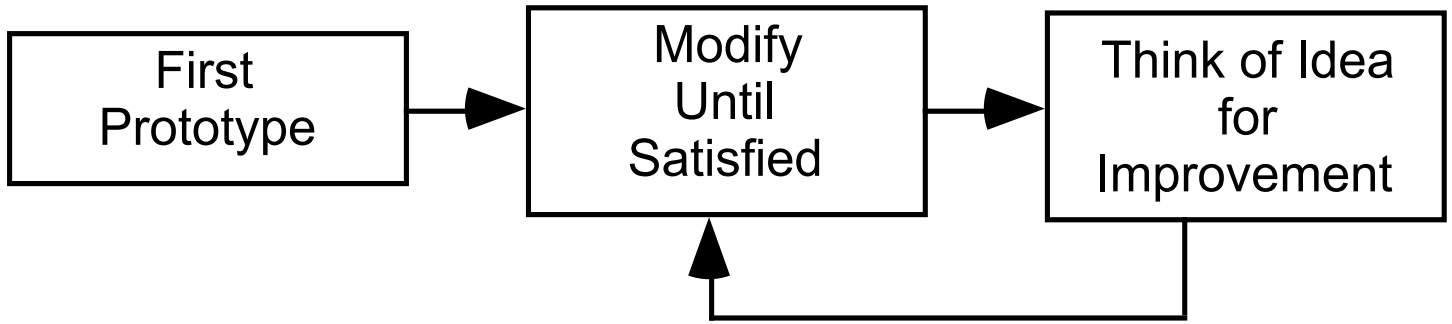
Software Development Life Cycle (SDLC)

The opportunistic approach



- OK for small, informal projects
- Inappropriate for professional environments/complex software where on-time delivery and high quality are expected

Why Life cycle model?

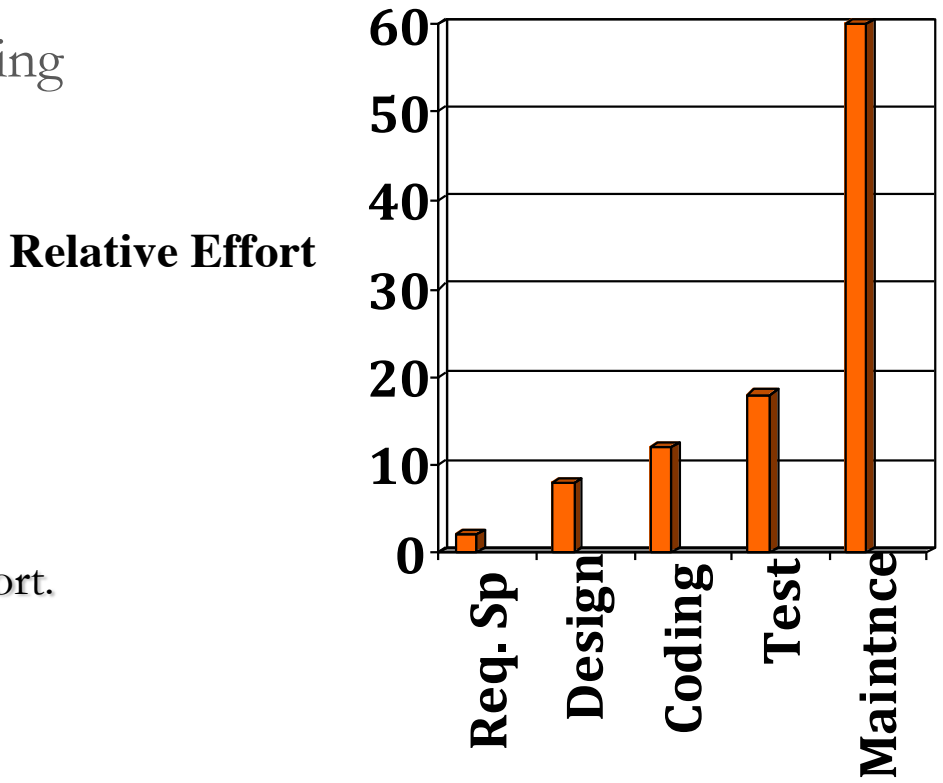
- A software project will never succeed if activities are not coordinated:
  - one engineer starts writing code,
  - another concentrates on writing the test document first,
  - yet another engineer first defines the file structure
  - another defines the I/O for his portion first
- Adherence can lead to accurate status reports
- Otherwise, it becomes very difficult to track the progress of the project
  - the project manager would have to depend on the guesses of the team members.

Software Development Life Cycle (SDLC)

- Typical software life cycle or software process consists of following phases:
  - Feasibility study (involves business case)
  - Requirements analysis and specification,
  - Design
  - Coding
  - Testing
  - Maintenance

Relative effort for Phases

- Phases between feasibility study and testing
  - known as development phases.
- Among all life cycle phases
  - maintenance phase consumes maximum effort.



Feasibility Study

- Main aim of feasibility study: determine whether developing the product
  - financially worthwhile
  - technically feasible.

- First roughly understand what the customer wants:
  - Inputs
  - Processing
  - Outputs
  - various constraints on the behaviour of the system

Requirements Analysis and Specification

- Aim of this phase:
  - understand the exact requirements of the customer,
  - document them properly.

- Consists of two distinct activities:
  - requirements gathering and analysis
  - requirements specification.

Design

- Design phase transforms requirements specification:
  - into a form suitable for implementation in some programming language.

- High-level design:
  - decompose the system into *modules*,
  - represent invocation relationships among the modules.

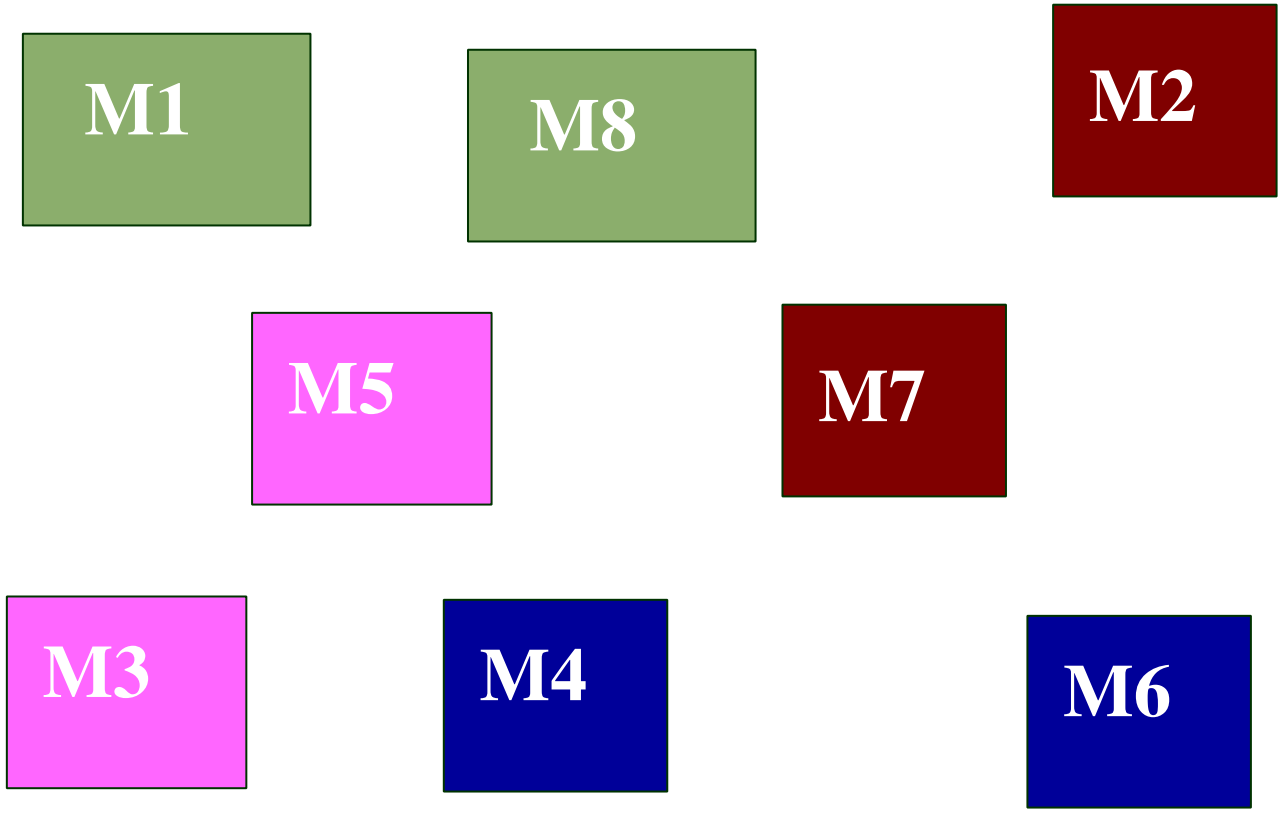
- Detailed design:
  - different modules designed in greater detail:
    - data structures and algorithms for each module are designed.

Implementation

- During the implementation phase:
  - each module of the design is coded,
  - each module is unit tested
    - tested independently as a stand alone unit, and debugged
- The purpose of unit testing:
  - test if individual modules work correctly.
- The end product of implementation phase:
  - a set of program modules that have been tested individually

Integration and System Testing

- Different modules are integrated in a planned manner:
  - modules are almost never integrated in one shot.
  - Normally integration is carried out through a number of steps.
- During each integration step,
  - the partially integrated system is tested.

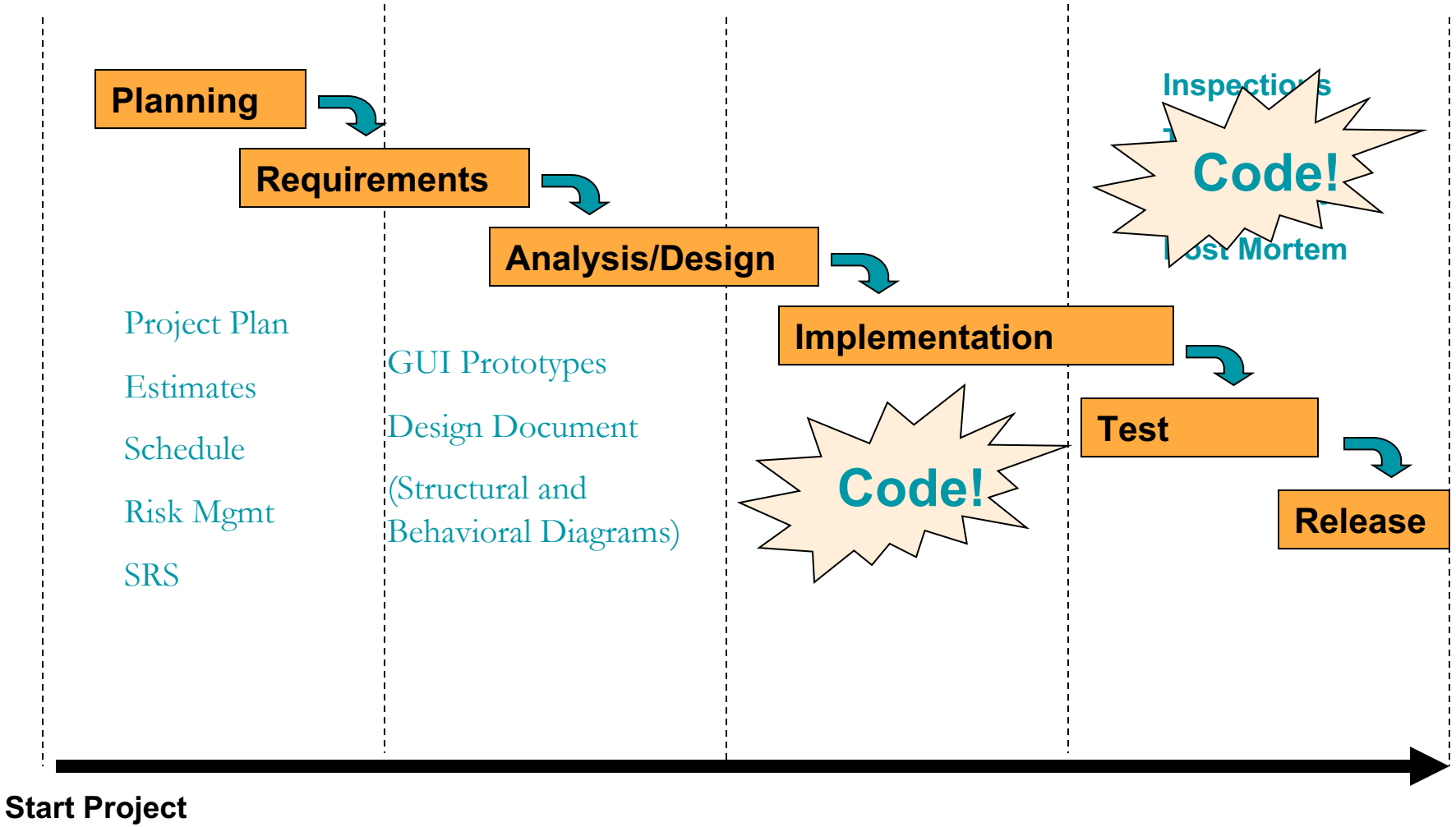


Maintenenace

- Maintenance of any software product:
  - requires much more effort than the effort to develop the product itself.
  - development effort to maintenance effort is typically 40:60.

SE Methods – Then and Now

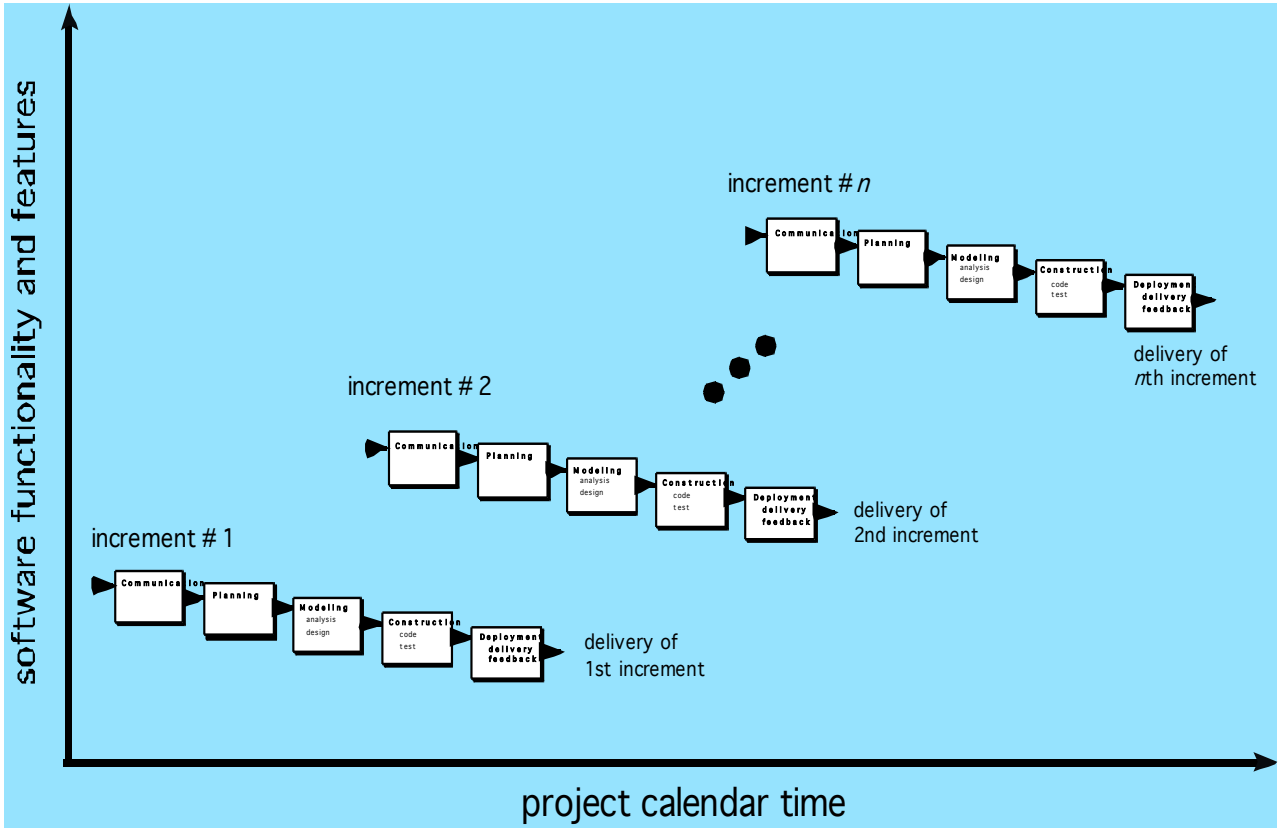
Traditional SDLC. (e.g. waterfall process)



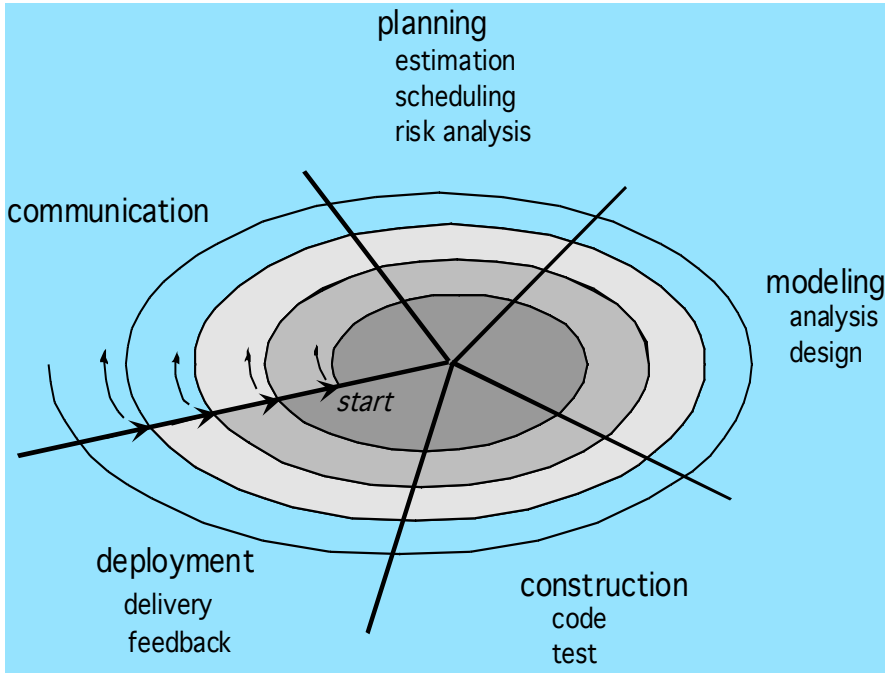
Challenges with traditional (plan-driven) approaches

- Lightweight applications/heavyweight process
- Document intensive (perceived)
- Less flexible design
- Big bang approach to coding/integration
- Testing short-shifted
- One-shot delivery opportunity
- Limited opportunity for process improvement

Software Development Life Cycle – Process models



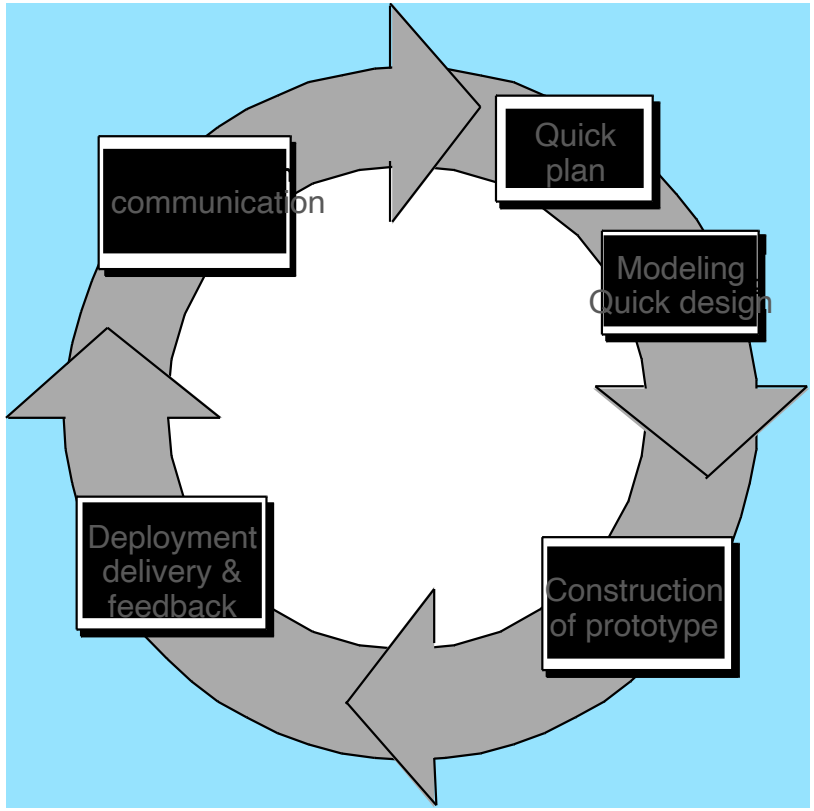
The Incremental Model



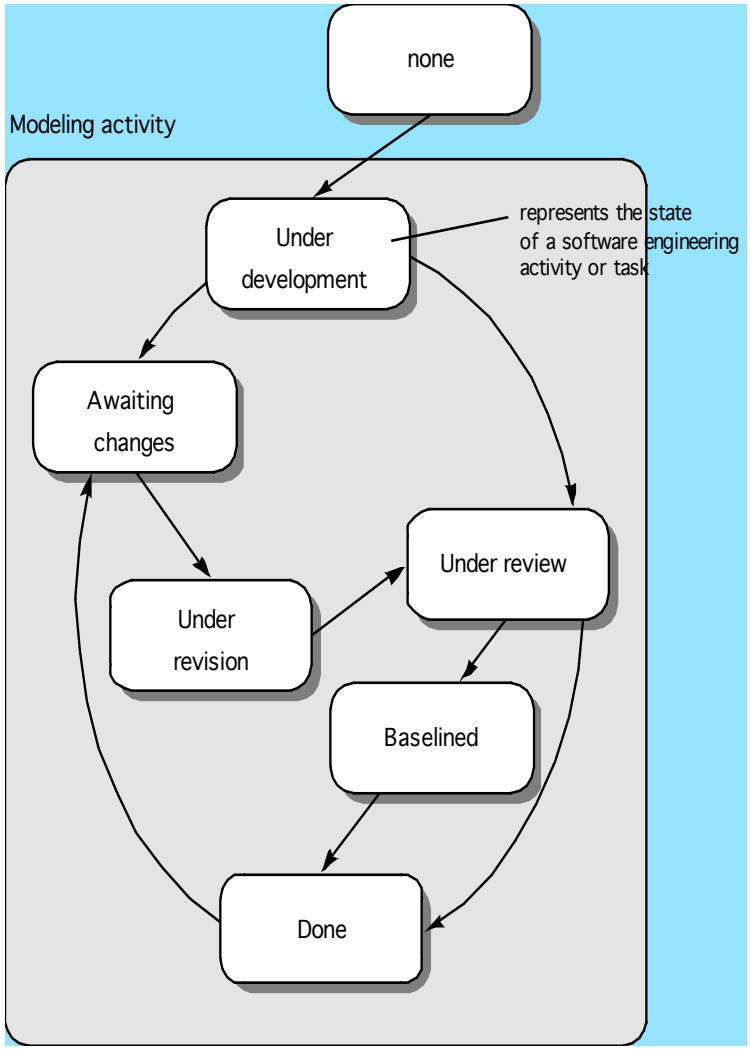
Spiral (meta-model)

Image Courtesy: Sofware Engineering: A Practioner's approach by Roger Pressman

Evolutionary Models



Prototype



Concurrent

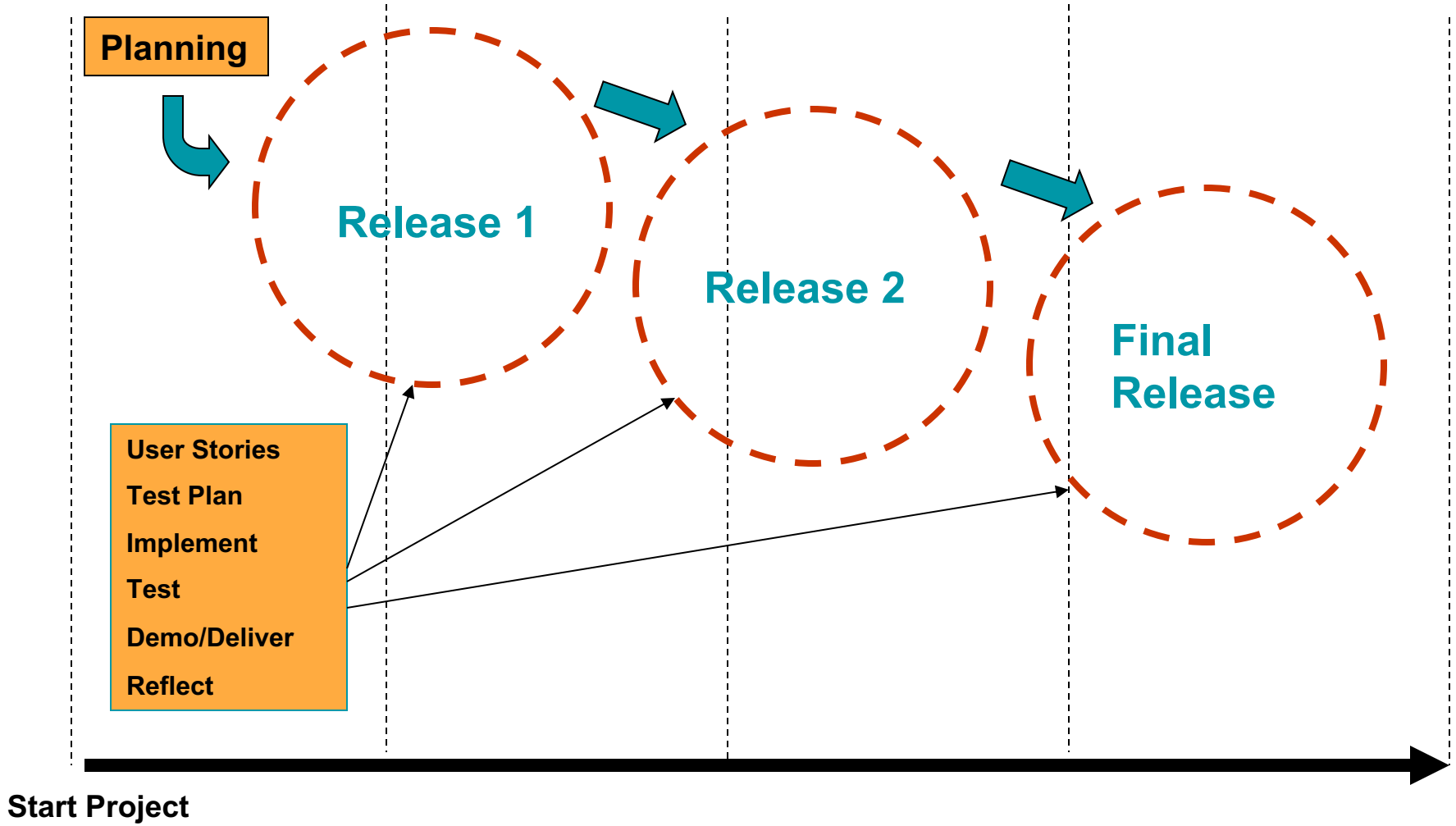
Image Courtesy: Sofware Engineering: A Practioner's approach by Roger Pressman

What Is Agile Software Development?

- In the late 1990's several methodologies began to get increasing public attention. All emphasized:
  - Close collaboration between developers and business experts
  - Face-to-face communication (as more efficient than written documentation)
  - Frequent delivery of new deployable business value
  - Tight, self-organizing teams
  - Ways to craft the code and the team such that the inevitable requirements churn was not a crisis.

- 2001 : Workshop in Snowbird, Utah, Practitioners of these methodologies met to figure out just what it was they had in common. They picked the word "agile" for an umbrella term and crafted the
  - Manifesto for Agile Software Development

Applying Agility



Agile Characteristics

- Incremental development – several releases
- Planning based on user stories
- Each iteration touches all life-cycle activities
- Testing – unit testing for deliverables; acceptance tests for each release
- Flexible Design – evolution vs. big upfront effort
- Reflection after each release cycle
- Several technical and customer focused presentation opportunities

Key Agile Components

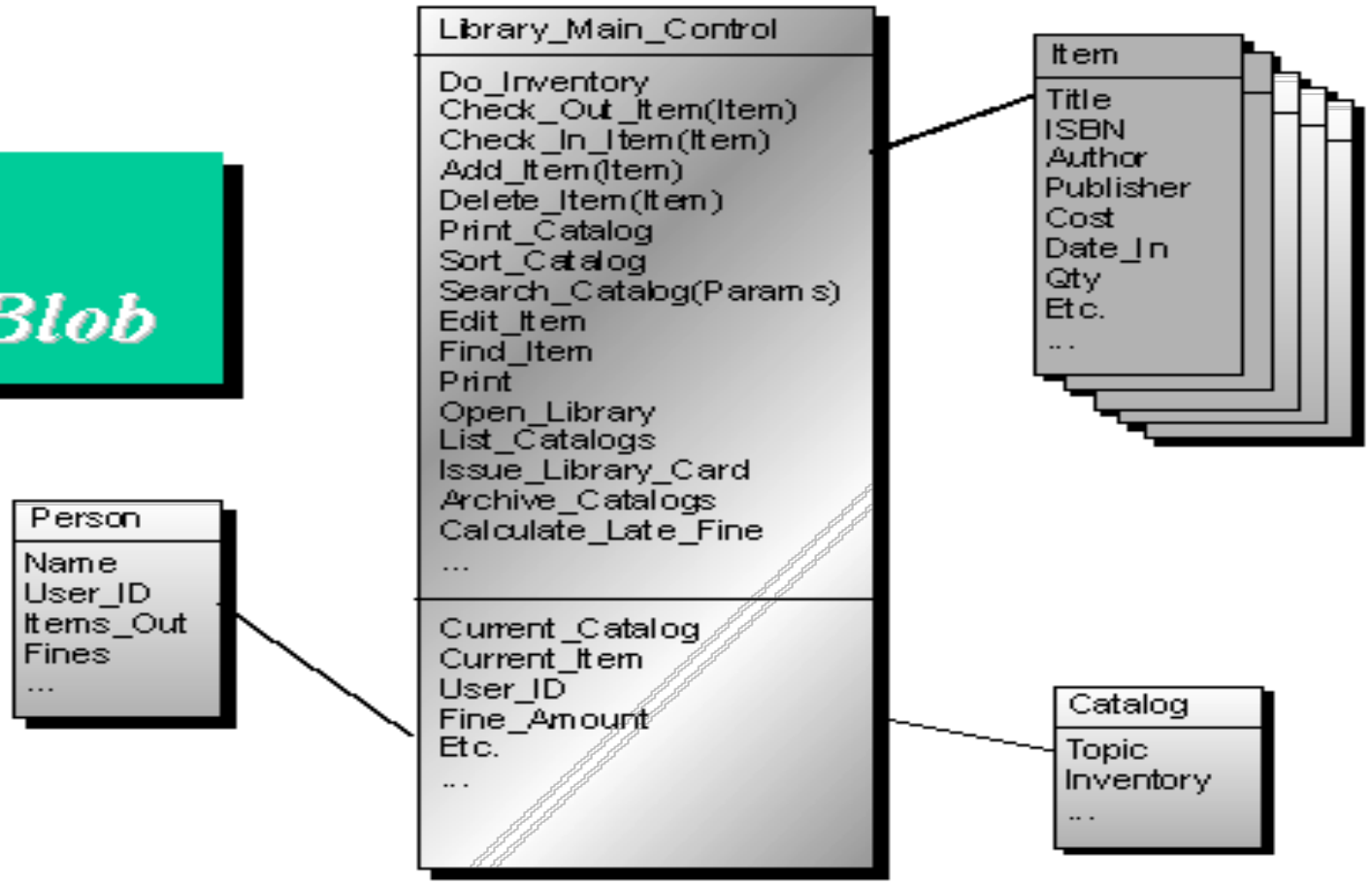
- User Stories
  - Requirements elicitation
  - Planning – scope & composition
- Evolutionary Design
  - Opportunity to make mistakes
- **Test driven development**
  - Dispels notion of testing as an end of cycle activity
- **Continuous Integration**
  - Code (small booms vs big bang)
- **Refactoring**
  - Small changes to code base to maintain design entropy
- Team Skills
  - Collaborative Development (Pair programming)
  - Reflections (process improvement)
- Communication/shared ownership
  - Interacting with customer / team members

BugZilla, JIRA, Kanboard, etc are some of the agile tools

Refactoring - Library system example – Existing design

What areas do you see as potential problem areas? Why did you identify each of those areas?

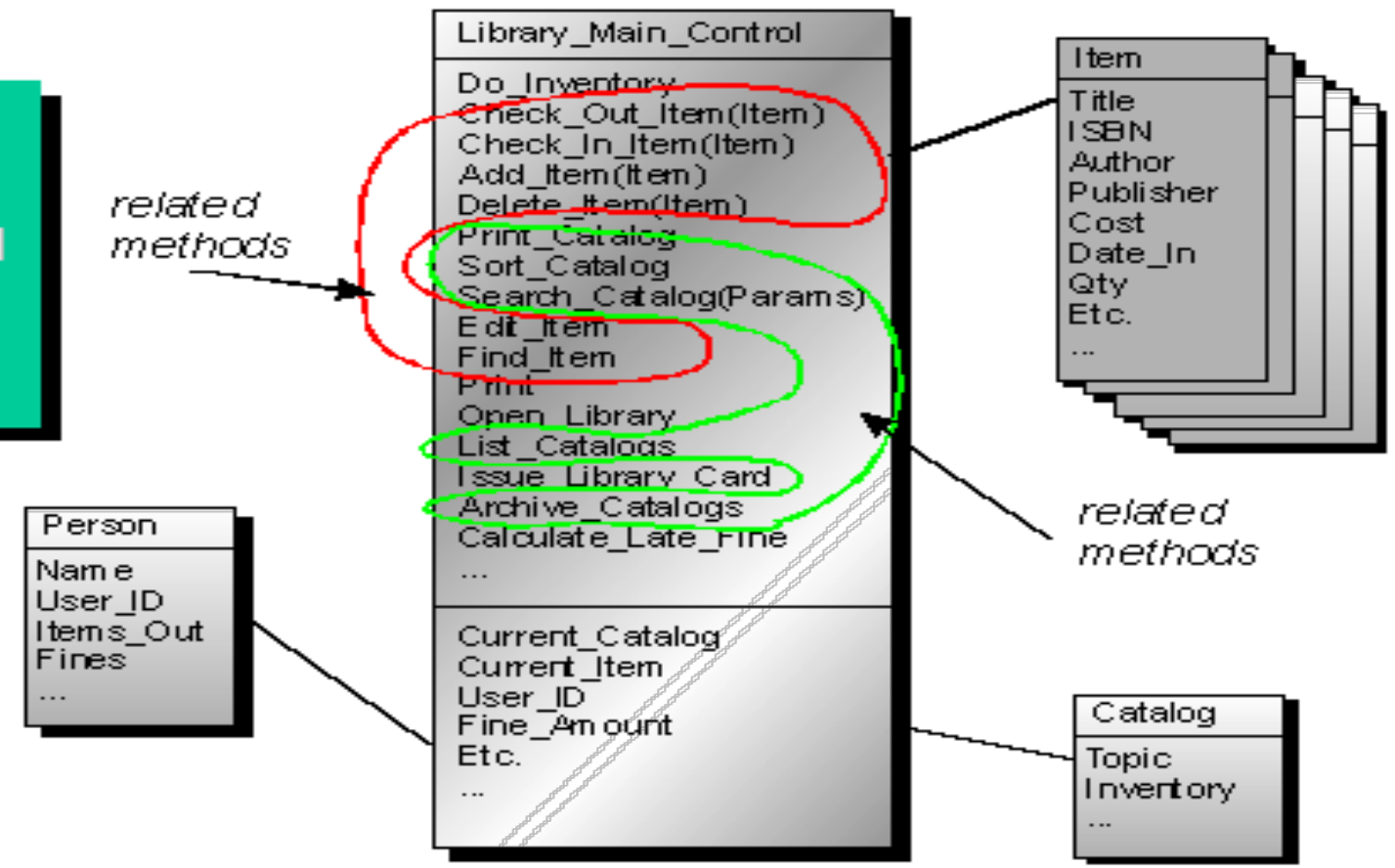
Example:
The Library Blob



Source: MITRE

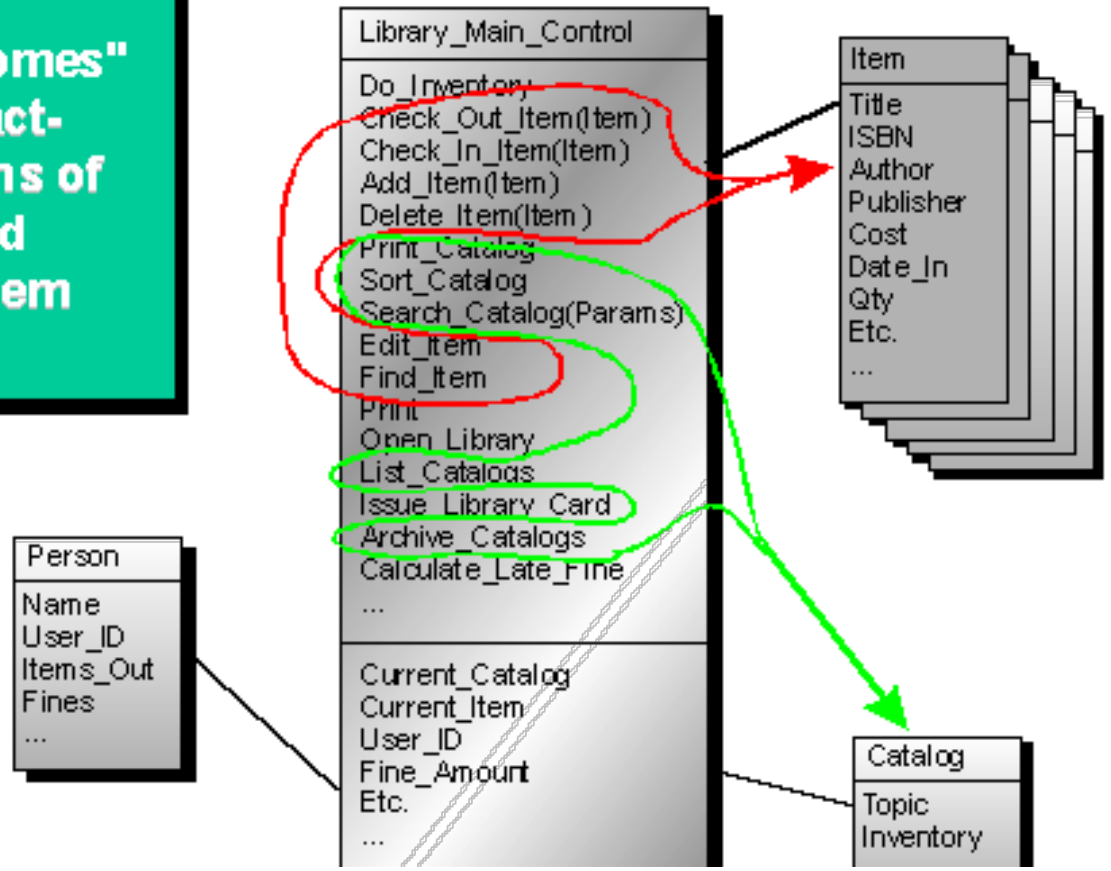
Library system – Changing the design

Step 1:
Identify or categorize related attributes and operations according to contracts.

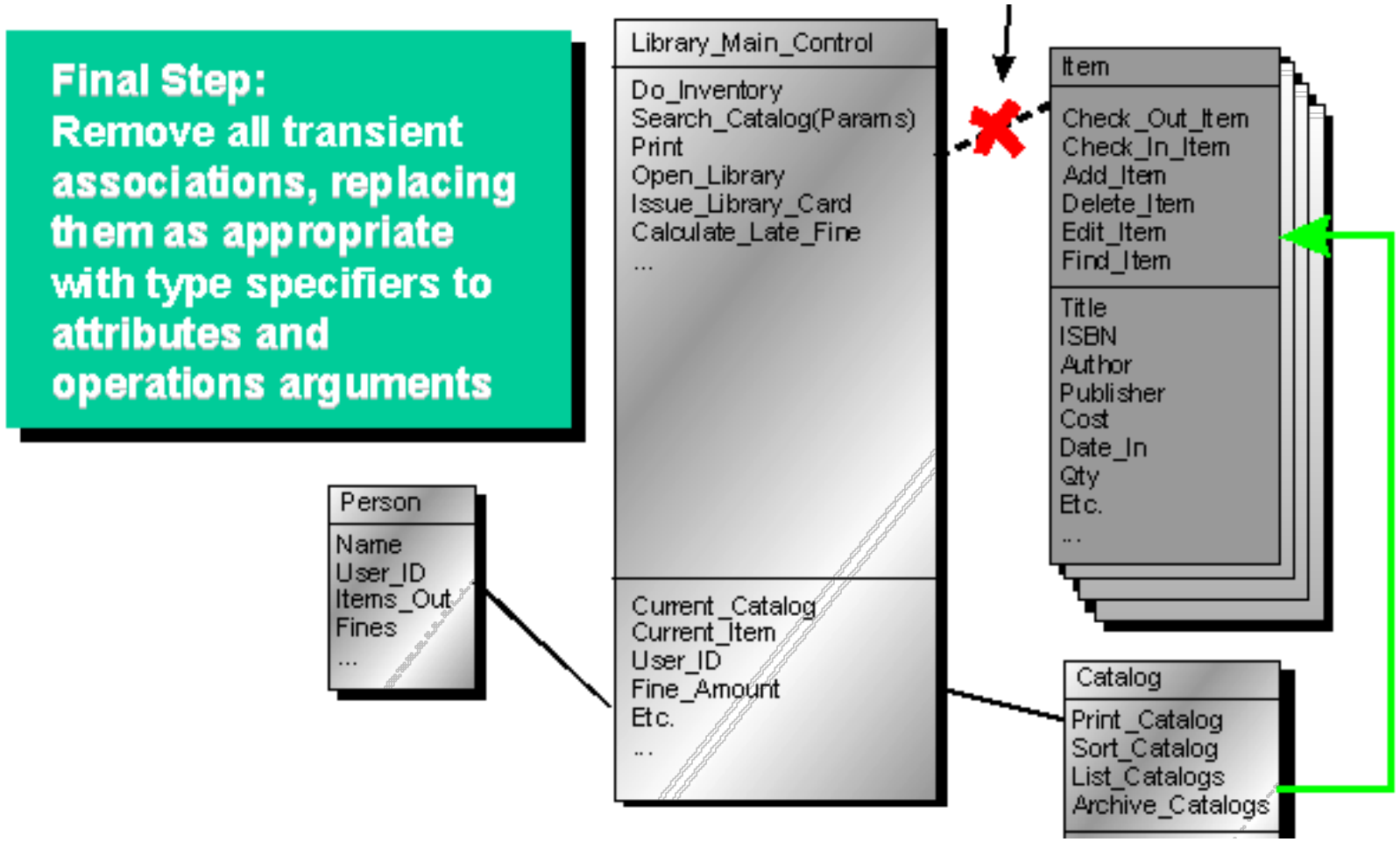


Library system – Changing the design

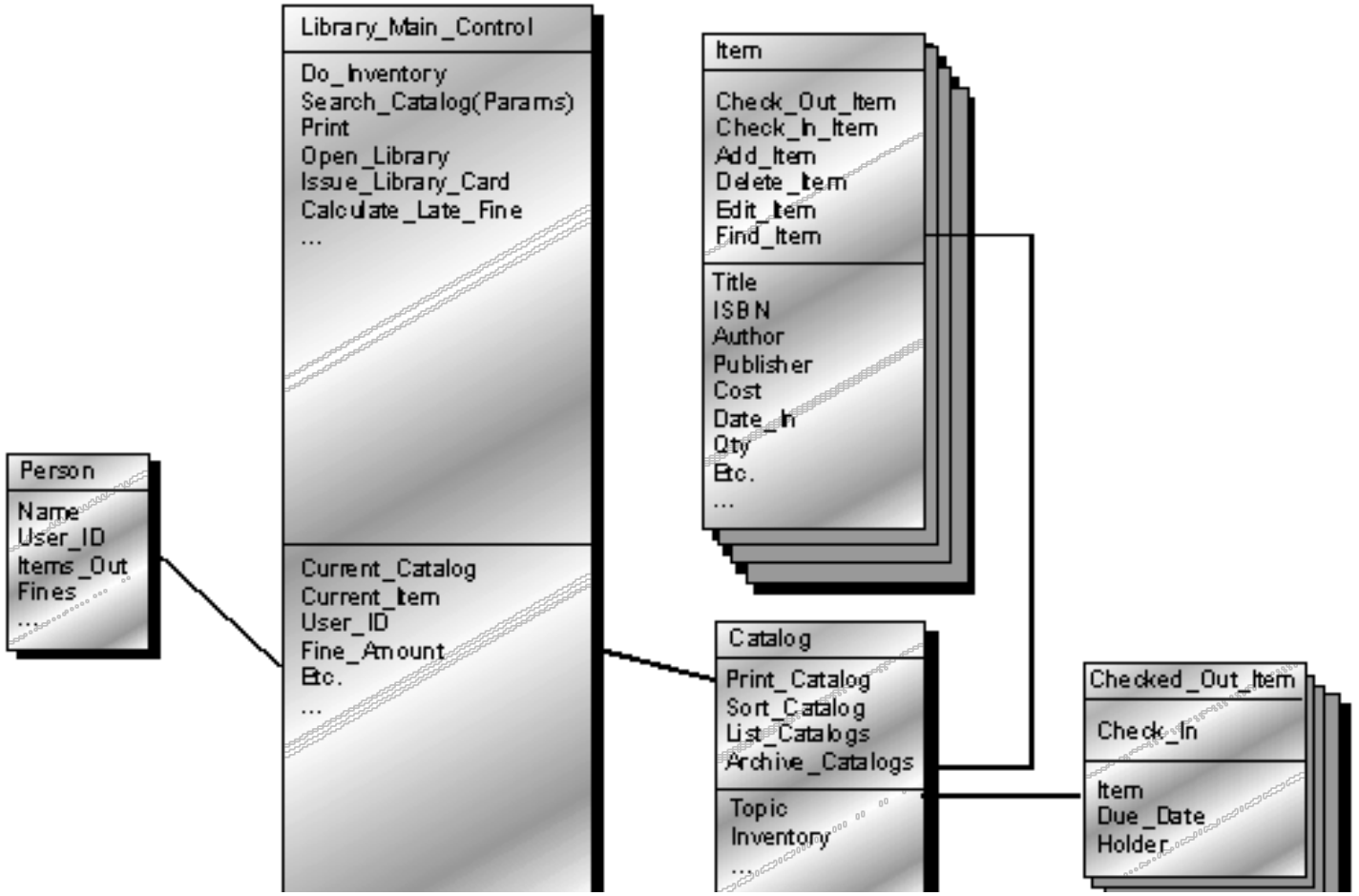
Step 2:
Find "natural homes" for these contract-based collections of functionality and them migrate them there



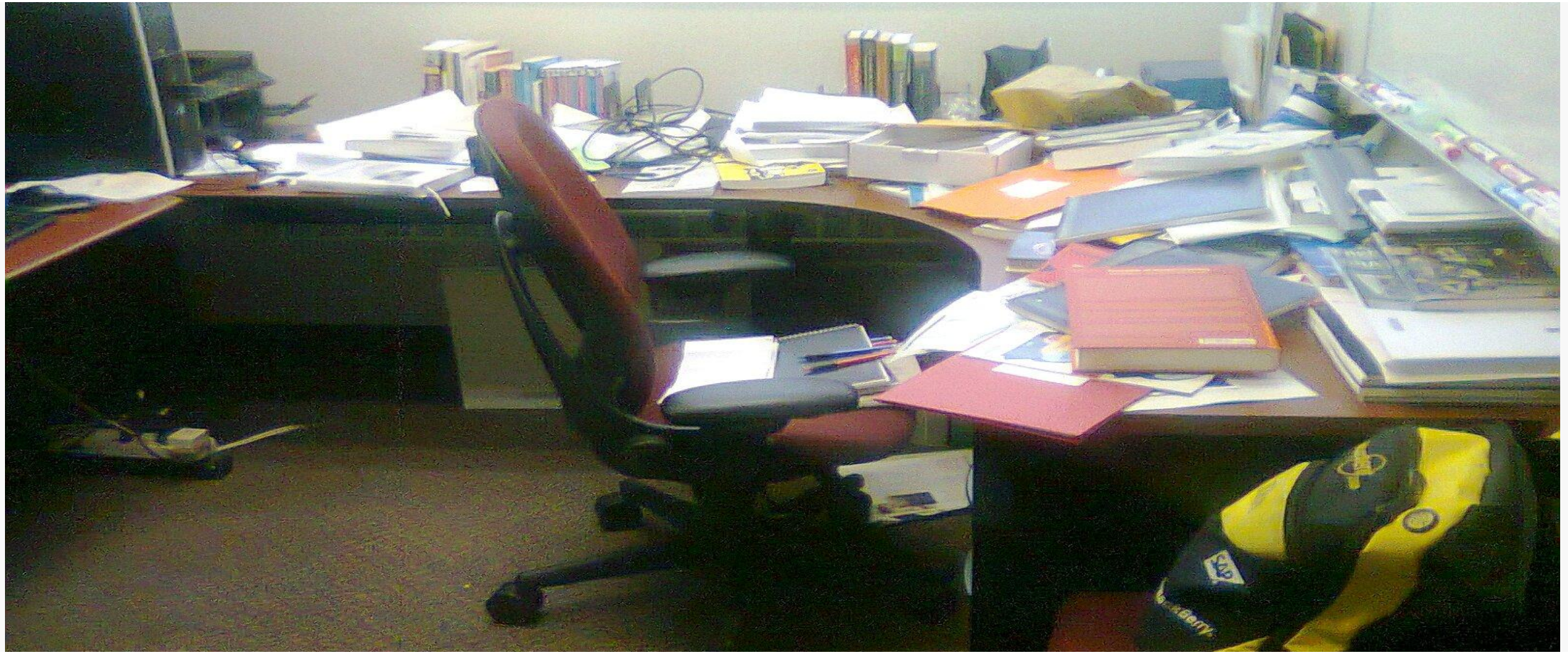
Library system – Changing the design



Library system – Changing the design



From this…

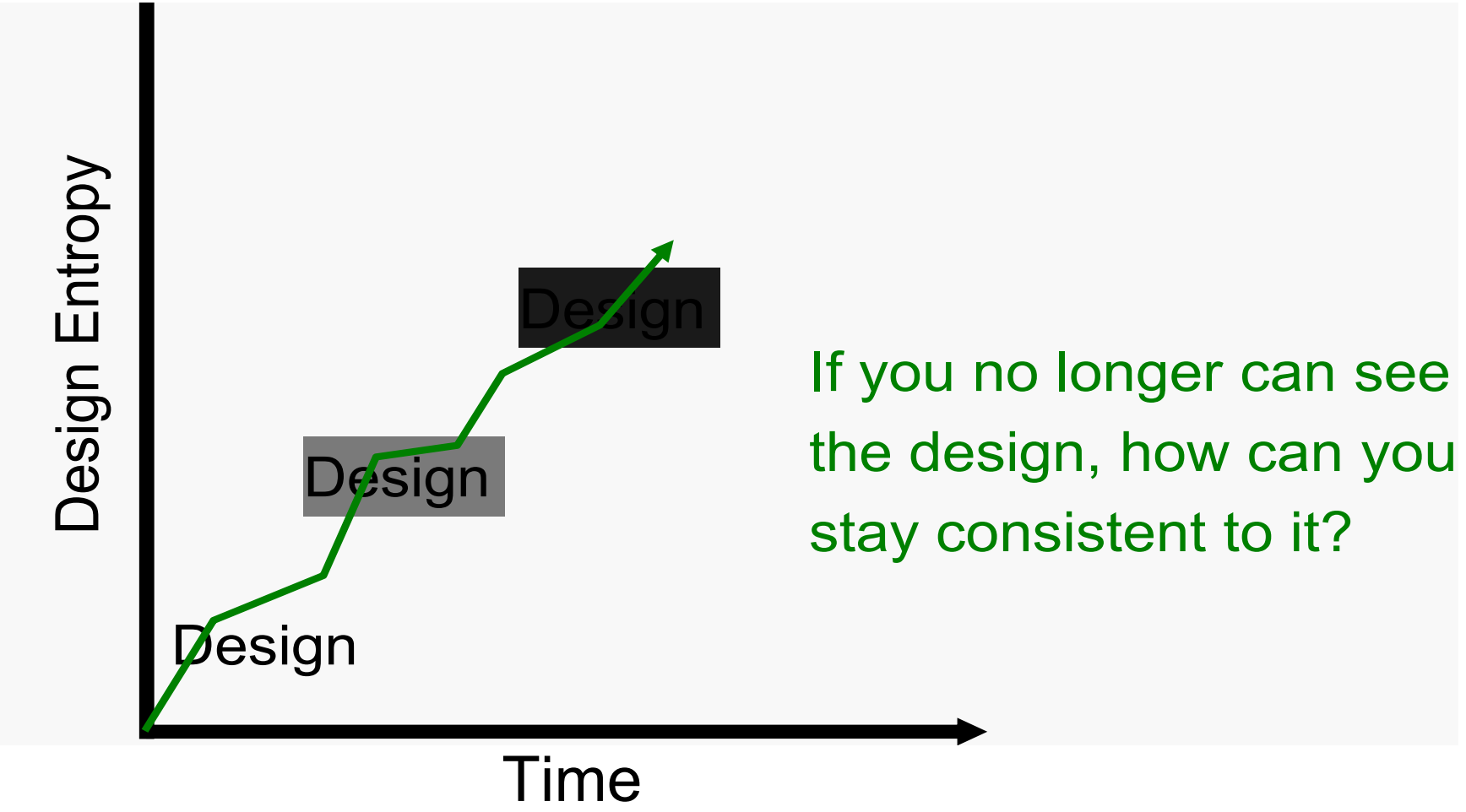


Source: Mike Lutz, RIT

… To this

Source: Mike Lutz, RIT

Design Entropy Vs Time



Test Driven Development (TDD)

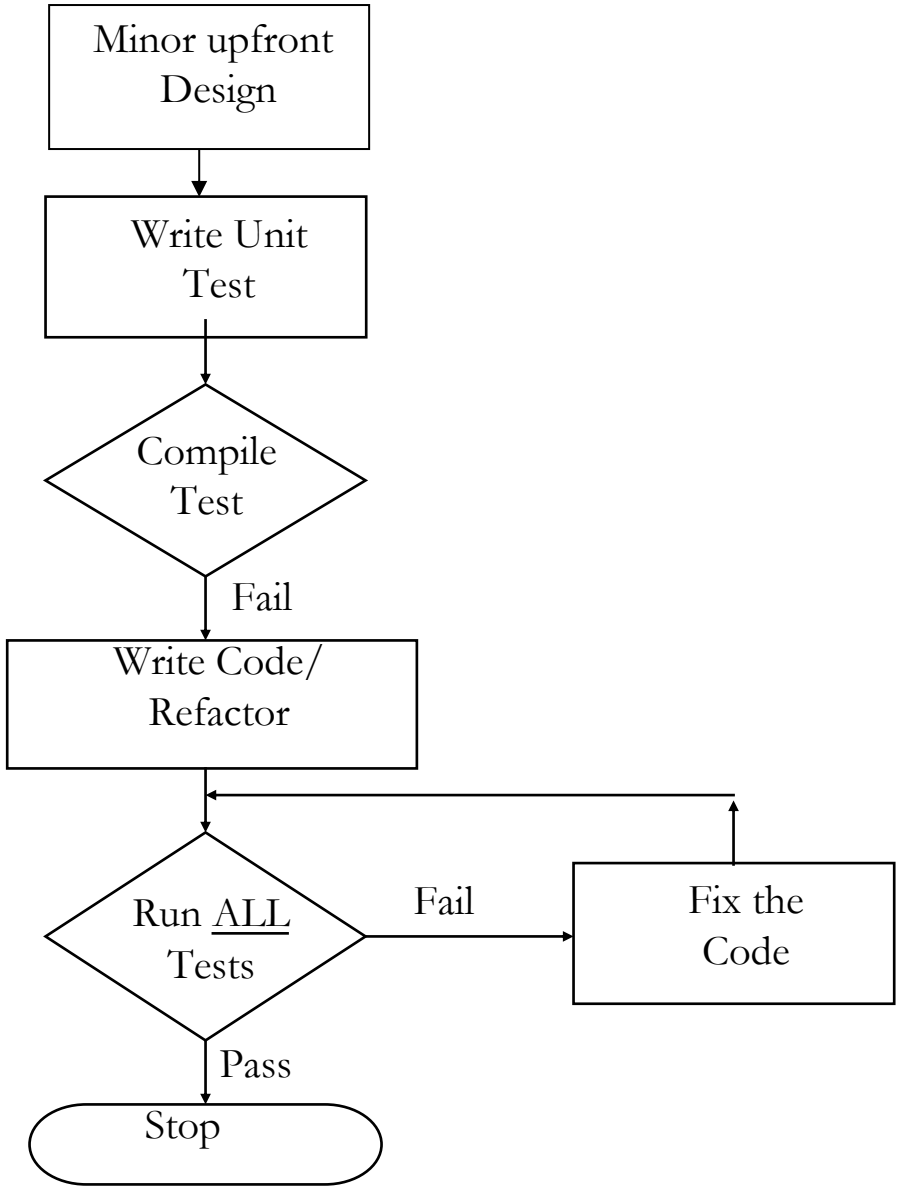
- "State of the art" → test-last
- "State of the practice" → test-whenever needed
- TDD → test-first

- Design evolves through coding/feedback

- Write unit tests for every piece of code that could possibly break

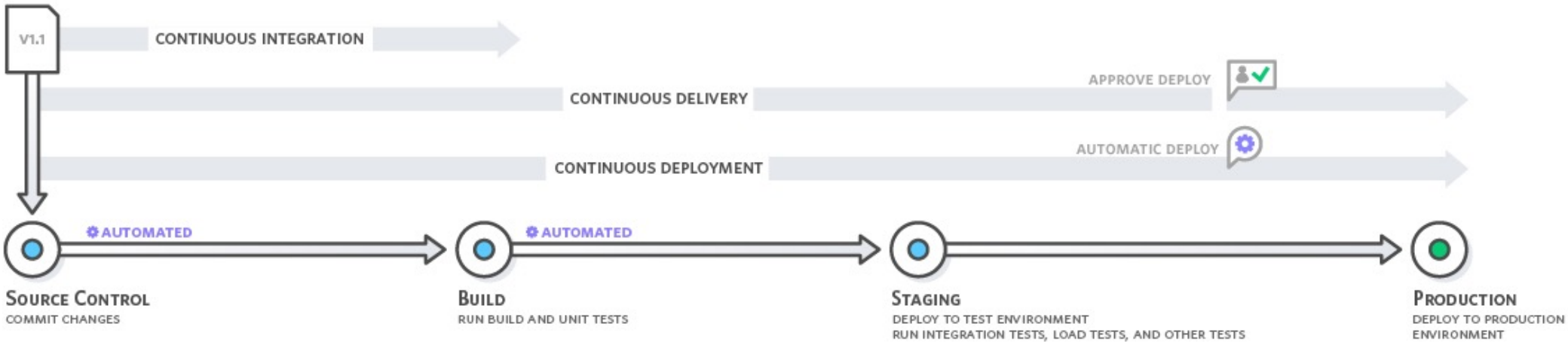
- Preferred testing tool are xUnits (open source)

TDD Explained



Continuous Integration

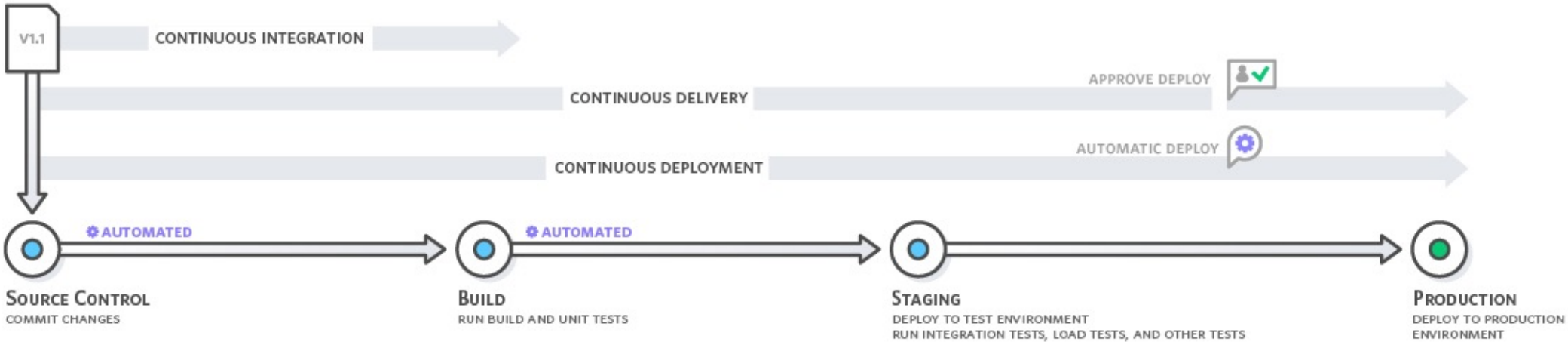
- Developers merge their code into the central repository regularly
- Automated builds and testing



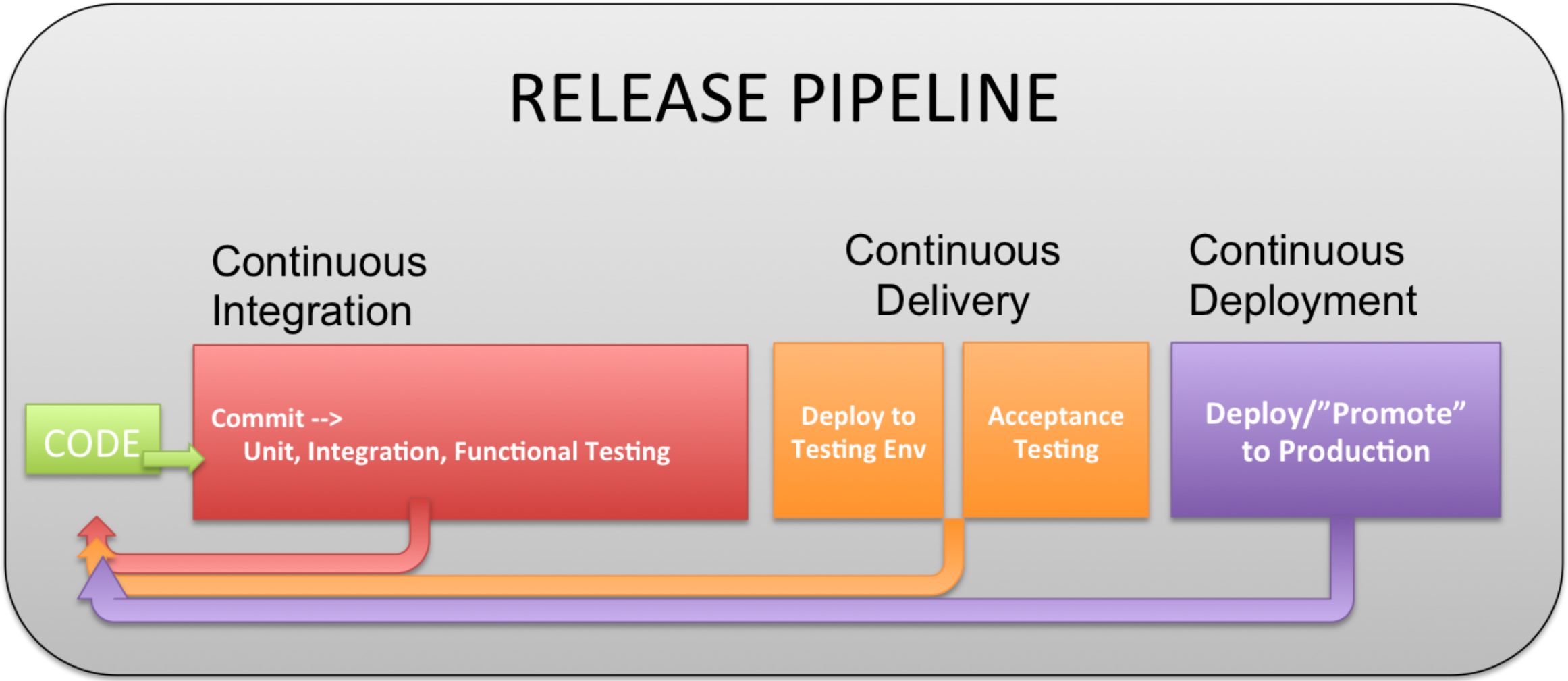
Imagesource:aws.amazon.cor

Continuous Delivery

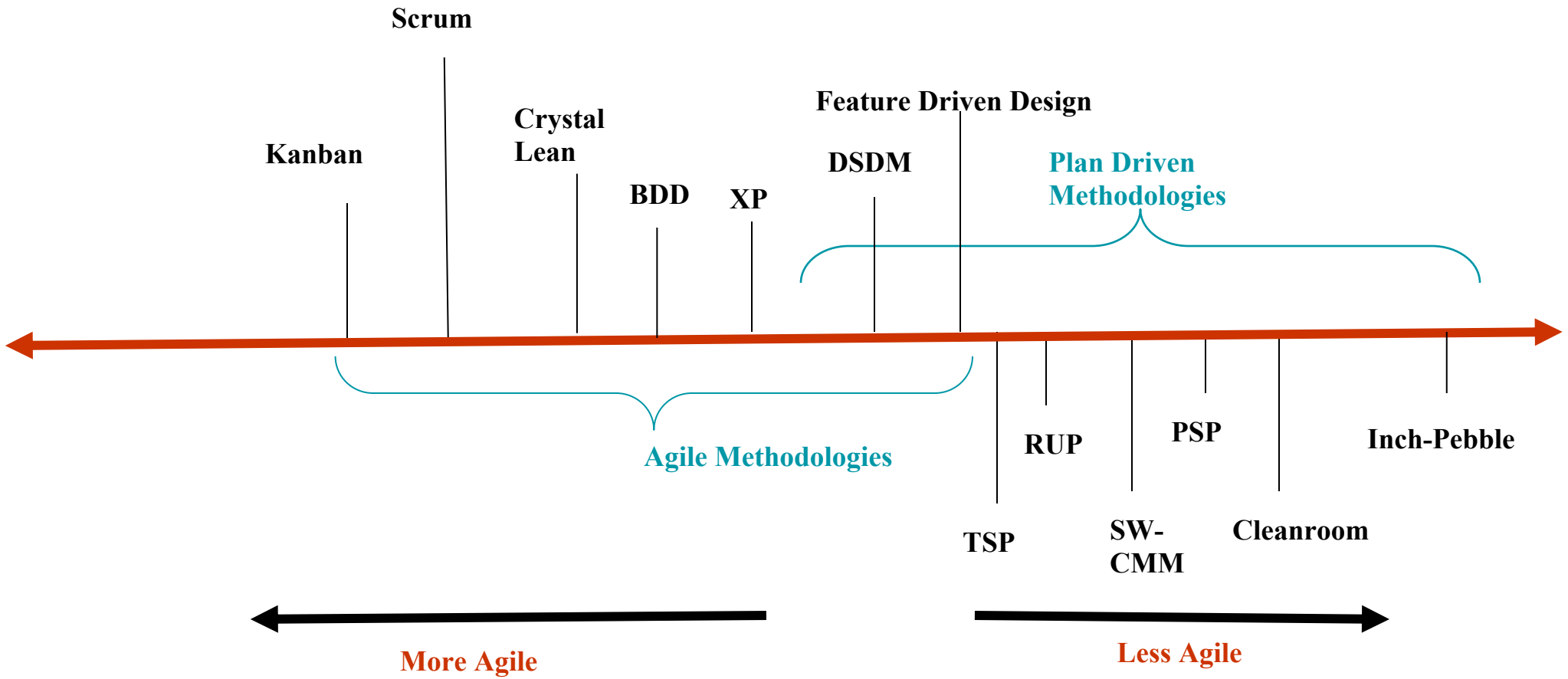
- Deploys all code changes to a testing and/or production environment after the build stage
- Deployment is manual



Imagesource:aws.amazon.cor


RELEASE PIPELINE
Continuous Integration
Continuous Delivery
Continuous Deployment
CODE
Commit --> Unit, Integration, Functional Testing
Deploy to Testing Env
Acceptance Testing
Deploy/"Promote" to Production

The Process Methodology Spectrum



It's not that black and white. The process spectrum spans a range of grey !

Development problems addressed – What about Release problems ?

- Database issues
- OS issues
- Too slow in real settings
- Infrastructure issues
- Source from many repositories
- Different versions (libraries, compilers, local utilities, etc)
- Missing dependencies
- …

Developers

- Designing
- Coding
- Testing, bug tracking, reviews
- Continuous Integration
- ...

Operations

Managing/Allocating hardware/OS updates/resources, database

Monitoring load spikes, performance, crashes hardware updates

Backups, Rollback releases

etc.

- ✓ Can there be better coordination between Developers and Operators?
- ✓ Reduce issues while moving changes from development to production
- ✓ Configurations as code
- ✓ Automation (Delivery and Monitoring)

DevOps

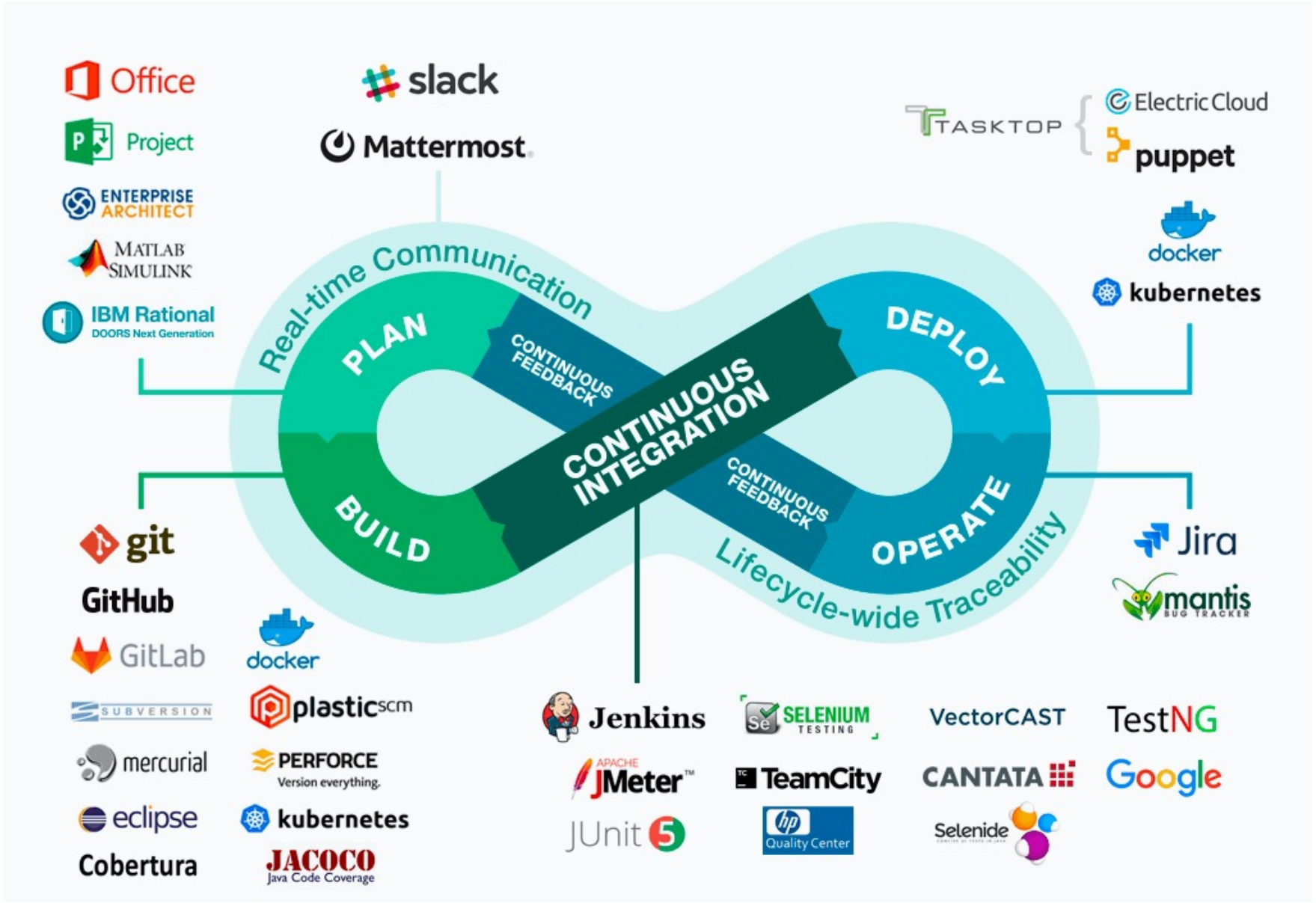
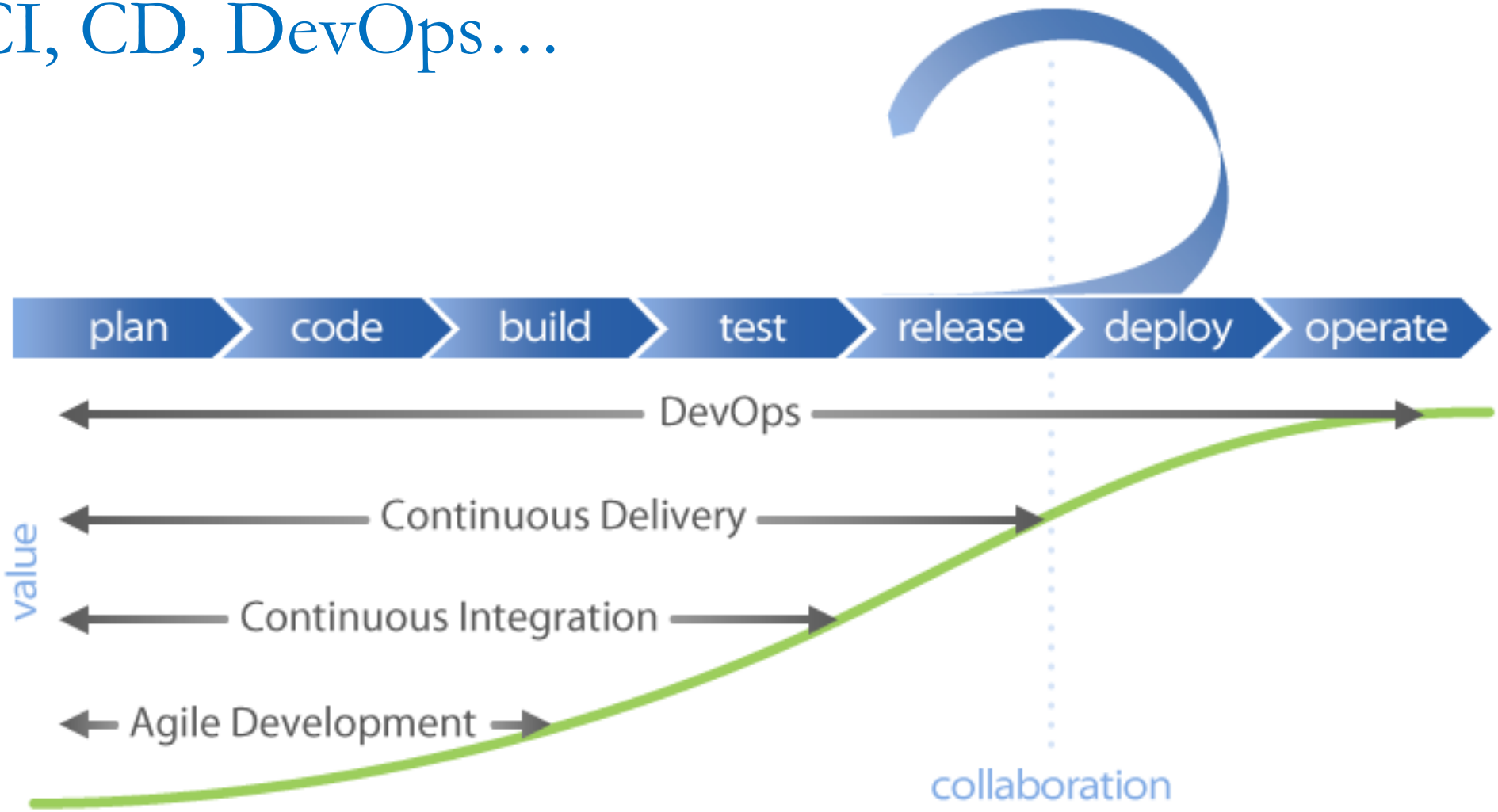


Image Source: https://intland.com/wp-content/uploads/2019/07/devops-infinity-1-1.png

DevOps – Common practices

- Continuous Integration
- Continuous Delivery
- Infrastructure as code, test and deploy in containers
- Monitoring and logging
- Microservice architecture
- Communicate and Collaborate

Agile, CI, CD, DevOps…



image@CollabNet