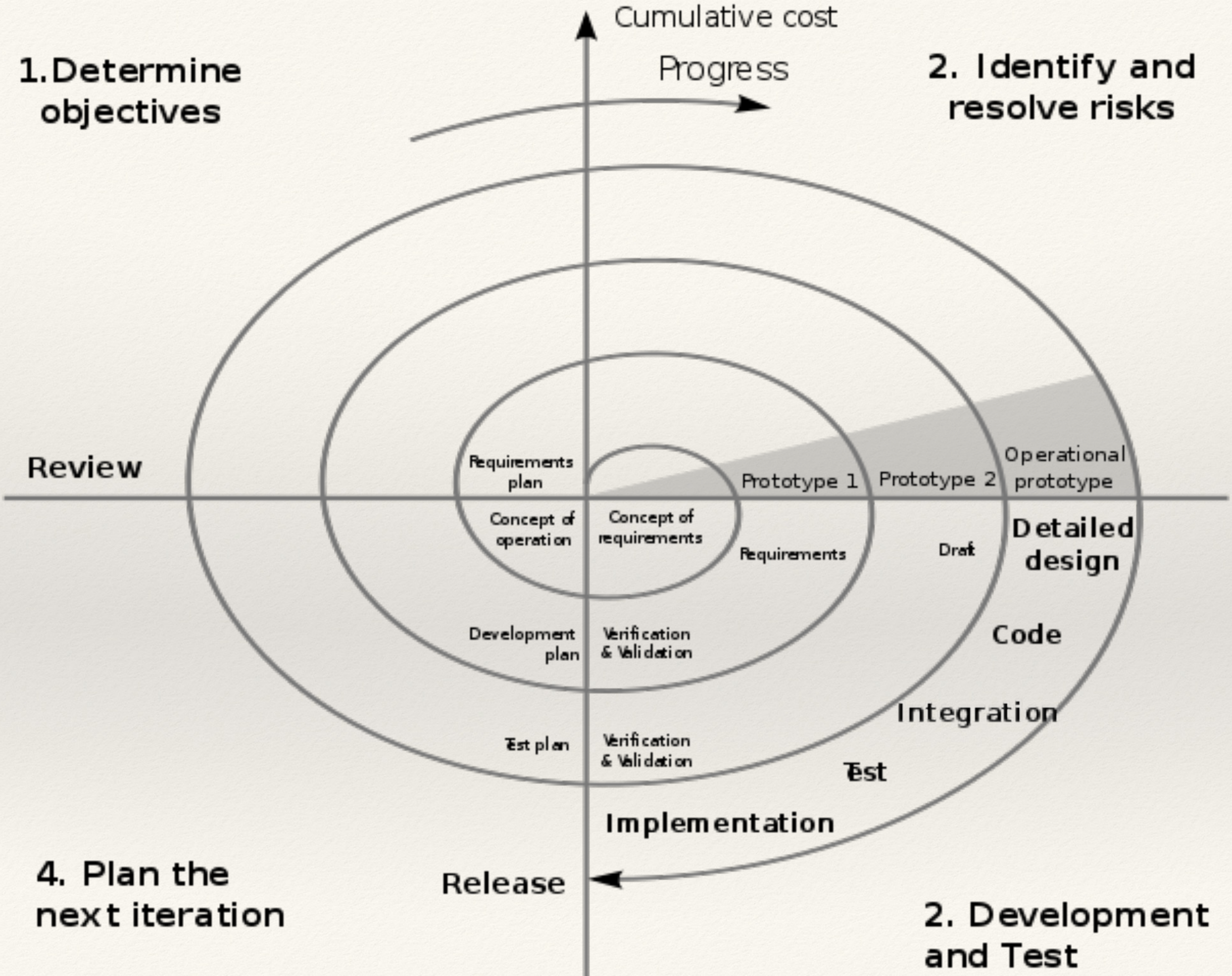
Methods

Engineering

SCIENCE

Software Engineering Stages

- User and System Requirements
  - abstract requirements
  - properties, anti-properties
- Modeling
  - set of components and relationships
  - interactions among components
- Design


Cumulative cost
Progress
1. Determine objectives
2. Identify and resolve risks
Review
Requirements plan
Concept of operation
Concept of requirements
Prototype 1
Prototype 2
Operational prototype
Requirements
Draft
Detailed design
Development plan
Verification & Validation
Code
Integration
Test plan
Verification & Validation
Test
Implementation
Release
4. Plan the next iteration
2. Development and Test

Design Decision: Make -vs- Buy

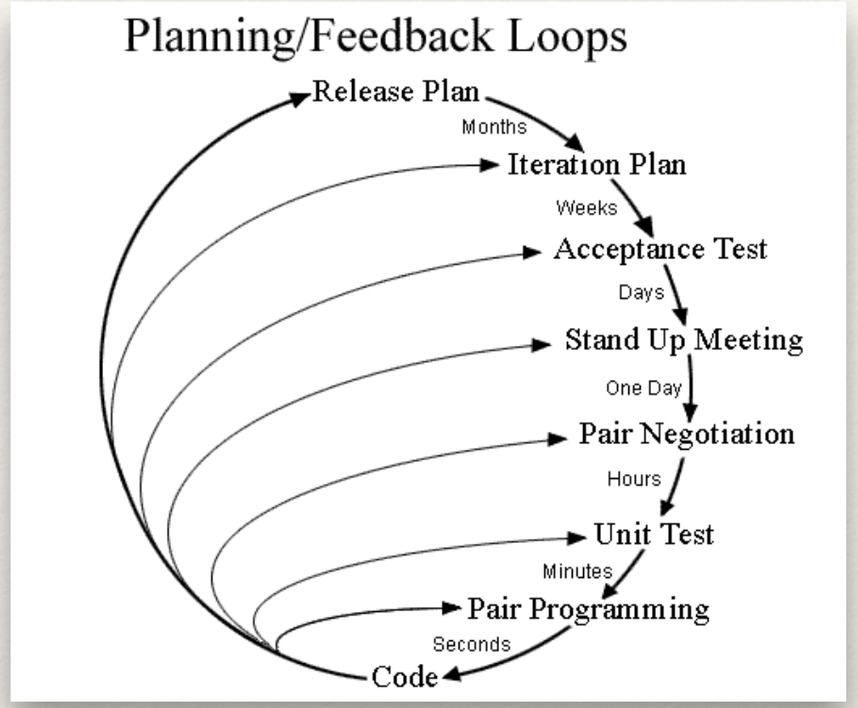
- Off-the-shelf components already exist, require no long-lead planning and design.
  BUT
  Off-the-shelf components almost certainly don't match requirements exactly. You aren't in control of all factors.
- Specially-built systems/components can be tailored to the specification (high compatibility / low reuse), at cost of formal requirements planning, design, implementation, maintenance.

Emergent System Properties

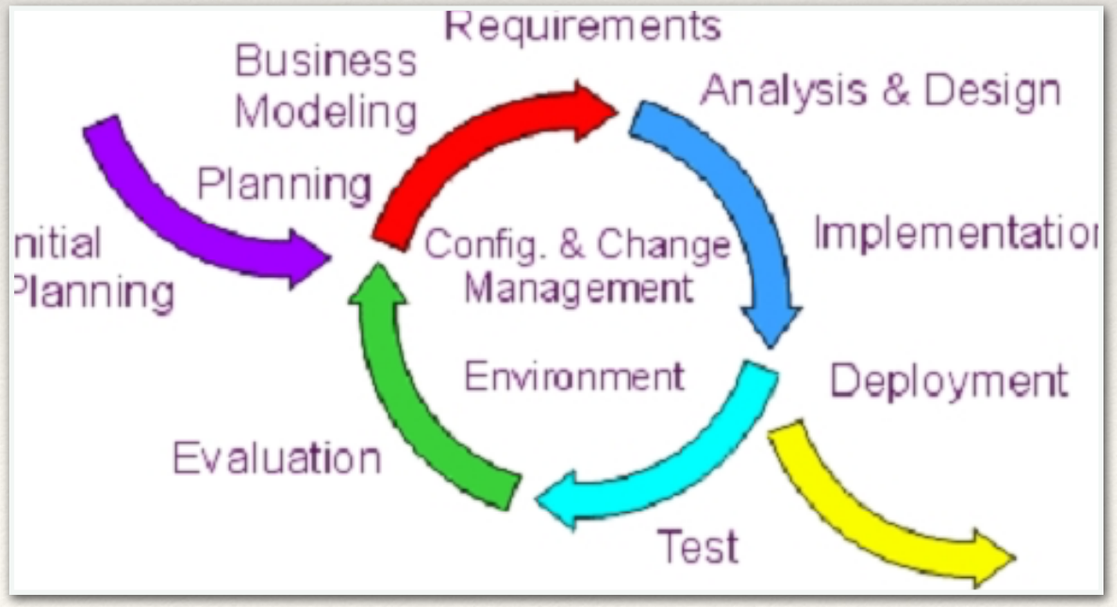
- A system is more than the sum of its parts
  - functional emergence: the functionality is more than the sum of the components (e.g. bicycle)
  - non-functional emergence: behavior in operation
    - reliability
    - performance
    - security
    - robustness / repairability

Post-Deployment Stages

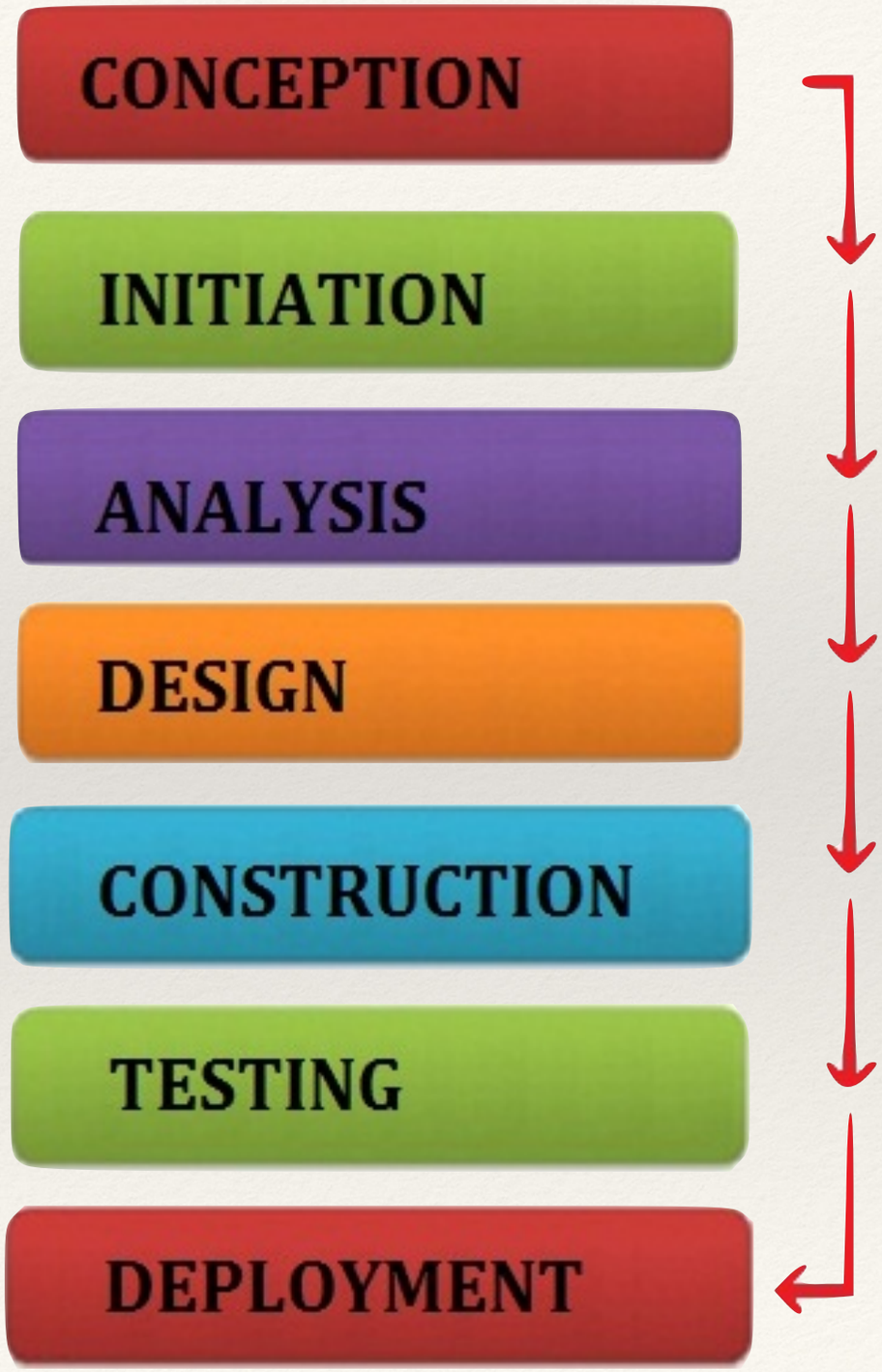
- Large systems should be built to have long lifetimes, thus:
  - correct errors in original design
  - replace components and reassess viability
  - evolve use cases and organizational interaction
- Changes to one subsystem likely impact others
  - modular independence is goal, but need “span” metric
- Reasons for original decisions may be unknown
- Decommissioning may be non-trivial



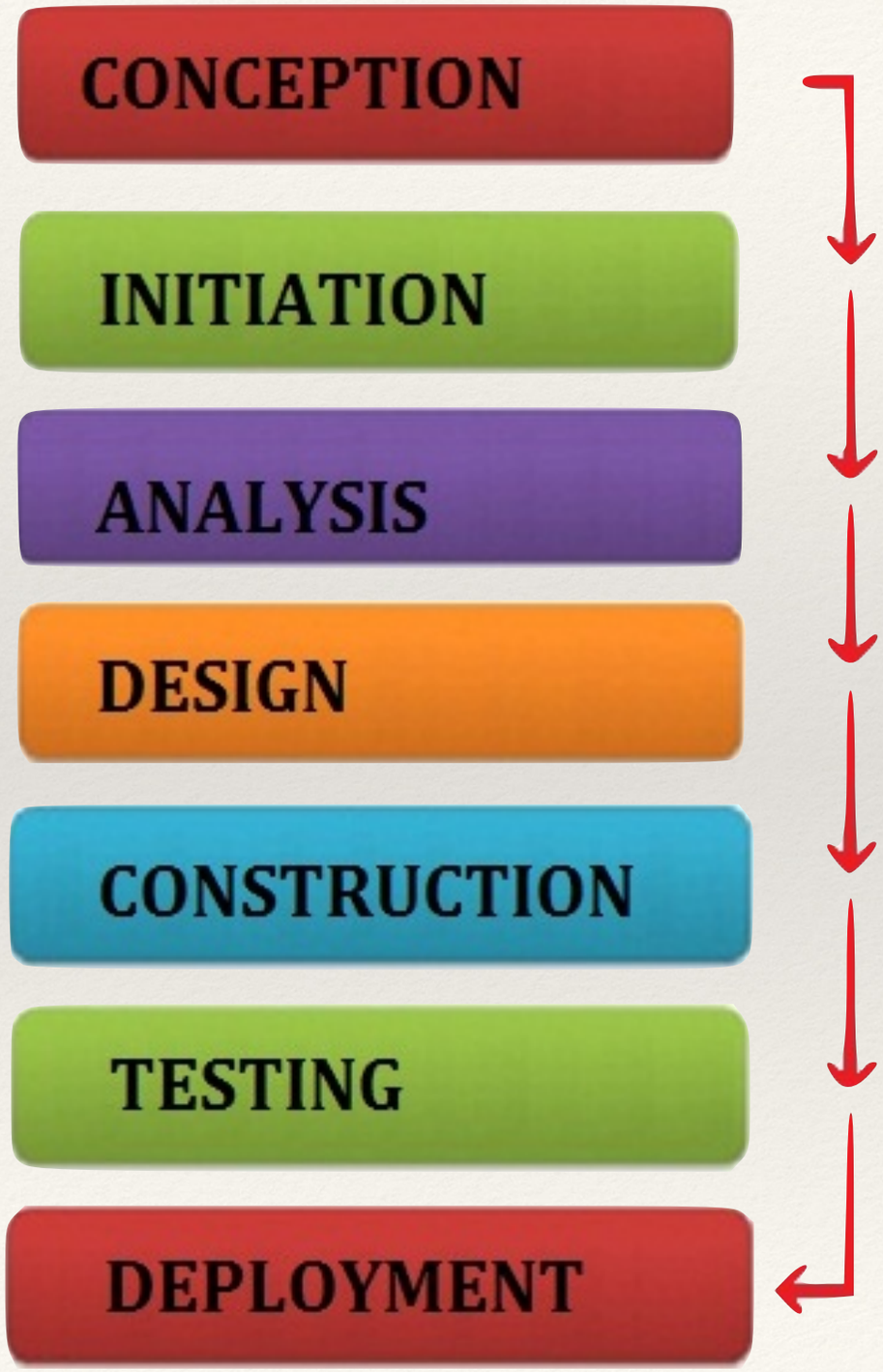
Methodologies



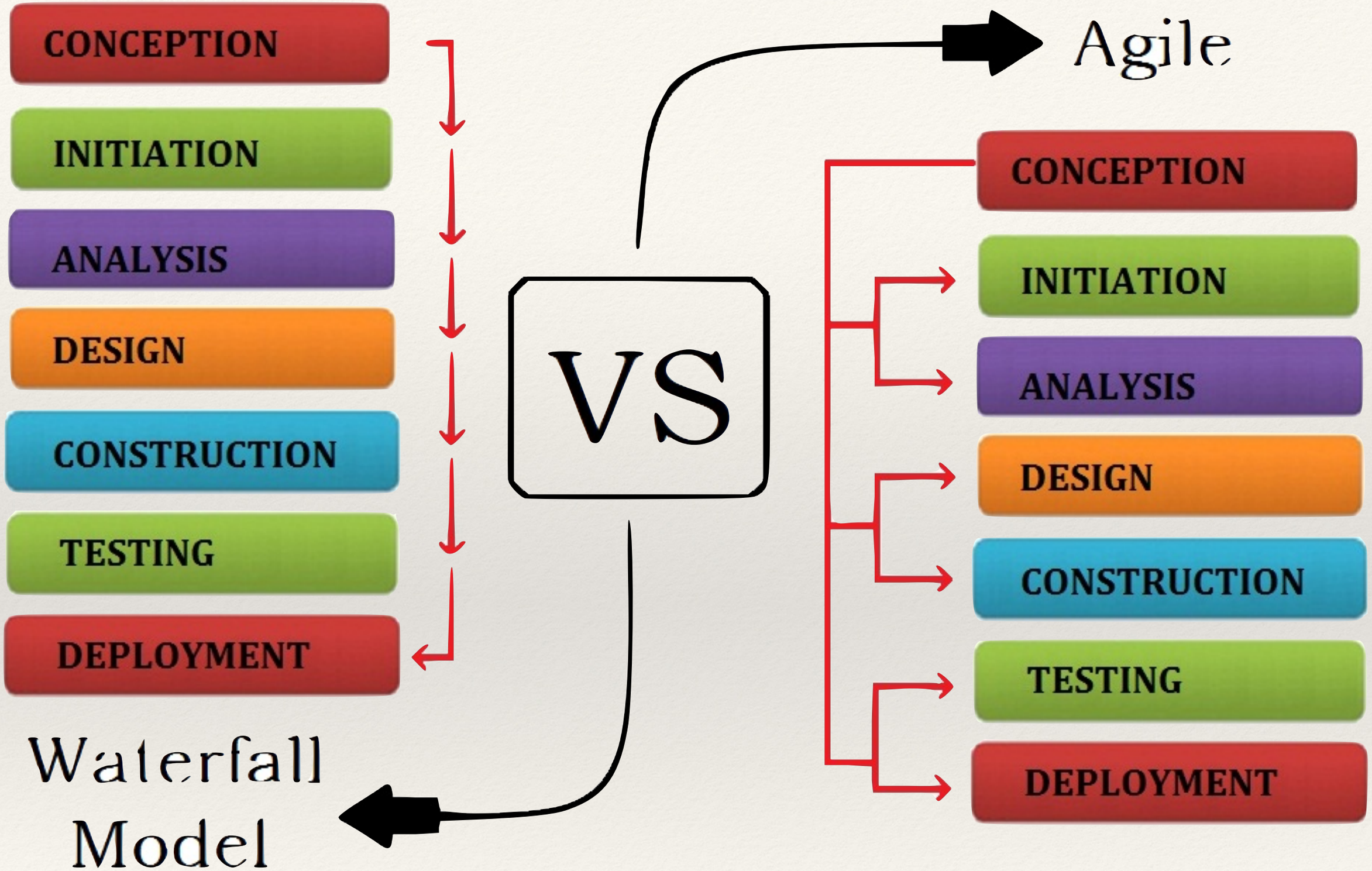
Traditional Waterfall Model



Traditional Waterfall Model



- In principle, next stage doesn't start until previous stage is completely signed off
- In practice, they overlap as a pipeline, or even a series of small cycles (breaks model, but fits better into real world).
- This is generally best suited when requirements are unlikely to change or evolve during dev.

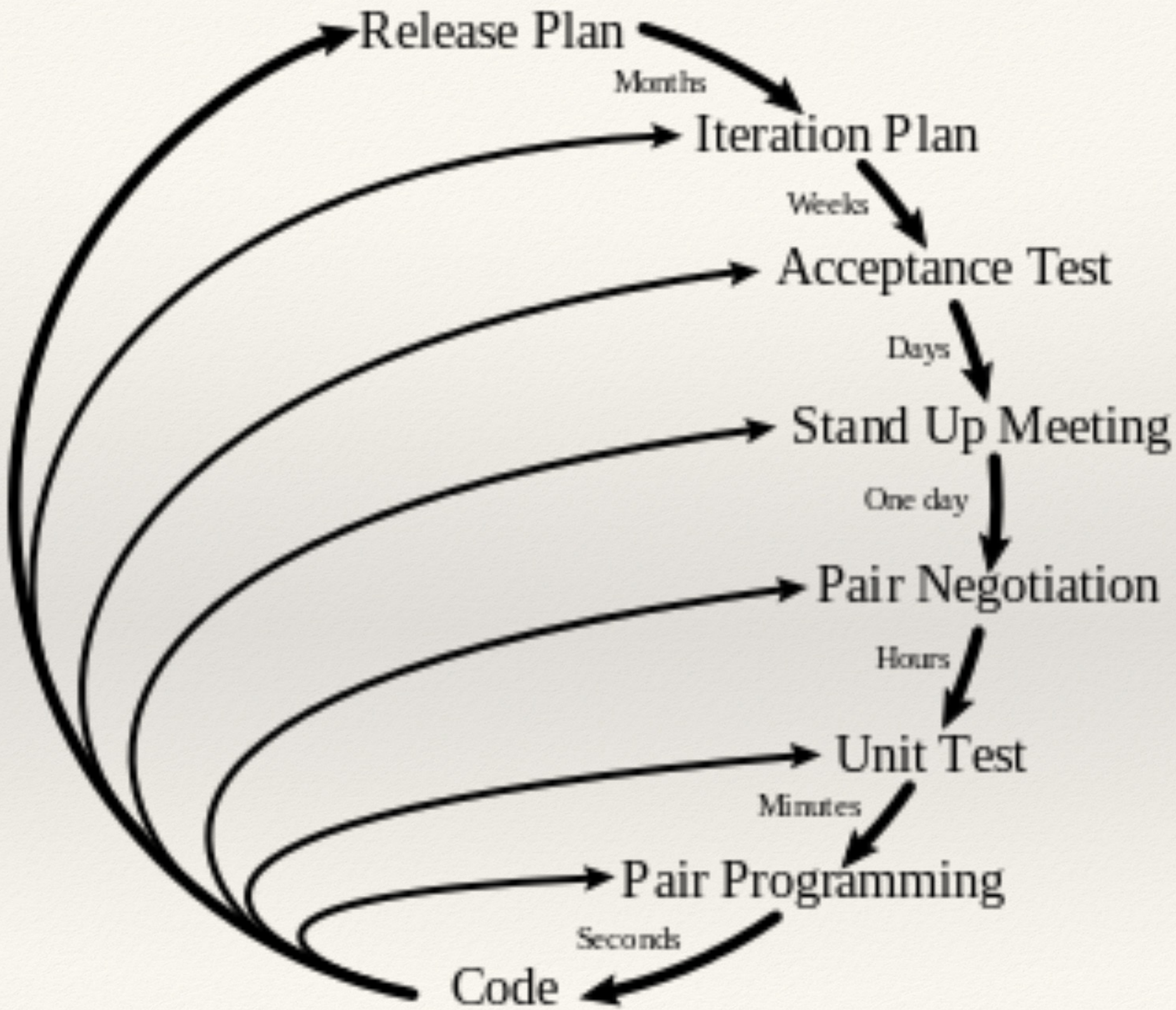

CONCEPTION
INITIATION
ANALYSIS
DESIGN
CONSTRUCTION
TESTING
DEPLOYMENT
Waterfall Model
VS
Agile
CONCEPTION
INITIATION
ANALYSIS
DESIGN
CONSTRUCTION
TESTING
DEPLOYMENT

Agile Manifesto

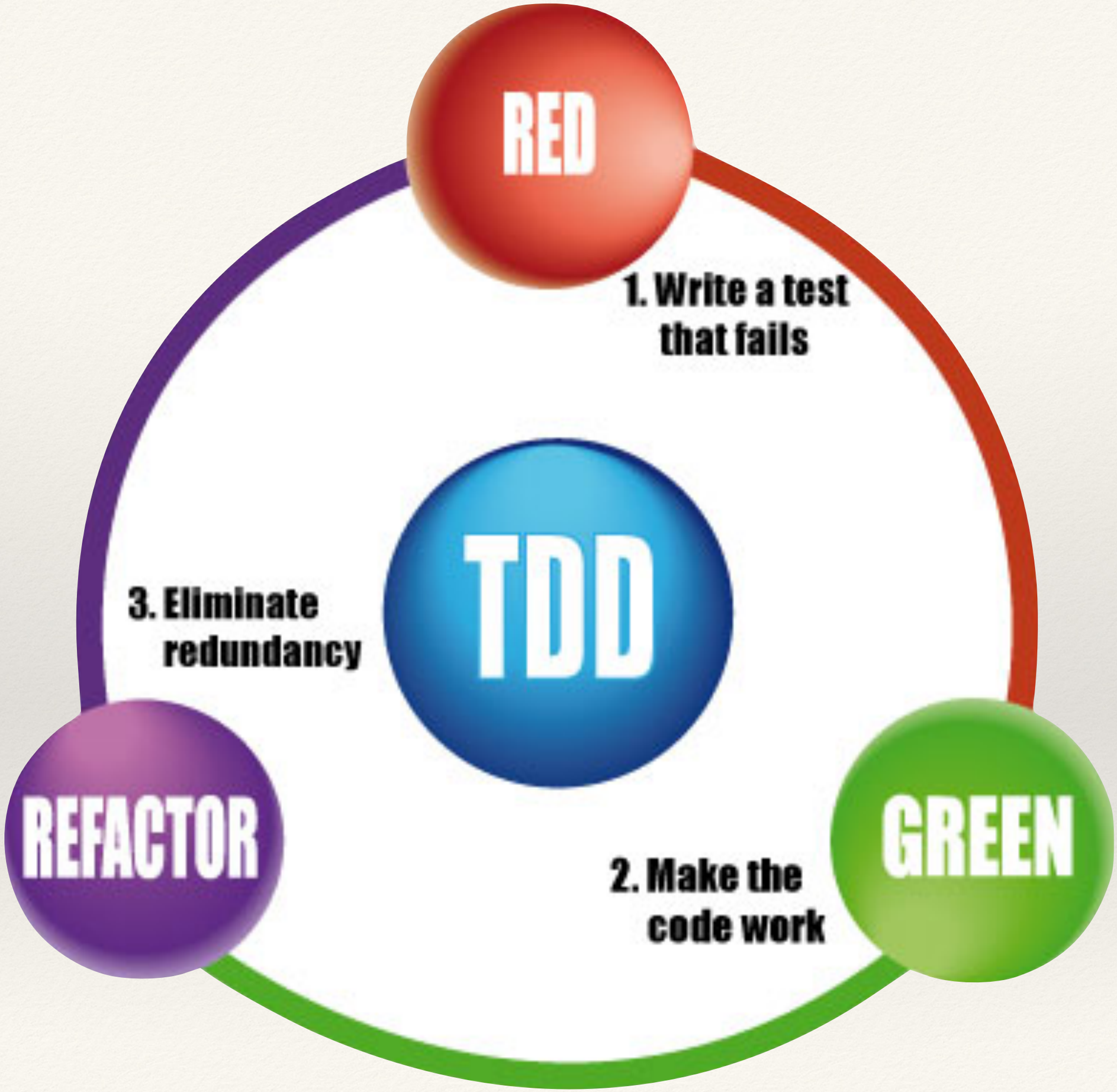
Customer satisfaction by **early delivery** of valuable software
Welcome **changing requirements**, even in late development
Working software is **delivered frequently** (weeks rather than months)
Close, daily **cooperation** between business people and developers
Projects are built around motivated individuals, who should be trusted
Face-to-face conversation is the best form of communication (colocation)
Working software is the principal measure of progress
Sustainable development, able to **maintain a constant pace**
Continuous attention to technical excellence and **good design**
Simplicity—the art of **maximizing work not done**—is essential
Architectures, requirements, designs emerge from **self-organization**
Regularly, the team reflects on how to become more effective, and adjusts

http://www.agilemanifesto.org/principles.html

Planning/Feedback Loops




I'LL GO UP AND FIND OUT
WHAT THEY NEED AND THE
REST OF YOU START CODING

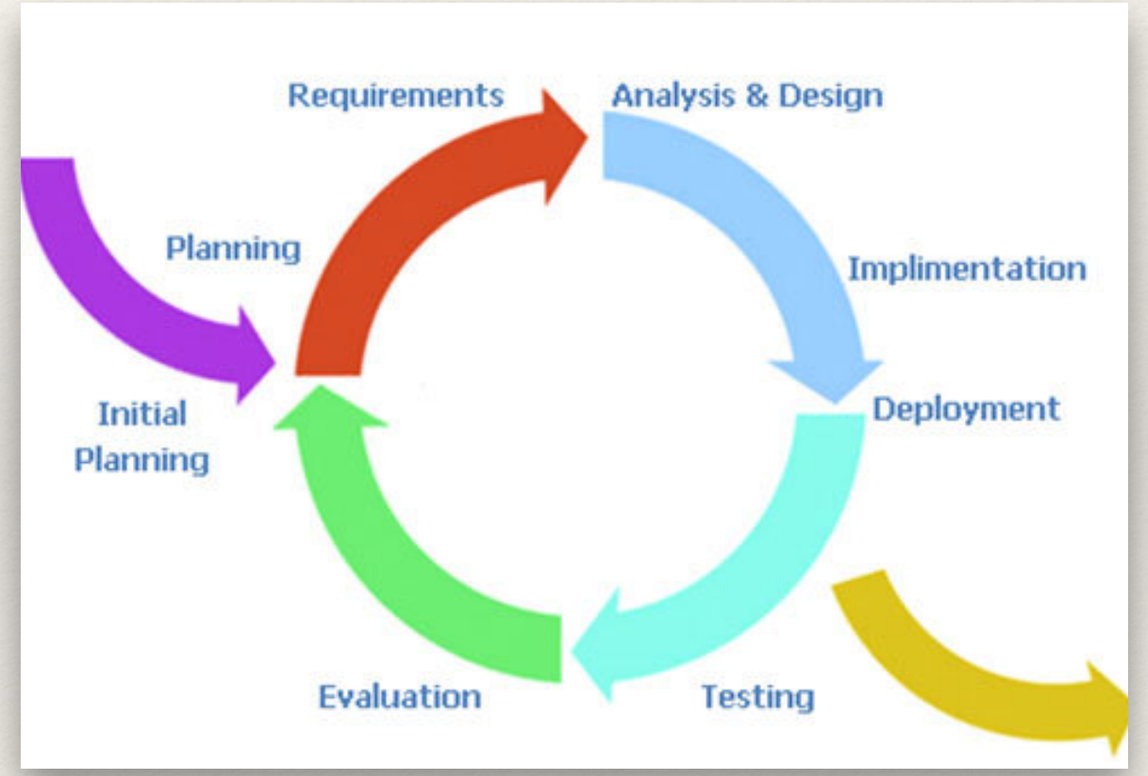

RED
1. Write a test that fails
TDD
3. Eliminate redundancy
REFACTOR
GREEN
2. Make the code work

Evolutionary Development

- Interleaved rather than separate steps, with strong feedback loops encouraging rapid feedback across steps
  - Requirements exploration: what is known? new adds? re-assess suitability? build and test. (back to start)
    - Focus on knowns and build out
  - Throwaway prototyping: what isn't understood yet? prototype one/several plausible solutions/features/versions. Assess.
    - Focus on unknowns and coalesce

RUP

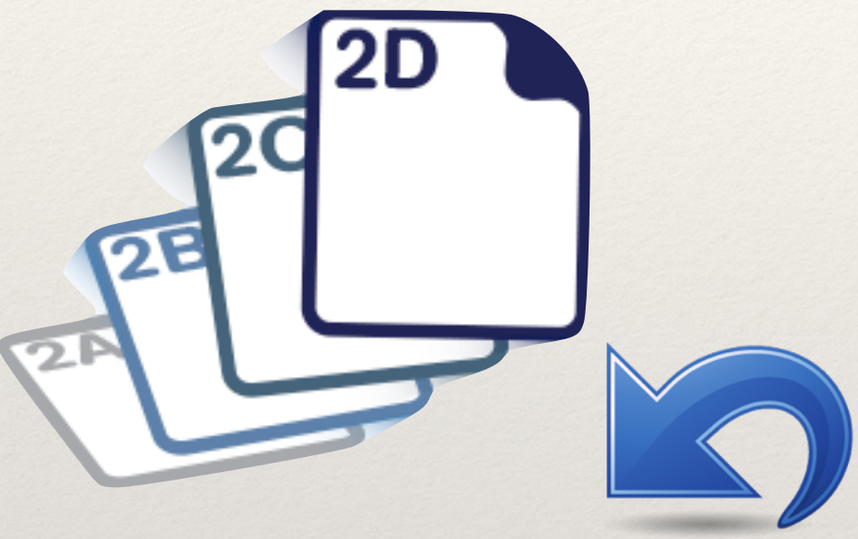
- Hybrid process model — shouldn't rely on one view:
  - Dynamic process over time
  - Static process at each time
  - Emphasis on good practices
- Iteration is supported within each phase of process
- Iteration is supported within larger process as loop of phases

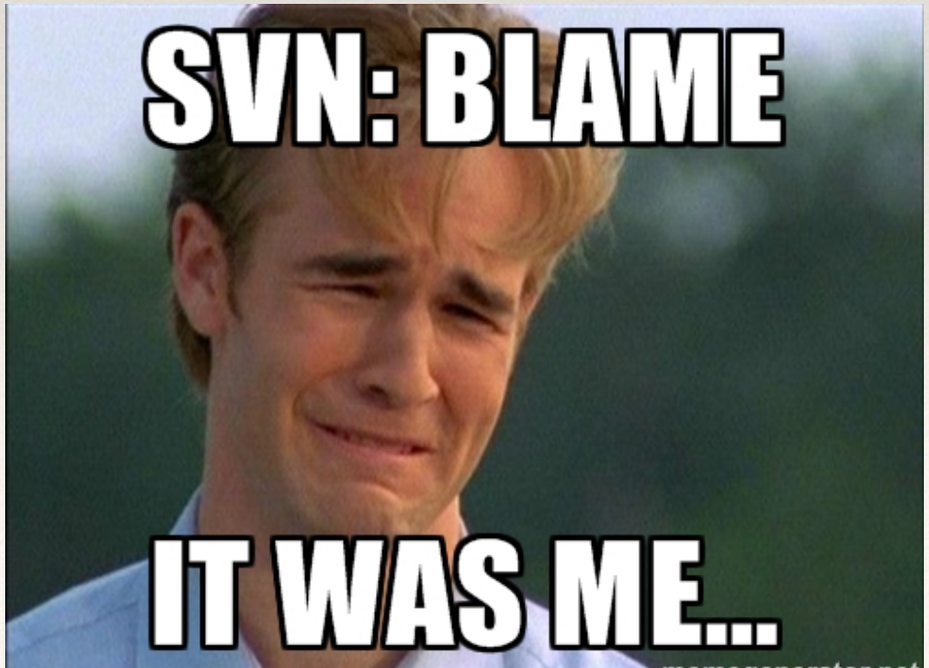



And I should care
Why?
ICANHASCHEEZBURGER.COM

Source Code Management

MANDATORY for COS333 Design Project





Source Management Options

Basic Workflow

- Create a repository that contains all files
  - (plus all old versions / bookkeeping)
- Each author checks out a working copy
  - “copy — modify — merge model”
  - edits in local working copy
- Commits/Check-ins are pushed back to the repository
  - simple conflicts merged automagically
  - true conflicts resolved manually

Subversion (svn)

- `svn co file:///repository.url work.dir`
- `cd work.dir`
- `svn blame existing.txt`
- `$EDITOR existing.txt new.txt`
- `svn add new.txt`
- `svn up`
- `svn status`
- `svn diff existing.txt`
- `svn commit -m “fiddled existing, added new”`

"… my hatred of CVS has meant that I see Subversion as being the most pointless project ever started. The slogan of Subversion for a while was *CVS done right,* or something like that, and if you start with that kind of slogan, there's nowhere you can go. There is no way to do CVS right."

–Linus Torvalds

"I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."

git

- `git clone repository.url`
- `git blame existing.txt`
- `$EDITOR existing.txt new.txt`
- `git add new.txt`
- `git stash && git pull && git stash pop`
- `git status`
- `git diff existing.txt`
- `git commit -m “fiddled existing, added new”`

Survive your last midterms, and have a great break!

NAAAAAAAAAG

Design Document due 3/15
Website, Project Meetings,
Elevator Pitches, starting 3/21