Software Development Life Cycle Models - (Process Models)

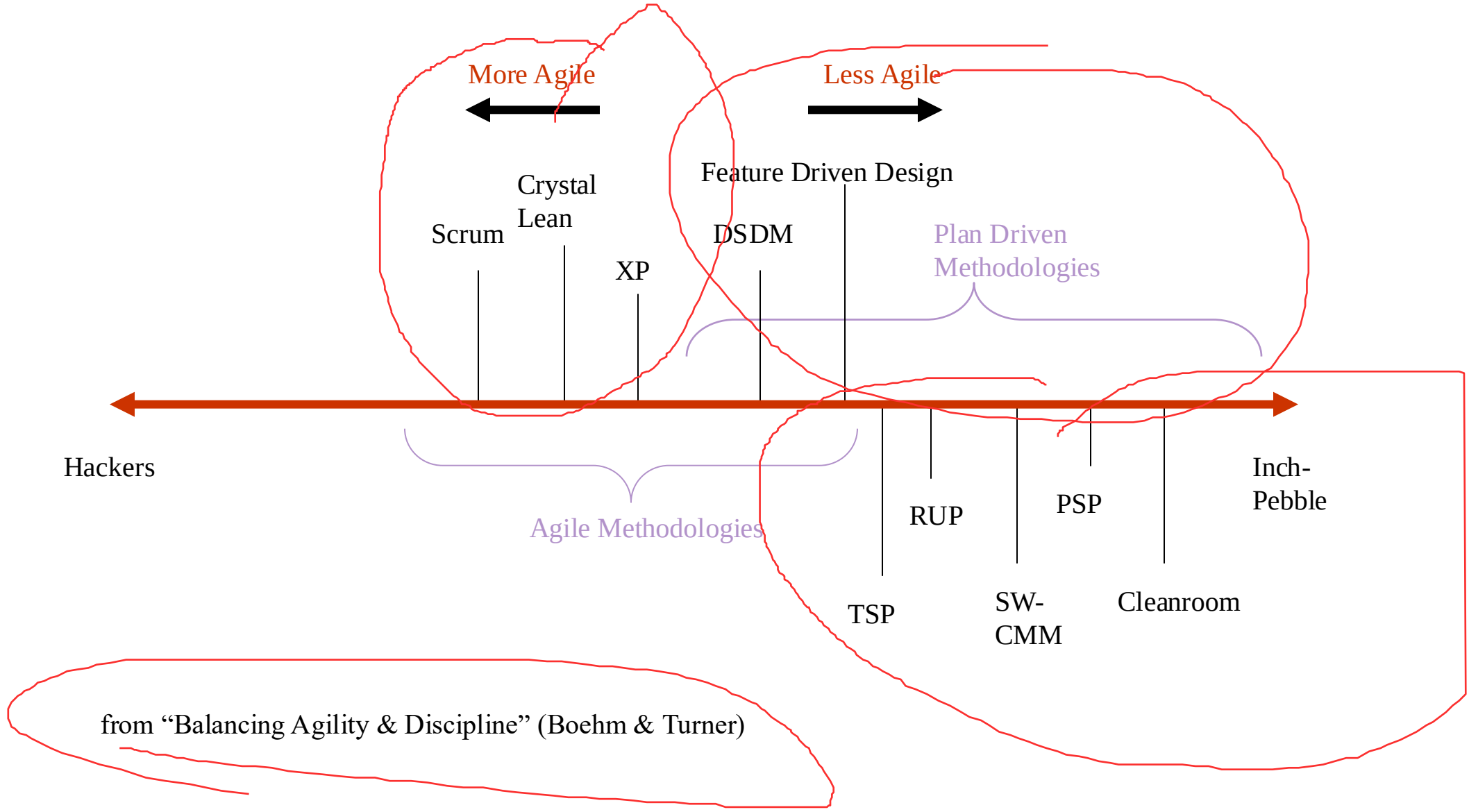
Week 2

PROCESS MODELS

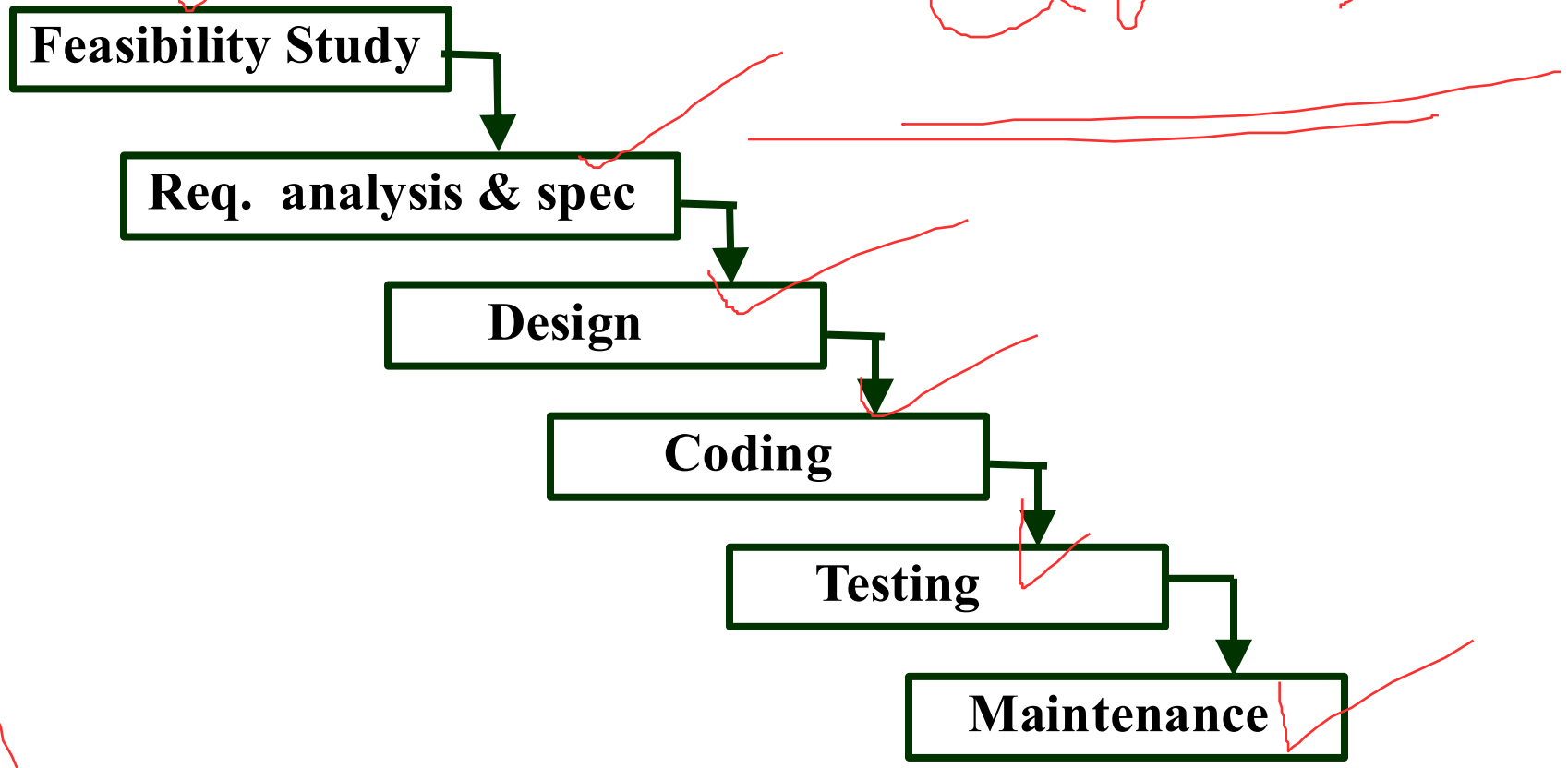
Many life cycle models have been proposed

- Traditional Models (plan-driven)
  - Classical waterfall model
  - Iterative waterfall
  - Evolutionary
  - Prototyping
  - Spiral model
  - Rational Unified Process (RUP)
- Agile Models
  - eXtreme Programming (XP)
  - Scrum
  - Crystal
  - Feature-Driven Development (FDD)

The Process Methodology Spectrum



CLASSICAL WATERFALL MODEL



CLASSICAL WATERFALL MODEL (CONT.)

- The guidelines and methodologies of an organization:
  - called the organization's software development methodology.
- Software development organizations:
  - expect fresh engineers to master the organization's software development methodology.

Problems with Classical Waterfall Model

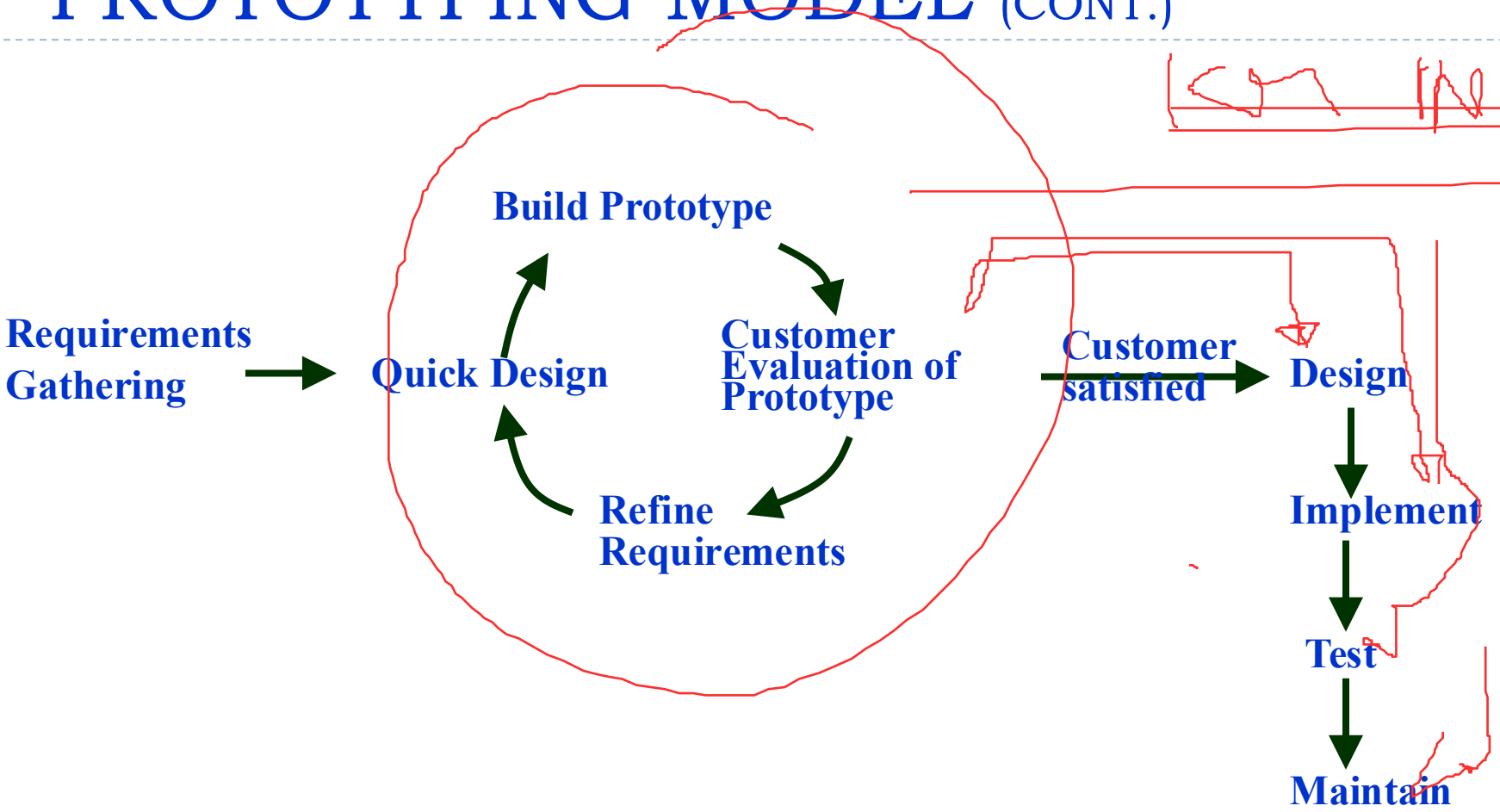
- Classical waterfall model is idealistic:
  - assumes that no defect is introduced during any development activity.
  - in practice:
    - defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
  - For example, a design defect might go unnoticed till the coding or testing phase

PROTOTYPING MODEL

- Before starting actual development,
  - a working prototype of the system should first be built.
- A prototype is a toy implementation of a system:
  - limited functional capabilities,
  - low reliability,
  - inefficient performance.

WHY PROTOTYPE?

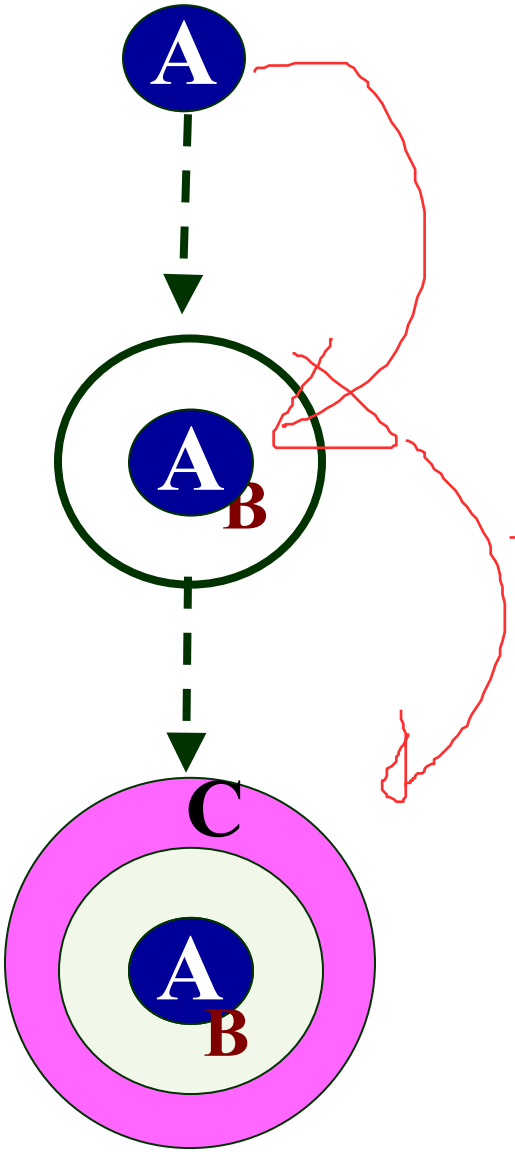
PROTOTYPING MODEL (CONT.)

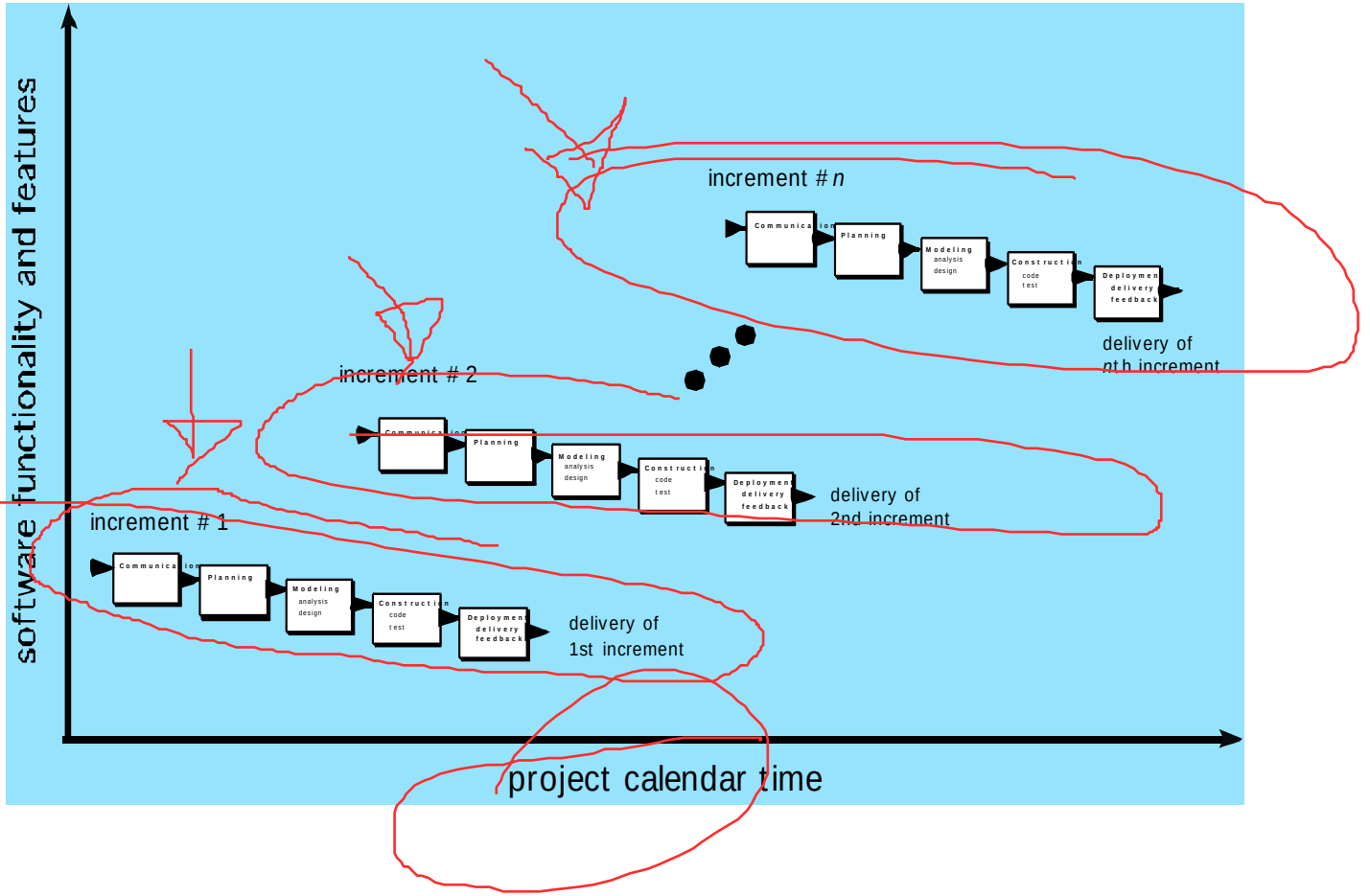


EVOLUTIONARY MODEL

- Evolutionary model (aka successive versions or incremental model):
  - The system is broken down into several modules which can be incrementally implemented and delivered.
- First develop the core modules of the system.
- The initial product skeleton is refined into increasing levels of capability:
  - by adding new functionalities in successive versions.

INCREMENTAL MODEL





ADVANTAGES OF EVOLUTIONARY MODEL

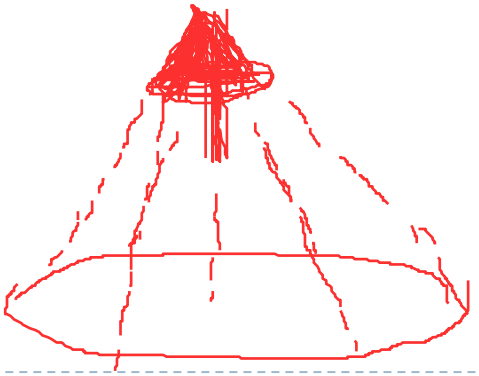
- Users get a chance to experiment with a partially developed system:
  - much before the full working version is released,
- Helps finding exact user requirements:
  - much before fully working system is developed.
- Core modules get tested thoroughly:
  - reduces chances of errors in final product.

DISADVANTAGES OF EVOLUTIONARY MODEL

- Often, difficult to subdivide problems into functional units:
  - which can be incrementally implemented and delivered.
  - evolutionary model is useful for very large problems,
    - where it is easier to find modules for incremental implementation.

EVOLUTIONARY MODEL WITH ITERATION (Iterative Incremental Model)

- Many organizations use a combination of iterative and incremental development:
  - a new release may include new functionality
  - existing functionality from the current release may also have been modified.



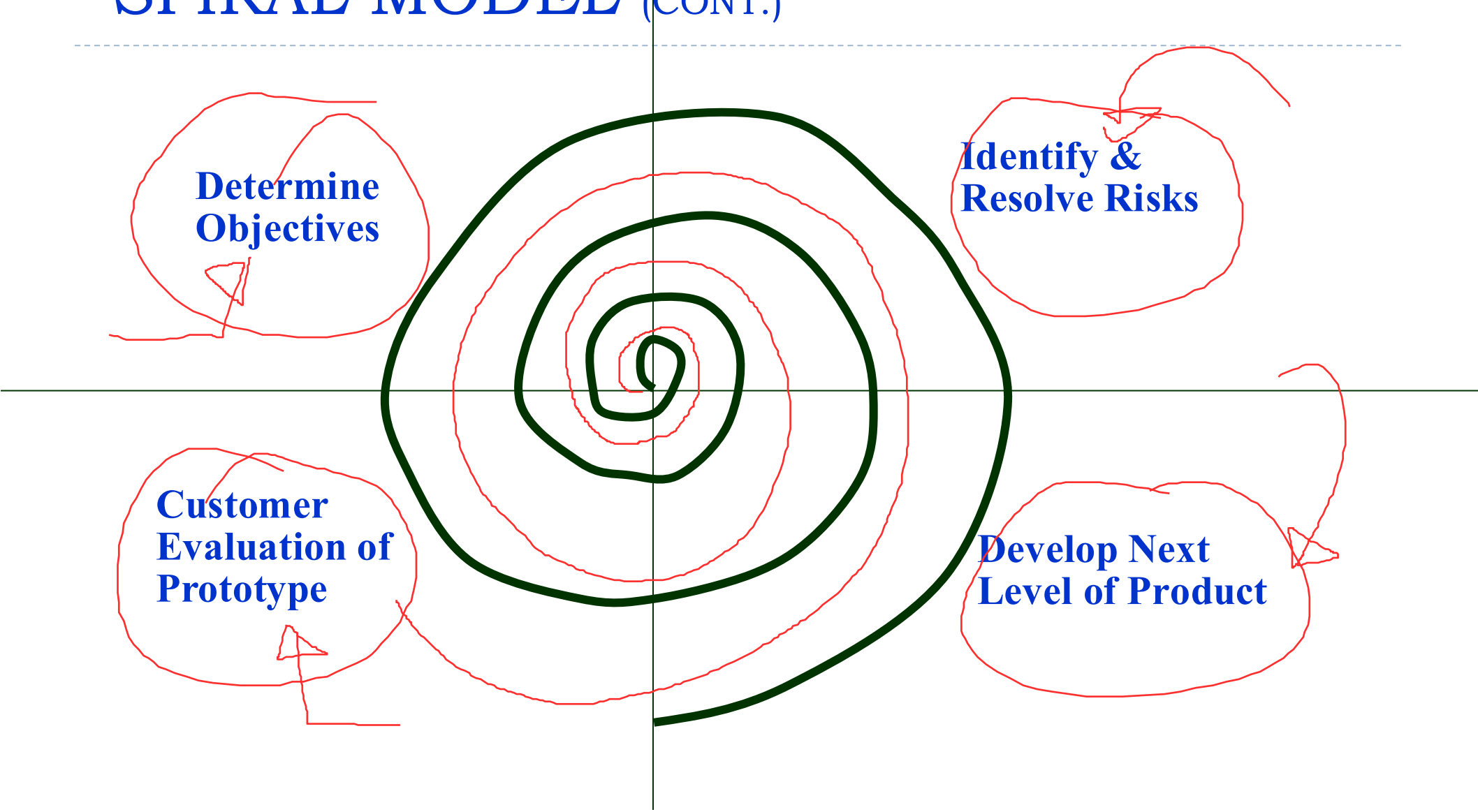
EVOLUTIONARY MODEL WITH ITERATION

- Several advantages:
  - Training can start on an earlier release
    - customer feedback taken into account
  - Markets can be created:
    - for functionality that has never been offered.
  - Frequent releases allow developers to fix unanticipated problems quickly.

SPIRAL MODEL

- Proposed by Boehm in 1988.
- Each loop of the spiral represents a phase of the software process:
  - the innermost loop might be concerned with system feasibility,
  - the next loop with system requirements definition,
  - the next one with system design, and so on.
- There are no fixed phases in this model, the phases shown in the figure are just examples.

SPIRAL MODEL (CONT.)



OBJECTIVE SETTING (FIRST QUADRANT)

- Identify objectives of the phase,
- Examine the risks associated with these objectives.
  - Risk:
    - any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

RISK ASSESSMENT AND REDUCTION (SECOND QUADRANT)

- For each identified project risk,
  - a detailed analysis is carried out.
- Steps are taken to reduce the risk.
- For example, if there is a risk that the requirements are inappropriate:
  - a prototype system may be developed.

SPIRAL MODEL (CONT.)

- Development and Validation (Third quadrant):
  - develop and validate the next level of the product.
- Review and Planning (Fourth quadrant):
  - review the results achieved so far with the customer and plan the next iteration around the spiral.
- With each iteration around the spiral:
  - progressively more complete version of the software gets built.

SPIRAL MODEL AS A META MODEL

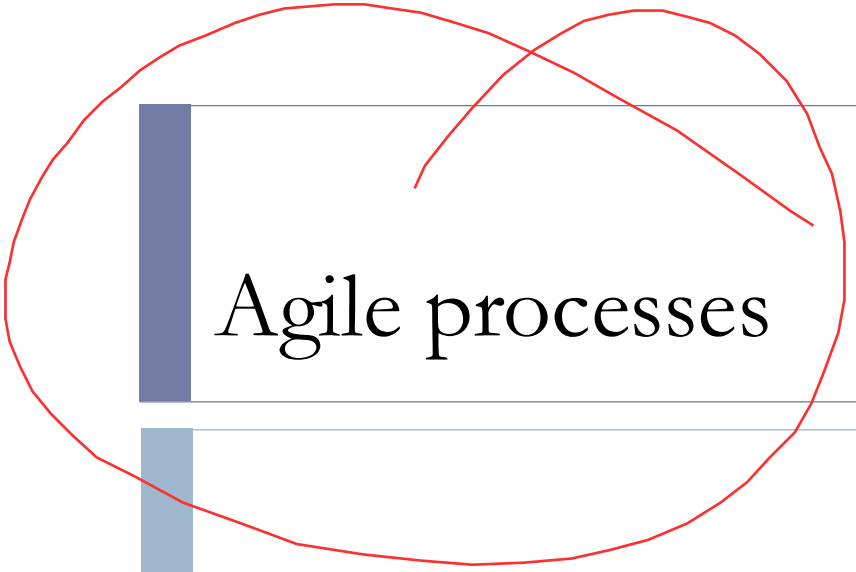
- Subsumes all discussed models:
  - a single loop spiral represents waterfall model.
  - uses an evolutionary approach --
    - iterations through the spiral are evolutionary levels.
  - enables understanding and reacting to risks during each iteration along the spiral.
  - uses:
    - prototyping as a risk reduction mechanism
    - retains the step-wise approach of the waterfall model.

COMPARISON OF DIFFERENT LIFE CYCLE MODELS

- Iterative waterfall model
  - most widely used model.
  - But, suitable only for well-understood problems.
- Prototype model is suitable for projects not well understood:
  - user requirements
  - technical aspects

COMPARISON OF DIFFERENT LIFE CYCLE MODELS (CONT.)

- Evolutionary model is suitable for large problems:
  - can be decomposed into a set of modules that can be incrementally implemented,
  - incremental delivery of the system is acceptable to the customer.
- The spiral model:
  - suitable for development of technically challenging software products that are subject to several kinds of risks.



Agile processes

Agile Manifesto

- We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
  - **Individuals and interactions** over processes and tools
  - **Working software** over comprehensive documentation
  - **Customer collaboration** over contract negotiation
  - **Responding to change** over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

Some Agile Methods

- ASD - Adaptive Software Development
- Crystal
- FDD - Feature Driven Development
- DSDM - Dynamic Systems Development Method
- Lean Software Development
- Scrum
- XP - eXtreme Programming

Four Values

- Simplicity
  - create the simplest thing that could work
- Communication
  - face-to-face, not document-to-face
- Feedback
  - lots of tests
- Aggressiveness

Four Basic Activities

- Coding
  - cannot do without it
- Testing
  - if it cannot be tested it doesn't exist
- Listening
  - to those with domain knowledge
- Designing
  - to keep the system from decaying

Twelve Practices

1. The Planning Game
2. Small releases
3. Metaphor
4. Simple design
5. Testing
6. Refactoring
7. Pair programming
8. Collective ownership
9. Continuous integration
10. 40-hour week
11. On-site customer
12. Coding standards

SUMMARY

- There are various process models:
  - Traditional (Plan-driven) Models
  - Agile Models

Helps to do various development activities in a systematic and disciplined manner.

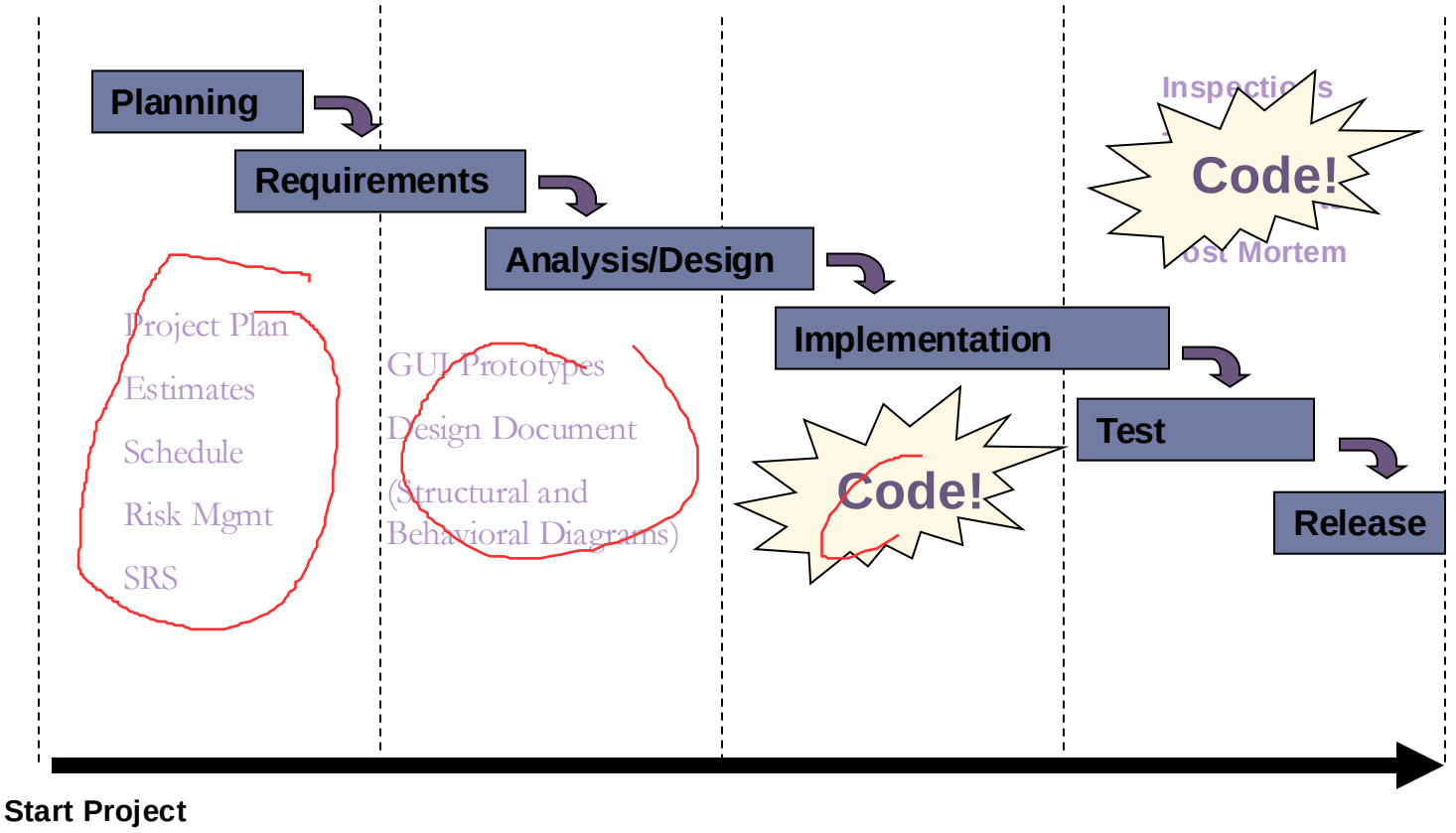
Also makes it easier to manage a software development effort.

In-class Activity 1 – Process Activity

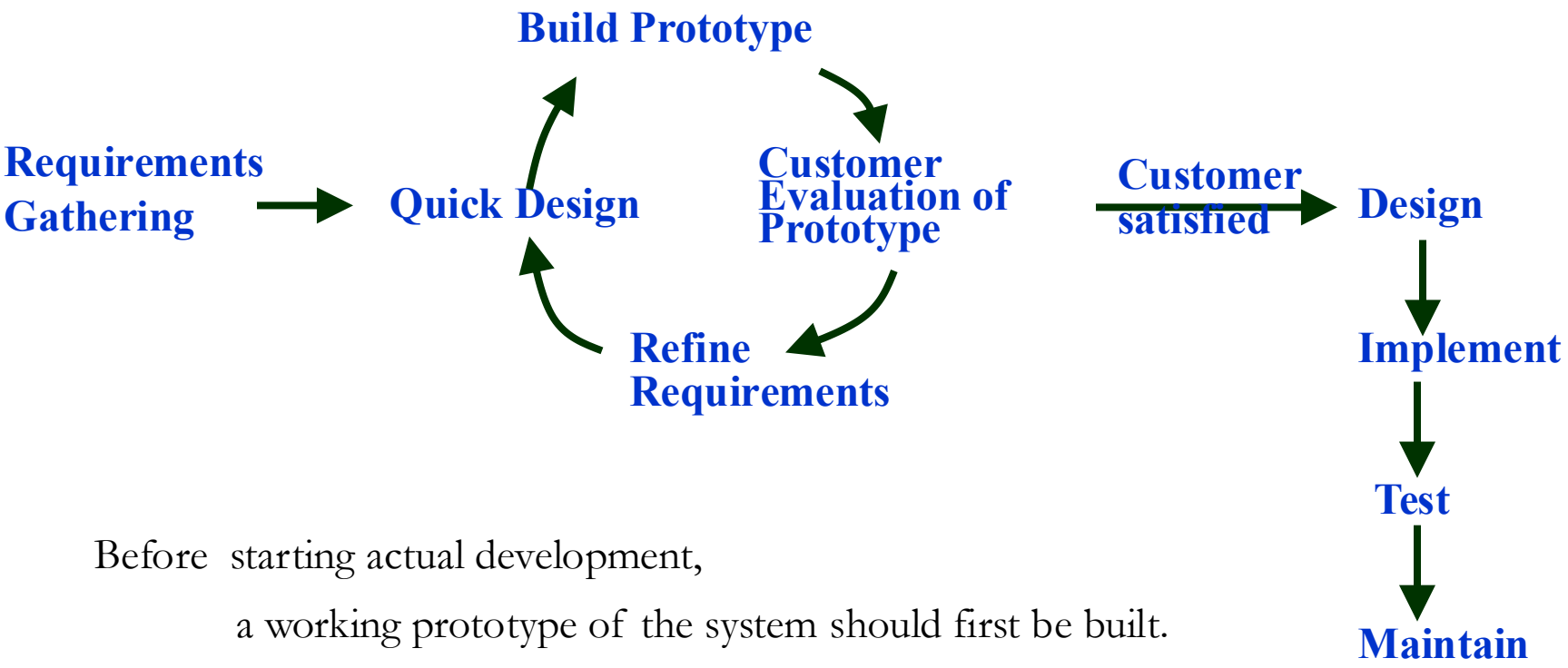
- You are working in a team of 5 people for 3 months, to build a utility that converts from one standard graphics file format to another. (Since these are standard formats, the requirements are fully defined upfront). What lifecycle model would you use for this, and why?

- Identify 3-4 reasons why the incremental approach is better for large commercial projects than the simple waterfall approach.

Traditional SDLC. (e.g. waterfall process)



PROTOTYPING MODEL



Before starting actual development,

a working prototype of the system should first be built.

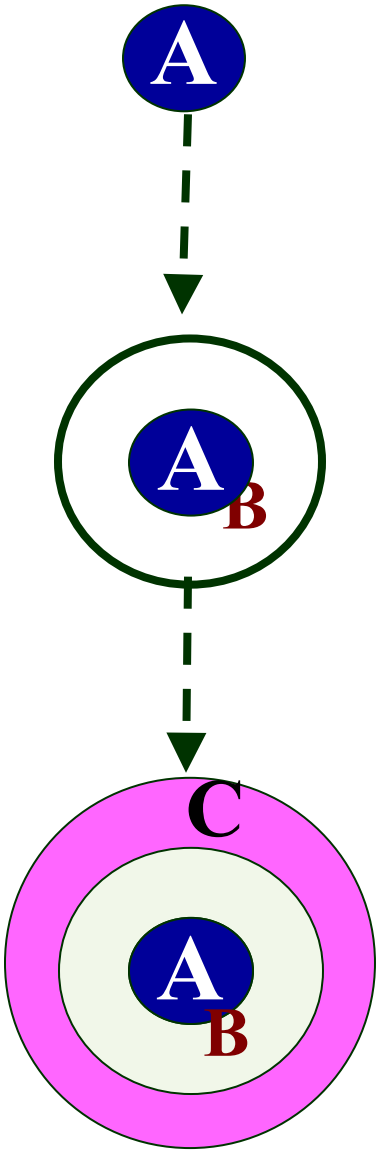
A prototype is a toy implementation of a system:

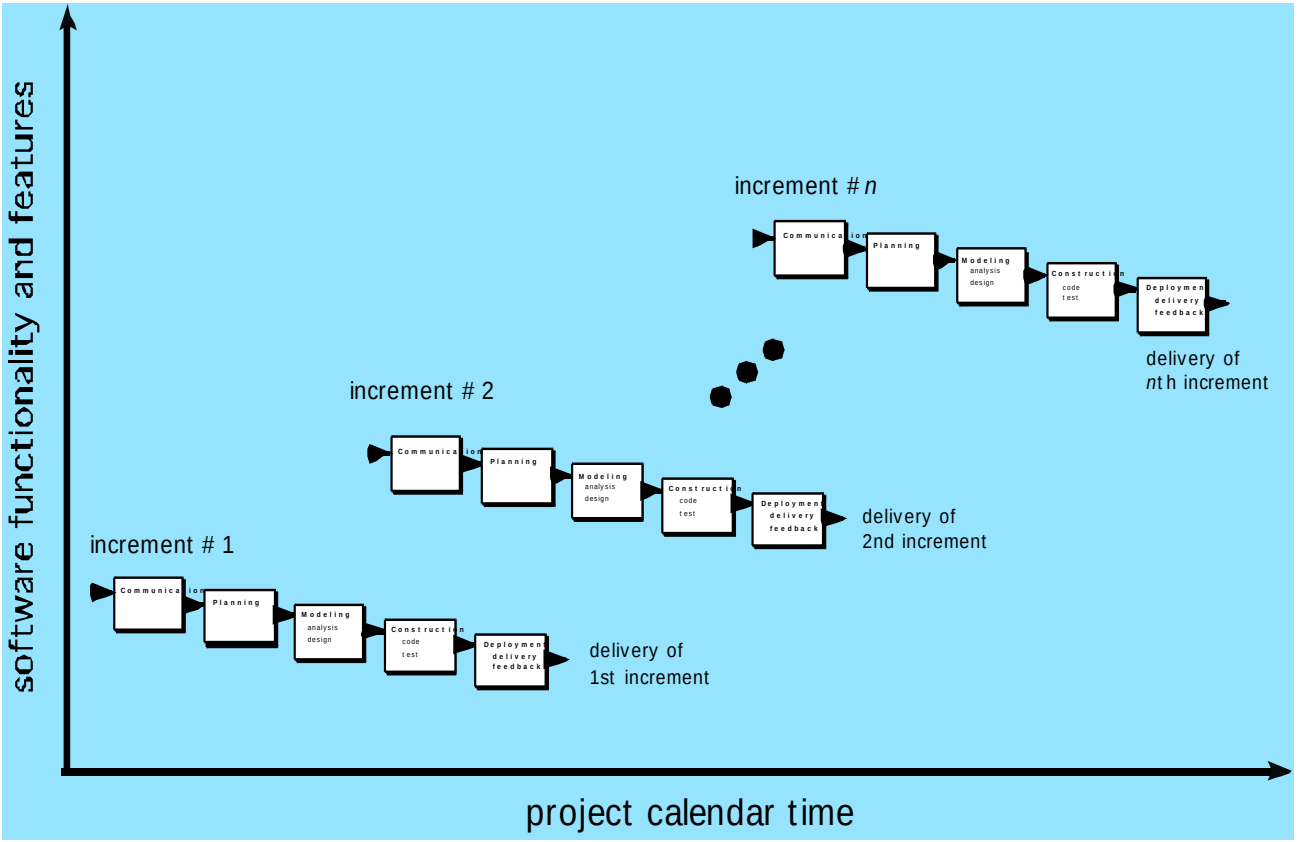
limited functional capabilities,

low reliability,

inefficient performance.

INCREMENTAL MODEL



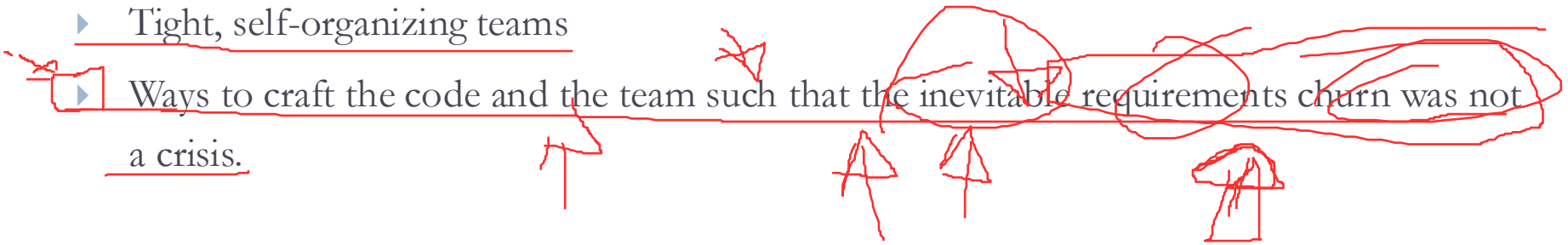


Challenges with traditional (plan-driven) approaches

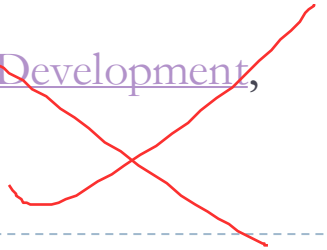
- Lightweight applications/heavyweight process
- Document intensive (perceived)
- Less flexible design
- Big bang approach to coding/integration
- Testing short-shifted
- One-shot delivery opportunity
- Limited opportunity for process improvement

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,



Agile Characteristics

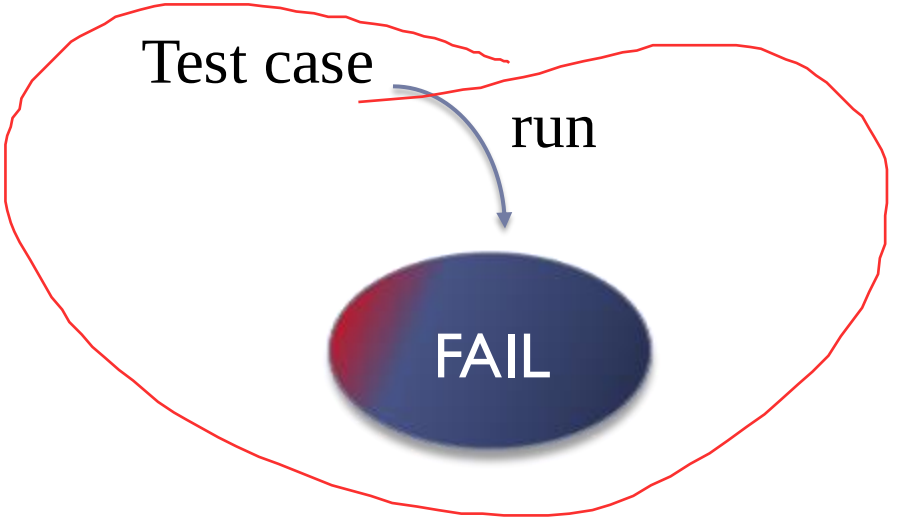
- Incremental development – several releases
- Planning based on user stories
- Each iteration touches all life-cycle activities
- Testing – unit testing for deliverables; acceptance tests for each release
- Flexible Design – evolution vs. big upfront effort
- Reflection after each release cycle
- Several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

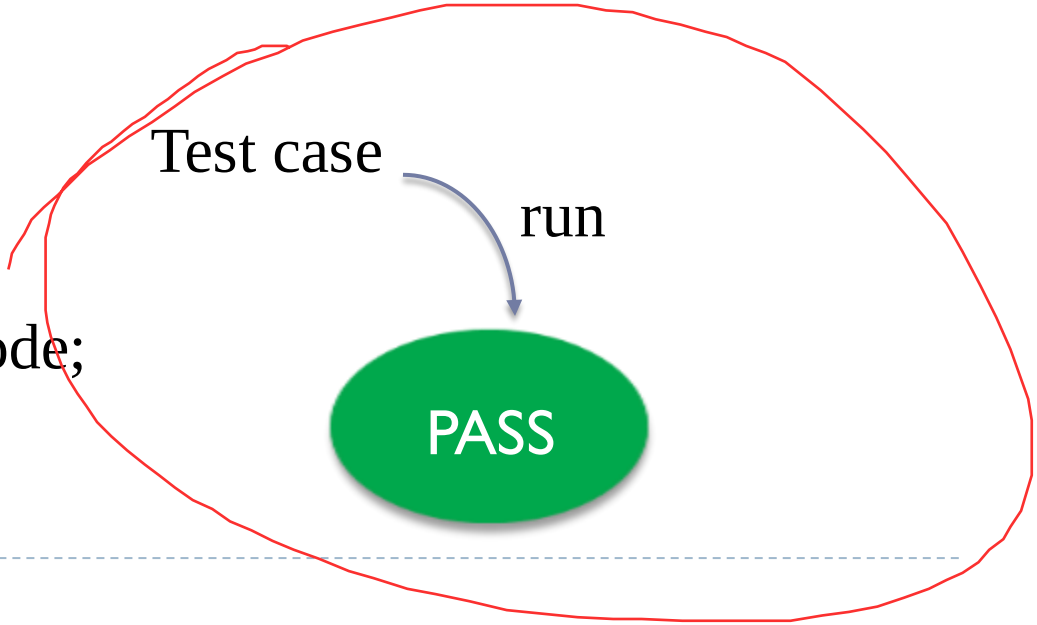
TDD - example

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new
Calculator();
        int result = calculator.add(3, 4);
        assertEquals(7, result);
    }
}
```




```
public class Calculator {

    public int add(int a, int b) {
        return a + b;
    }

}
```

Write Code;



subtraction… (repeat the process)

```
@Test
public void testSubtraction() {

    Calculator calculator = new Calculator();
    int result = calculator.subtract (7, 4);
    assertEquals(3, result);
}
```

Implement the functionality…

```
public class Calculator {

    public int add (int a, int b) {
        return a + b;
    }

    public int subtract (int a, int b) {
        return a - b;
    }

}
```

If you would like to add more functionality… but have common interface

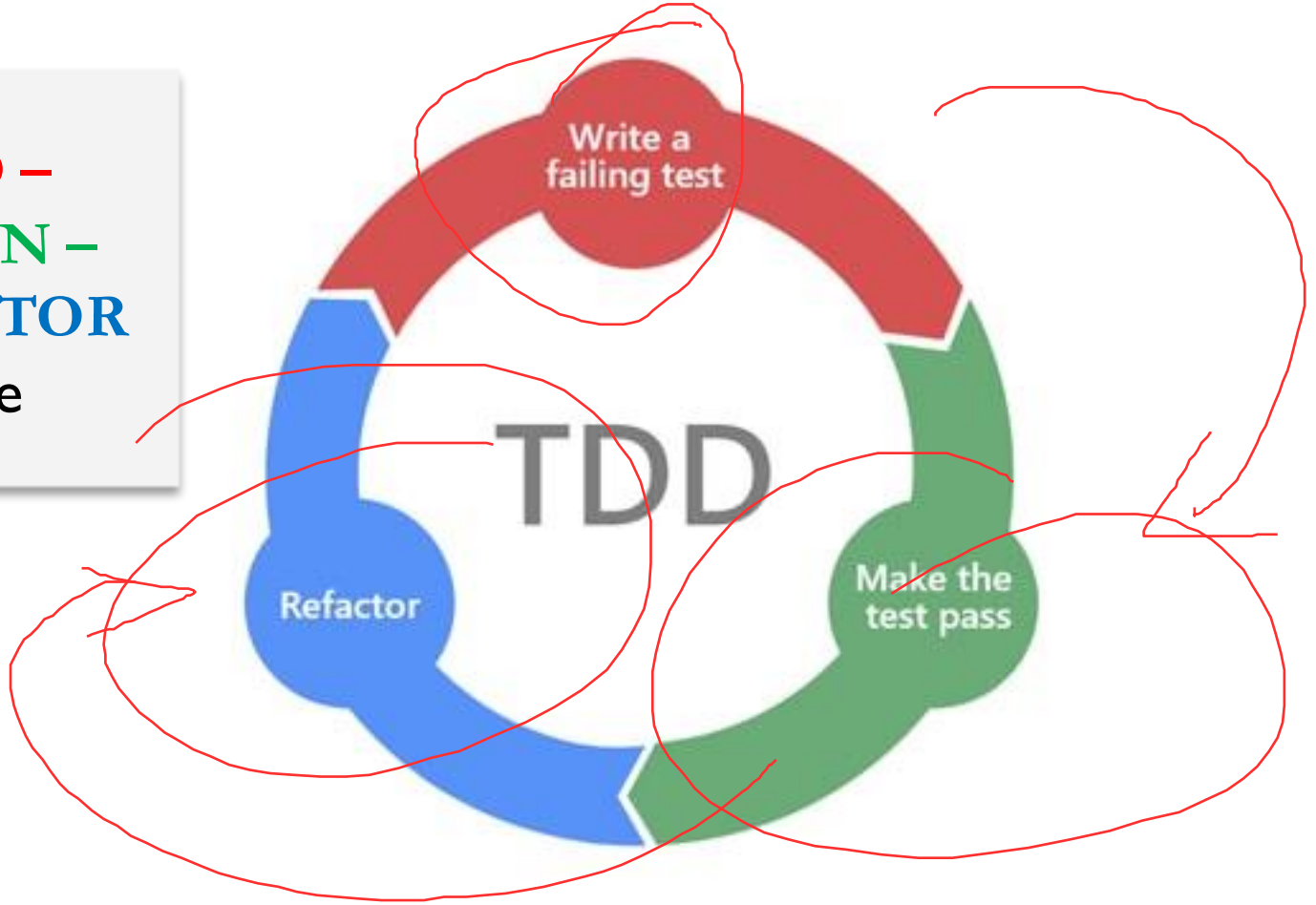
```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class CalculatorTest {

    @Test
    public void testAddition() {
        Calculator calculator = new Calculator(new Addition());
        int result = calculator.calculate (3, 4);
        assertEquals(7, result);
    }

    @Test
    public void testSubtraction() {
        Calculator calculator = new Calculator(new Subtraction());
        int result = calculator.calculate (7, 4);
        assertEquals(3, result);
    }
}
```

REFACTOR the CODE

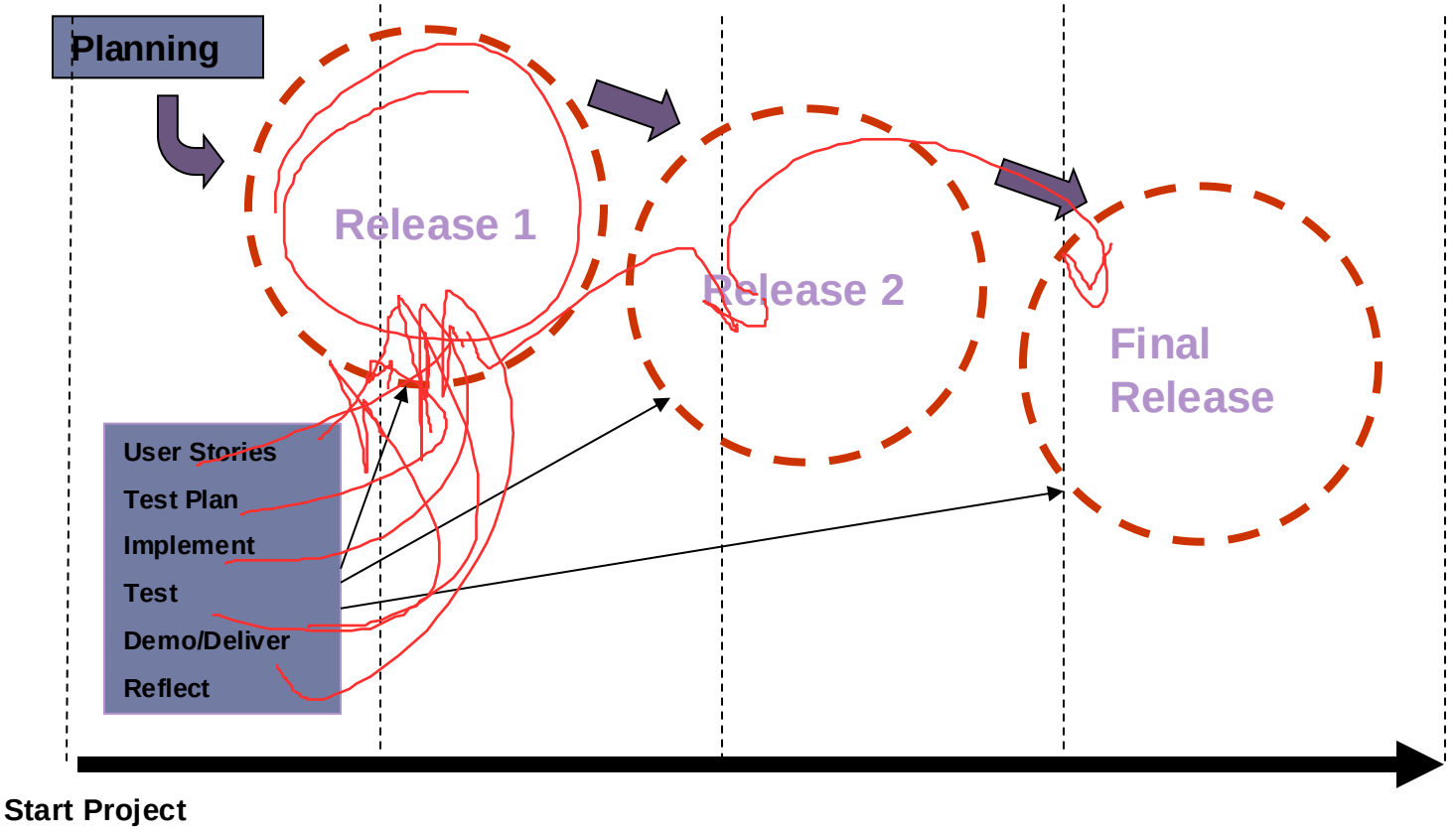
```
public interface Operation {
    int apply (int a, int b);
}

public class Addition implements Operation {
    public int apply (int a, int b) {
        return a + b;
    }
}

public class Subtraction implements Operation {
     public int apply (int a, int b) {
        return a - b;
    }
}

public class Calculator {

    private final Operation operation;

    public Calculator(Operation operation) {
        this.operation = operation;
    }

    public int calculate(int a, int b) {
        return operation.apply (a, b);
    }
}
```

TDD Explained

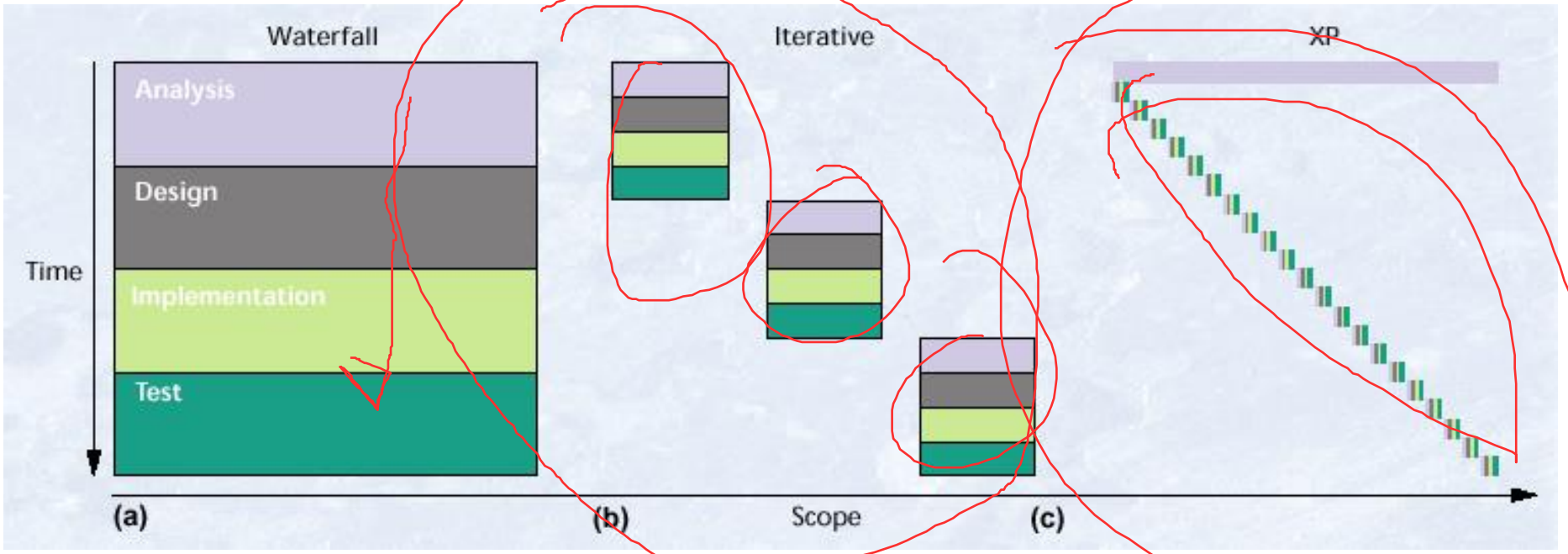
RED –
GREEN –
REFACTOR
cycle



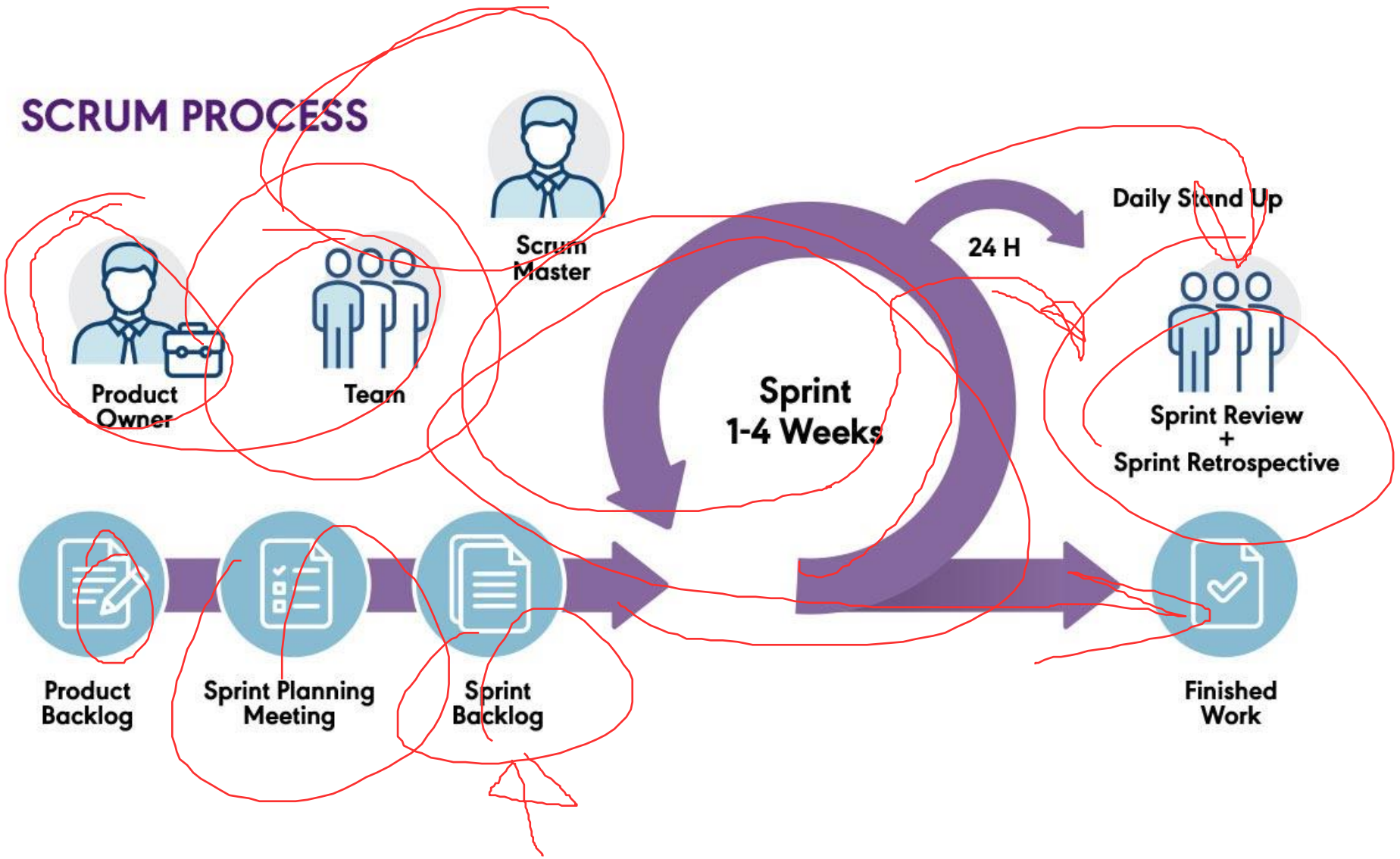
Applying Agility



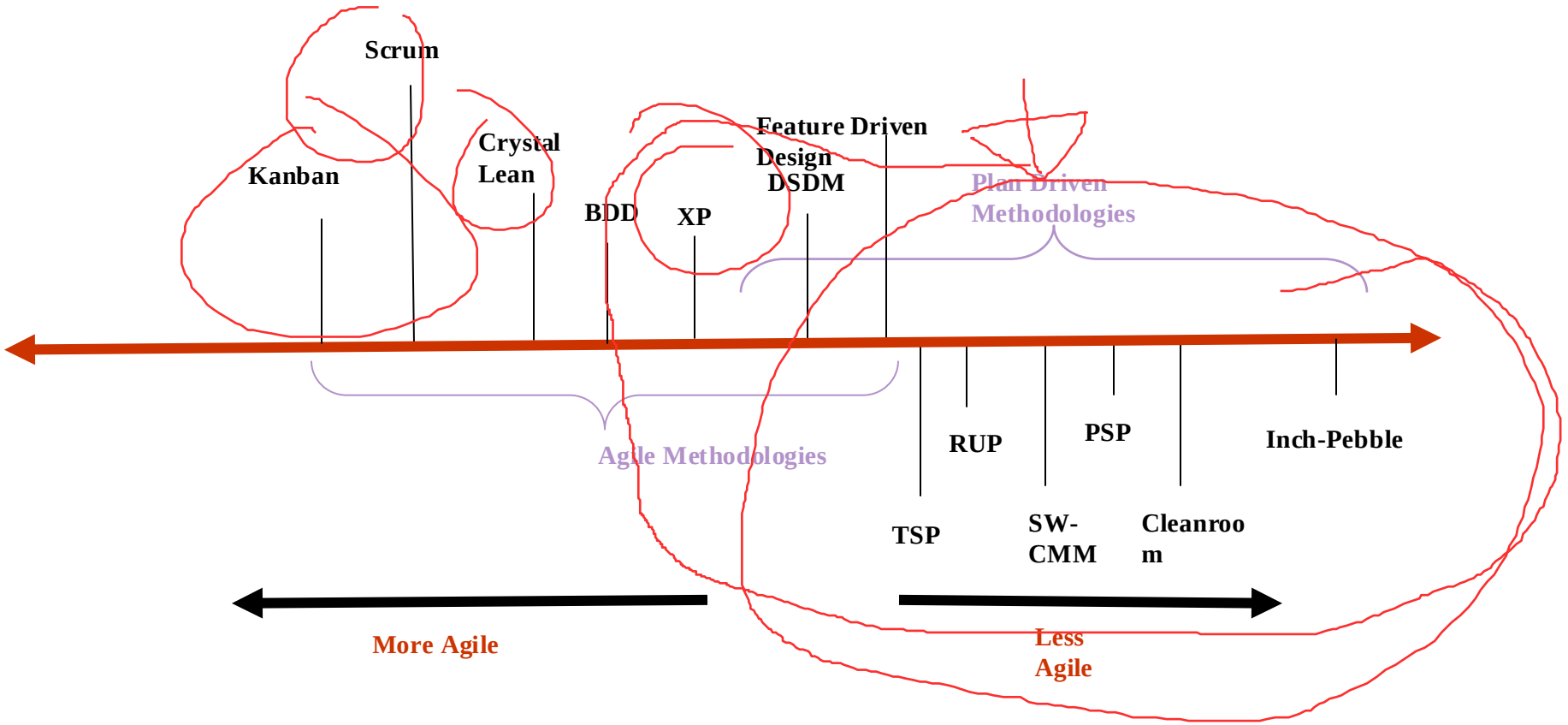
Waterfall to XP Evolution



Source: "Embracing change with extreme programming" by Kent Beck, *IEEE Computer*, October 1999.


SCRUM PROCESS
Scrum Master
Product Owner
Team
Daily Stand Up
24 H
Sprint 1-4 Weeks
Sprint Review + Sprint Retrospective
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Finished Work

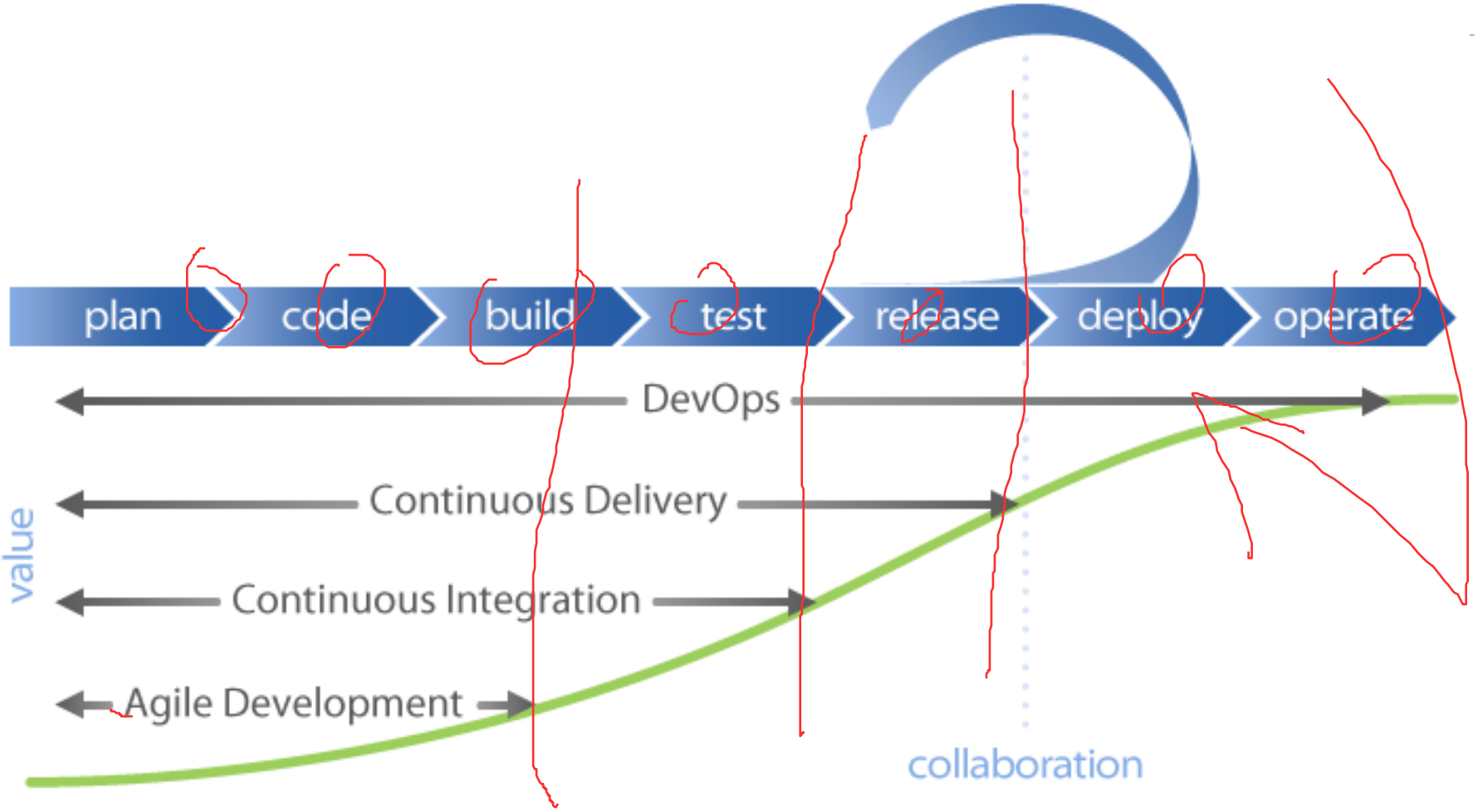
The Process Methodology Spectrum



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

@ (Barry Boehm & Turner)

Agile, CI, CD, DevOps…



image@CollabNet

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

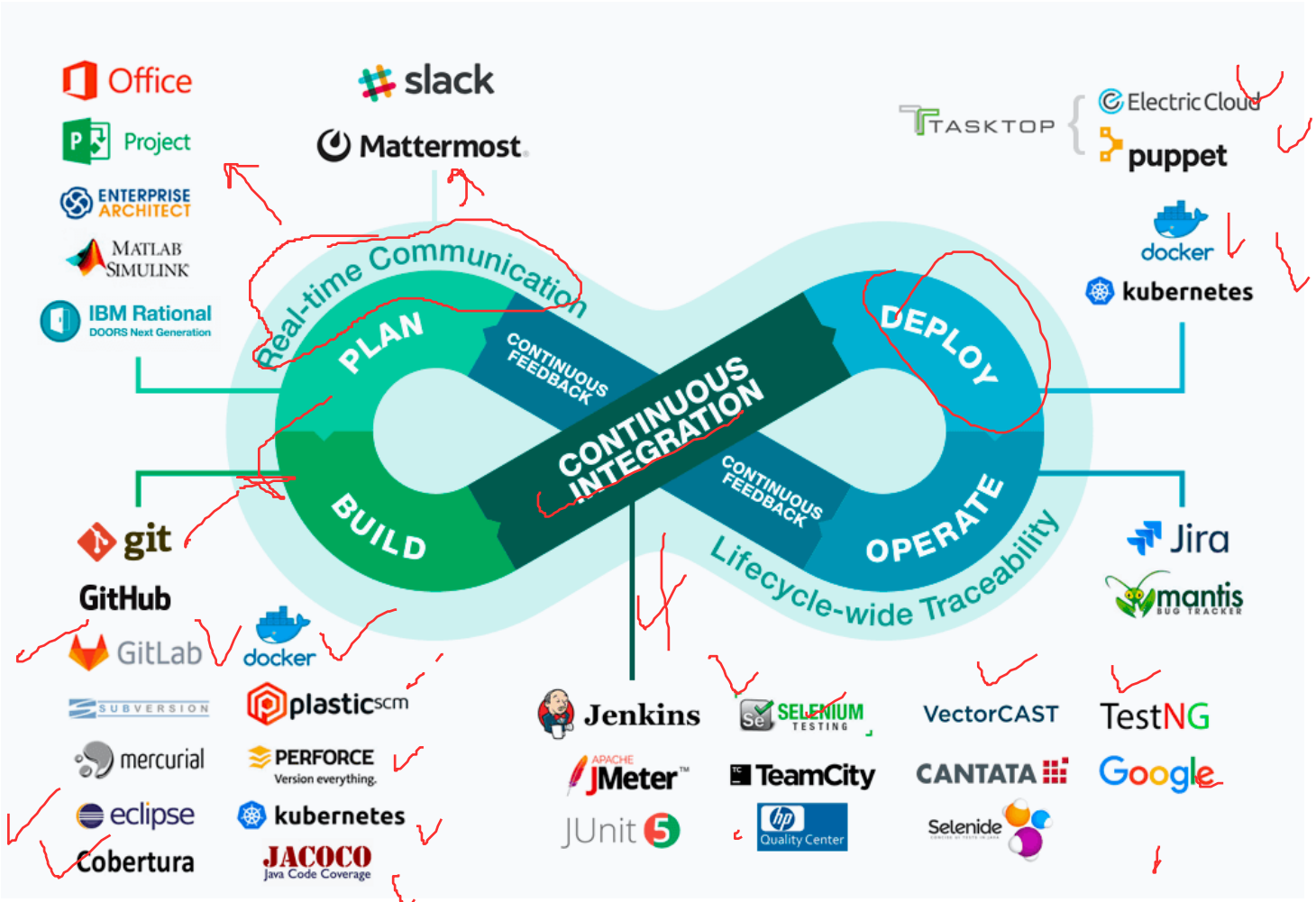


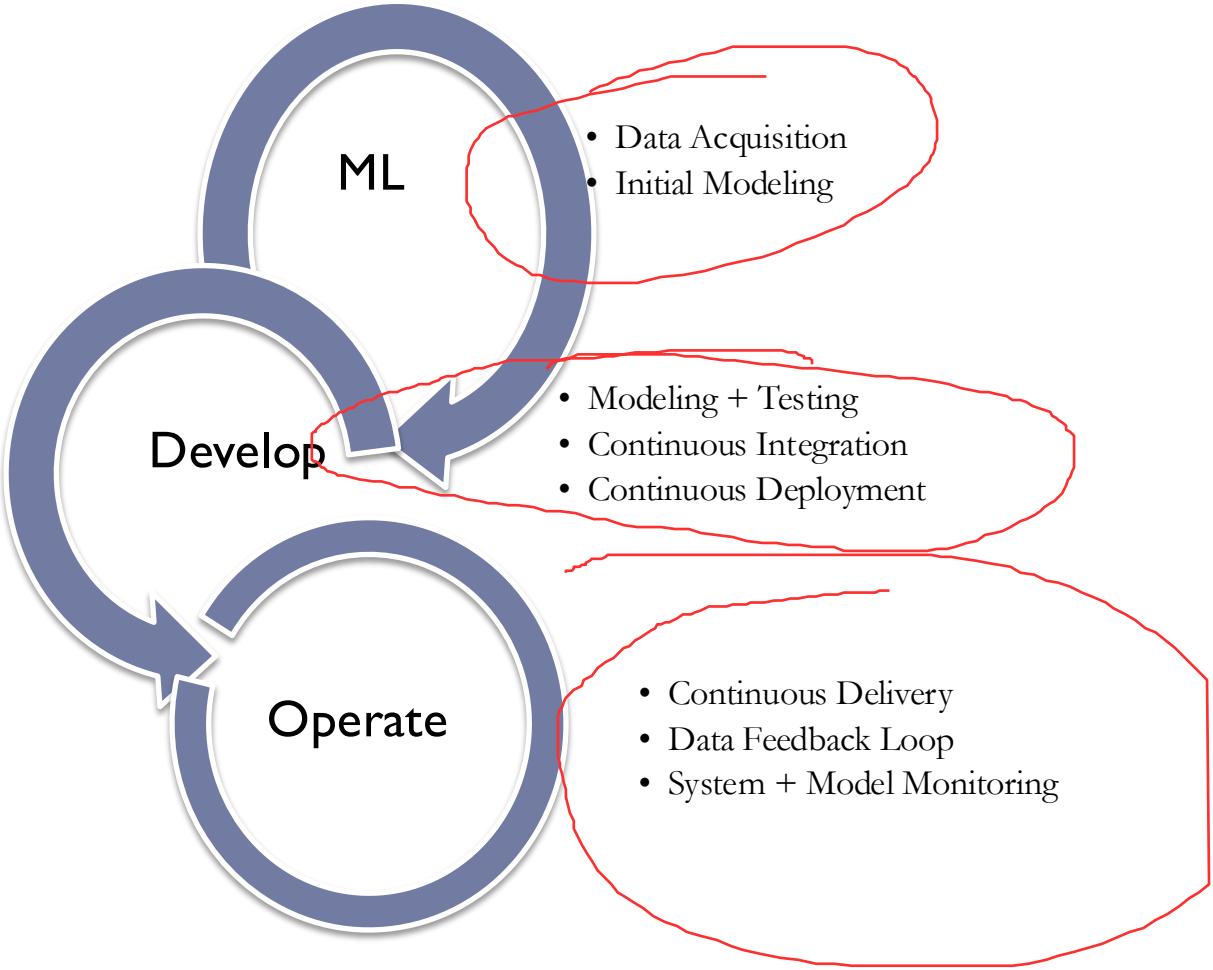
Image Source: https://intland.com/wp-content/uploads/2019/07/devops-infinity-1-1.png

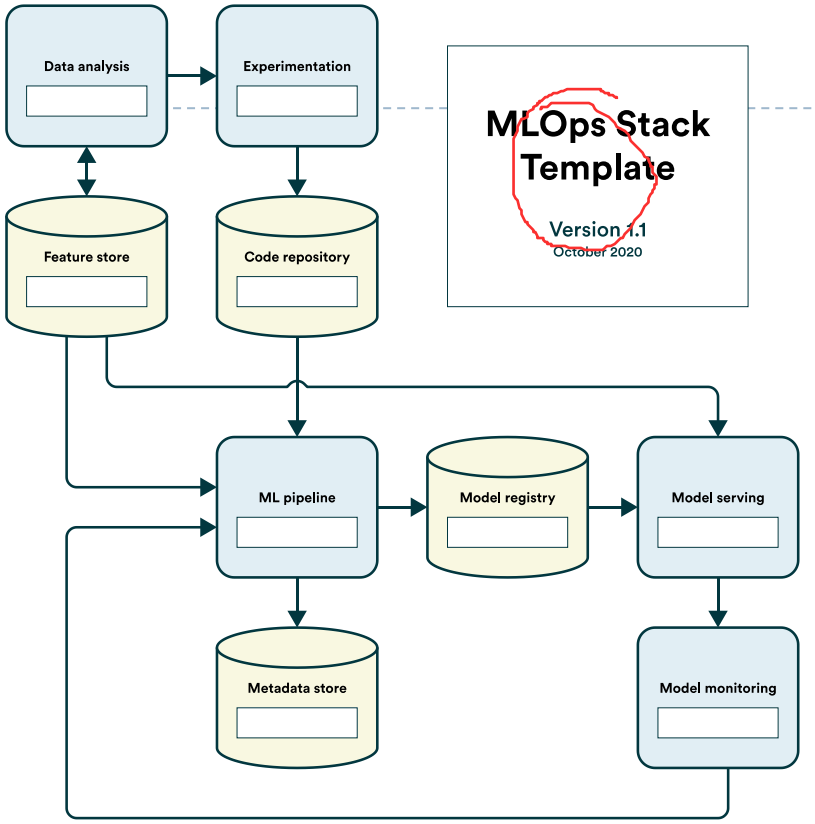
DevOps – Common practices

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

MLOps

- ML + Dev + Ops
- Adopts best practices of DevOps.
- Allows for experimentation (Canary Releases)





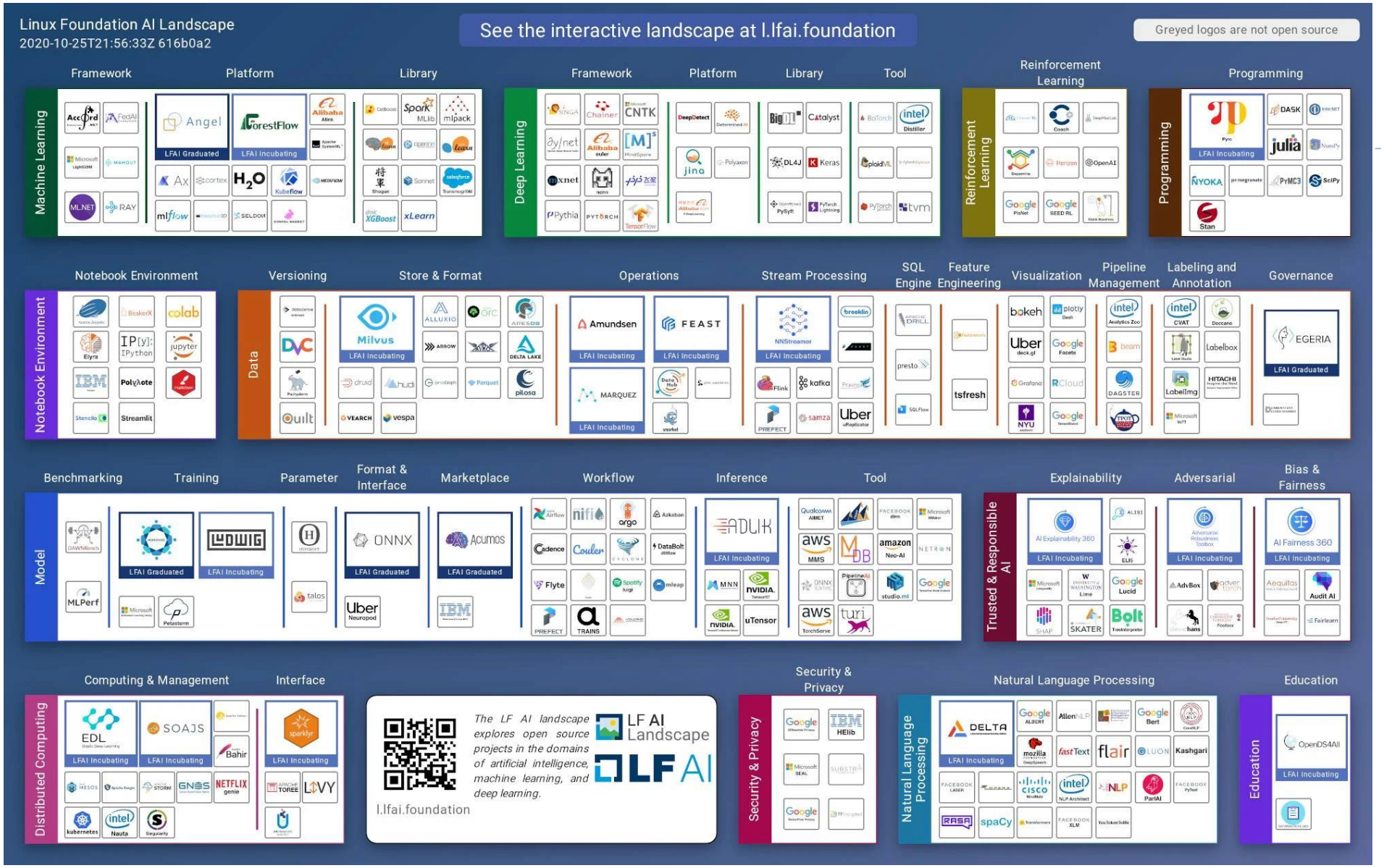
MLOps Stack Template

Version 1.1
October 2020

Component	Requirements	Tooling
Data analysis		
Experimentation		
Feature store		
Code repository		
ML pipeline		
Metadata store		
Model registry		
Model serving		
Model monitoring		

MLOps Setup Components	Tools
Data Analysis	Python, Pandas
Source Control	Git
Test & Build Services	PyTest & Make
Deployment Services	Git, DVC
Model & Dataset Registry	DVC [aws s3]
Feature Store	Project code library
ML Metadata Store	DVC
ML Pipeline Orchestrator	DVC & Make

Source: ml-ops.org


Linux Foundation AI Landscape
2020-10-25T21:56:33Z 616b0a2
See the interactive landscape at l.lfai.foundation
Greyed logos are not open source
Machine Learning
Framework
Platform
Library
Deep Learning
Framework
Platform
Library
Tool
Reinforcement Learning
Reinforcement Learning
Programming
Programming
Notebook Environment
Notebook Environment
Data
Versioning
Store & Format
Operations
Stream Processing
SQL Engine
Feature Engineering
Visualization
Pipeline Management
Labeling and Annotation
Governance
Model
Benchmarking
Training
Parameter
Format & Interface
Marketplace
Workflow
Inference
Tool
Trusted & Responsible AI
Explainability
Adversarial
Bias & Fairness
Distributed Computing
Computing & Management
Interface
LF AI Landscape
LF AI
l.lfai.foundation
The LF AI landscape explores open source projects in the domains of artificial intelligence, machine learning, and deep learning.
Security & Privacy
Security & Privacy
Natural Language Processing
Natural Language Processing
Education
Education
LFAI Graduated
LFAI Incubating