# SOFTWARE PROCESS

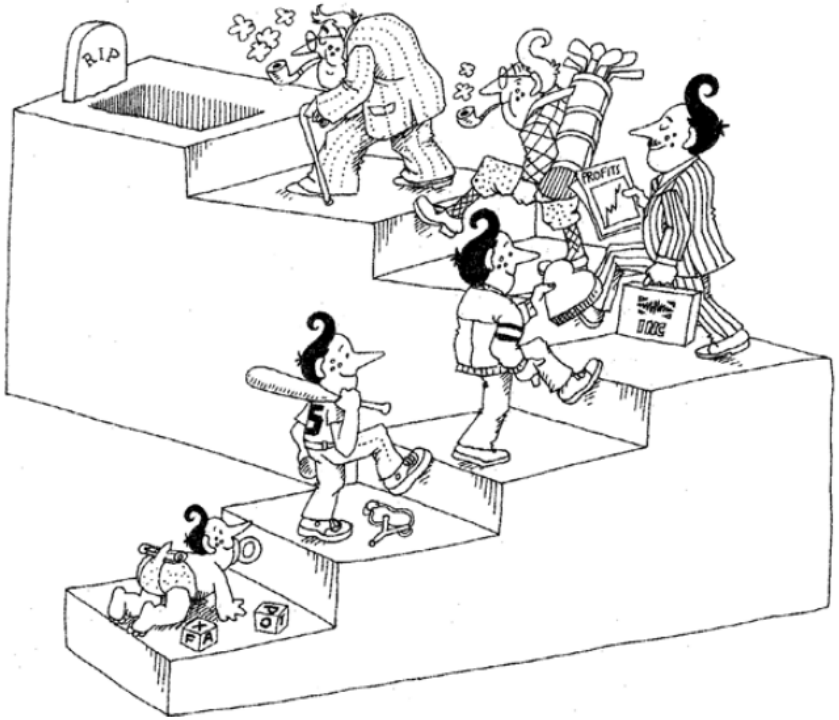
# OR

# SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

**TEXTBOOK – CHAPTER 3**

# Software Life Cycle

- The term “Life Cycle” is based on the metaphor of the life of a person:

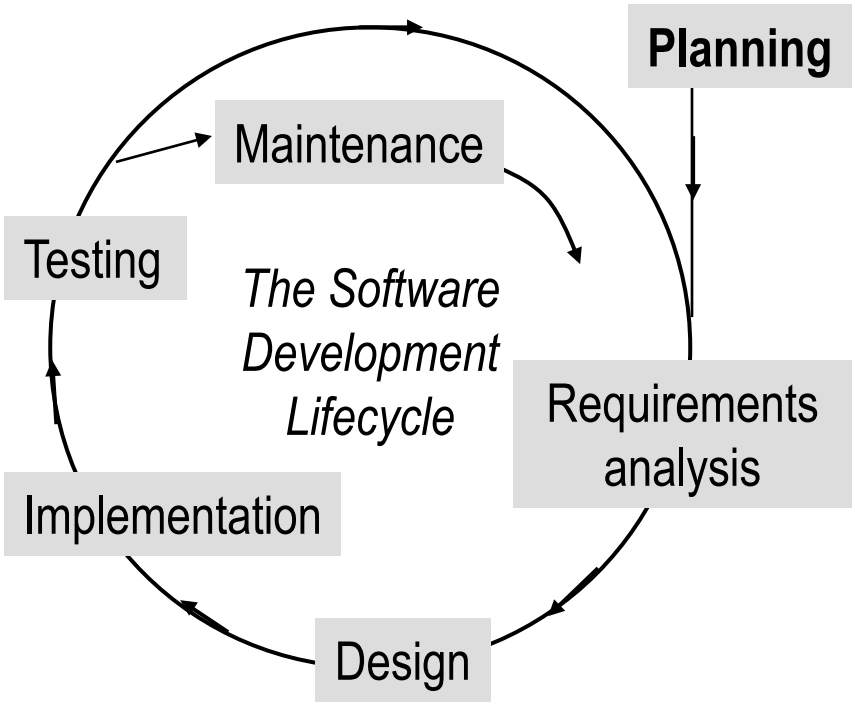



BOOCH, G. (1994): *Object Oriented Analysis and Design with Applications*

# Software Processes

In a nutshell:

Coherent sets of activities for specifying, designing, implementing, testing, and maintaining software systems.




Phase most relevant to this chapter is shown in bold

Learning goals of this chapter

- What are the main activities of software processes?
- What are the main software process types?
- How would a team, such a student team, go about selecting a process?

# Process

- Framework for carrying out the activities of a project in an *organized* and *disciplined* manner.

- Imposes structure

- In Software Engineering:
  - Waterfall
  - Iterative
  - Etc…

# Software Process

- Software project composed of activities
  - E.g. planning, design, testing, etc.
- Activities organized into phases or stages
- A *software process:*
  - prescribes the order and frequency of phases
  - specifies criteria for moving from one phase to the next
  - defines the deliverables of the project.

# Umbrella Activities

- Generic activities implemented ***throughout*** the life of a project
  - Project management
  - Configuration management
  - Quality management
  - Risk management.

# Software Process Benefits

- Process **DOES NOT** mean
  - "overhead"
  - "unnecessary paperwork"
  - "longer schedules"
  - etc.

- Software process has **positive** effect if applied correctly
  - Meet schedules
  - Higher quality
  - More maintainable.

# Software Process Phases

## Phases of Software Processes

1. **Inception**

   Software product is conceived and defined

2. **Planning**

   Initial schedule, resources and cost are determined

3. **Requirements Analysis**

   Specify what the application must do; answers **"*what*?"**

4. **Design**

   Specify the parts and how they fit; answers **"*how*?"**

5. **Implementation**

   Write the **code**

6. **Testing**

   **Execute** the application with input test data

7. **Maintenance**

   Repair defects and add capability

# Example – Video Store Application

## Software Process Phases: Video Store Example

- **Inception**

  "…An application is needed to keep track of video rentals …"

- **Planning** (Software Project Management Plan)

  "…The project will take 12 months, require 10 people and cost $2M…"

- **Requirements Analysis** (Product: Software Requirements Spec.)

  "…The clerk shall enter video title, renter name and date rented. The system shall …"

- **Design** (Software Design Document: Diagrams and text)

  " … classes *DVD*, *VideoStore*, …, related by …"

- **Implementation** (Source and object code)

  … class DVD{ String title; … } …

- **Testing** (Software Test Documentation: test cases and test results)

  "… Ran test case: *Rent "The Matrix" on Oct 3; rent "SeaBiscuit" on Oct 4; return "The Matrix" on Oct 10…*

  Result: *"SeaBiscuit" due Oct 4, 2004 balance of $8. (correct) …"*

- **Maintenance** (Modified requirements, design, code, and text)

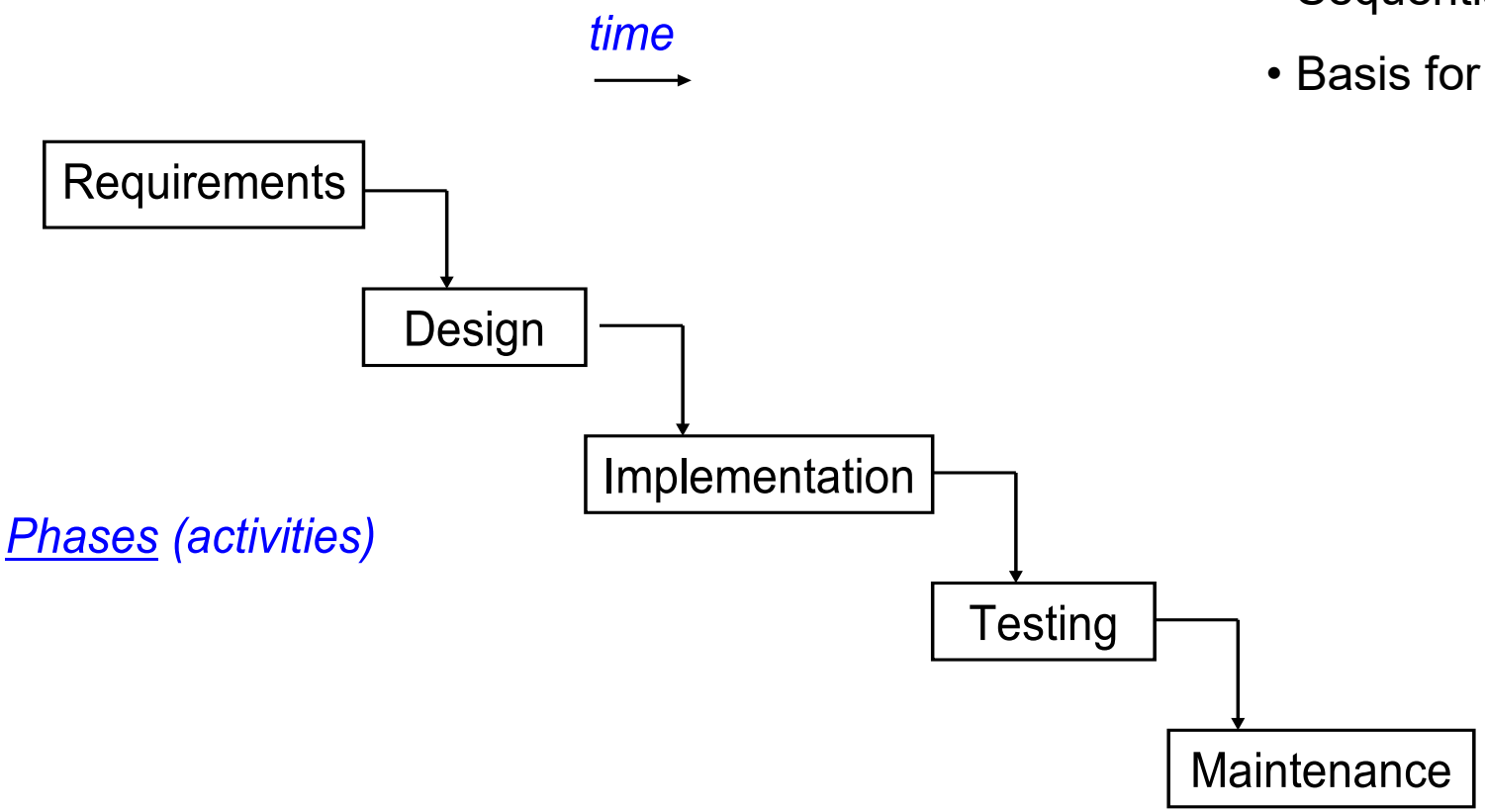
  Defect repair: "Application crashes when balance is $10 and attempt is made to rent *"Gone With the Wind"* . …"

  Enhancement: "Allow searching by director."

# Waterfall Process

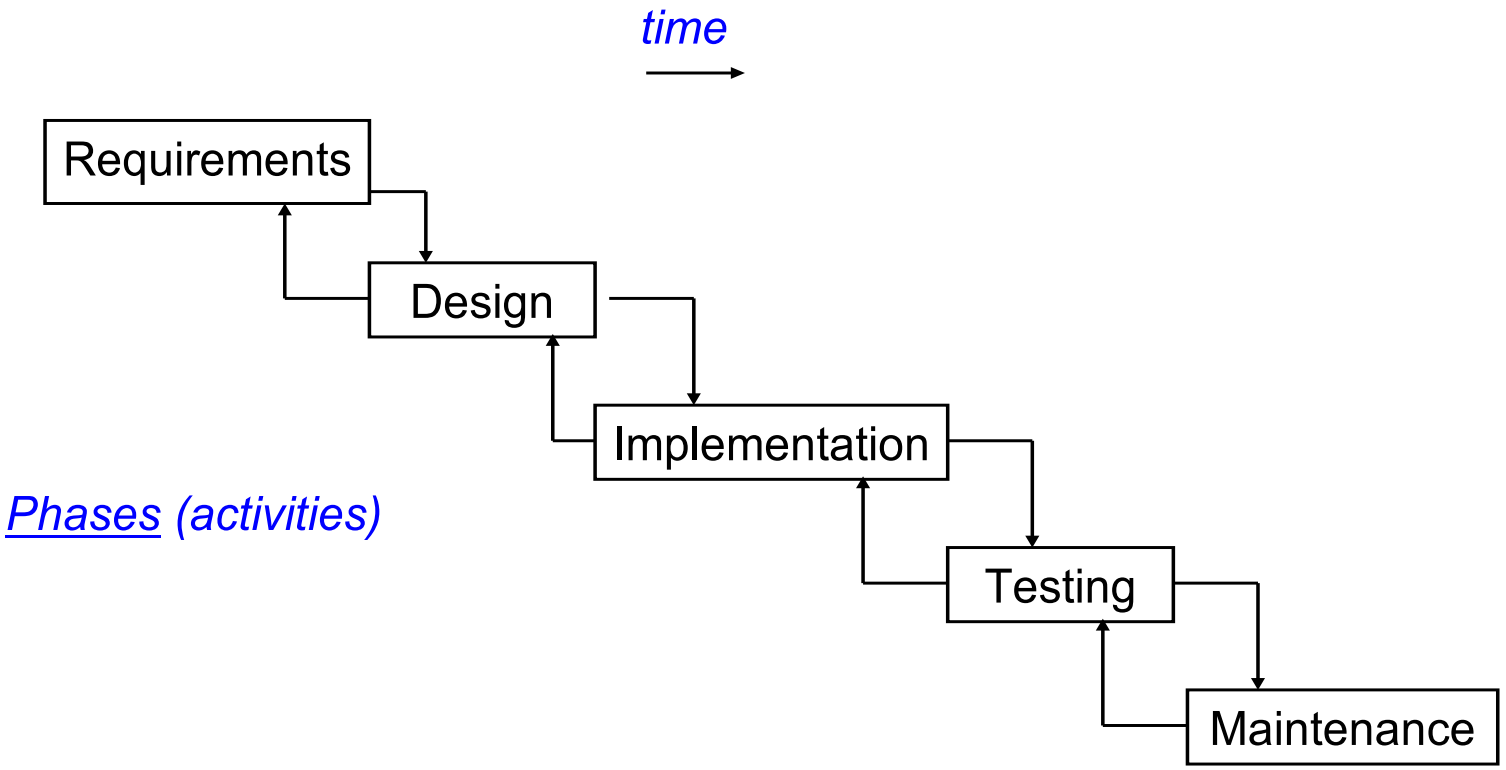
## The Waterfall Software Process

- Simplest process
- Sequential
- Basis for others



# Waterfall Process – with feedback

## The Waterfall Software Process with Feedback



# Waterfall Process - Advantages

- Simple and easy to use
- Practiced for many years
- Easy to manage
- Facilitates allocation of resources
- Works well for projects where requirements are very well understood.

# Waterfall Process - Disadvantages

- Requirements must be known up front
- Hard to estimate resources/cost reliably
- No feedback of system by stakeholders until after testing phase
- This makes it difficult to respond to changing customer requirements
- Major problems aren't discovered until late in process
- Therefore, this model is only appropriate when the requirements are well-understood.

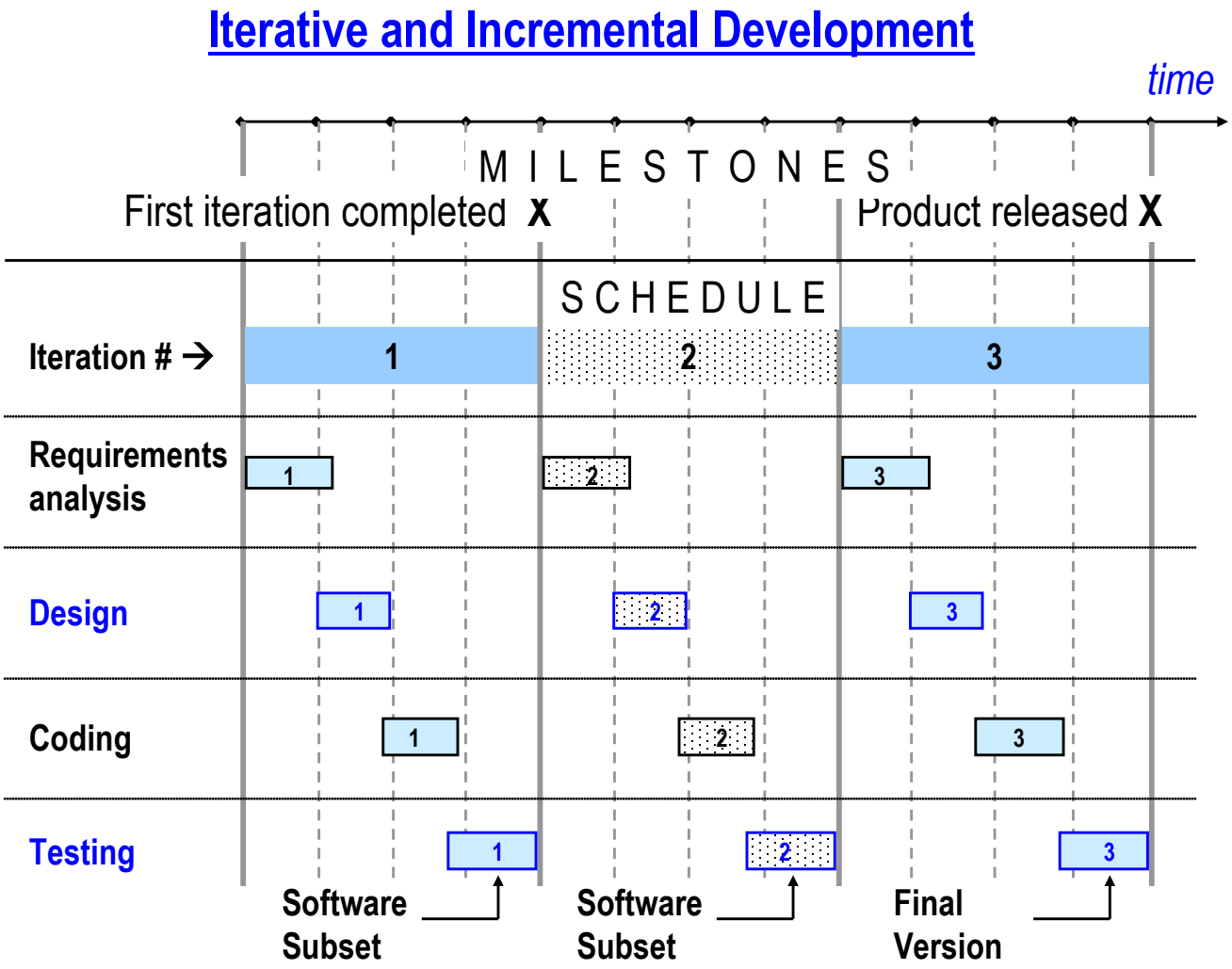
# Iterative Process

- Software projects rarely follow ***strict*** waterfall
- Some ***iteration*** between specifications, design, implementation and test
- Some parallelism among tasks
- Requires discipline, e.g. update specifications when design changes.

# Iterative and Incremental

- ***Iterative***
  - repeated execution of the waterfall phases, in whole or in part, resulting in a refinement of the requirements, design and implementation
- ***Incremental***
  - operational code produced at the end of an iteration
  - supports a subset of the final product functionality and features
- Artifacts evolve during each phase
- Artifacts considered complete only when software is released.

# Iterative and Incremental (cont.)

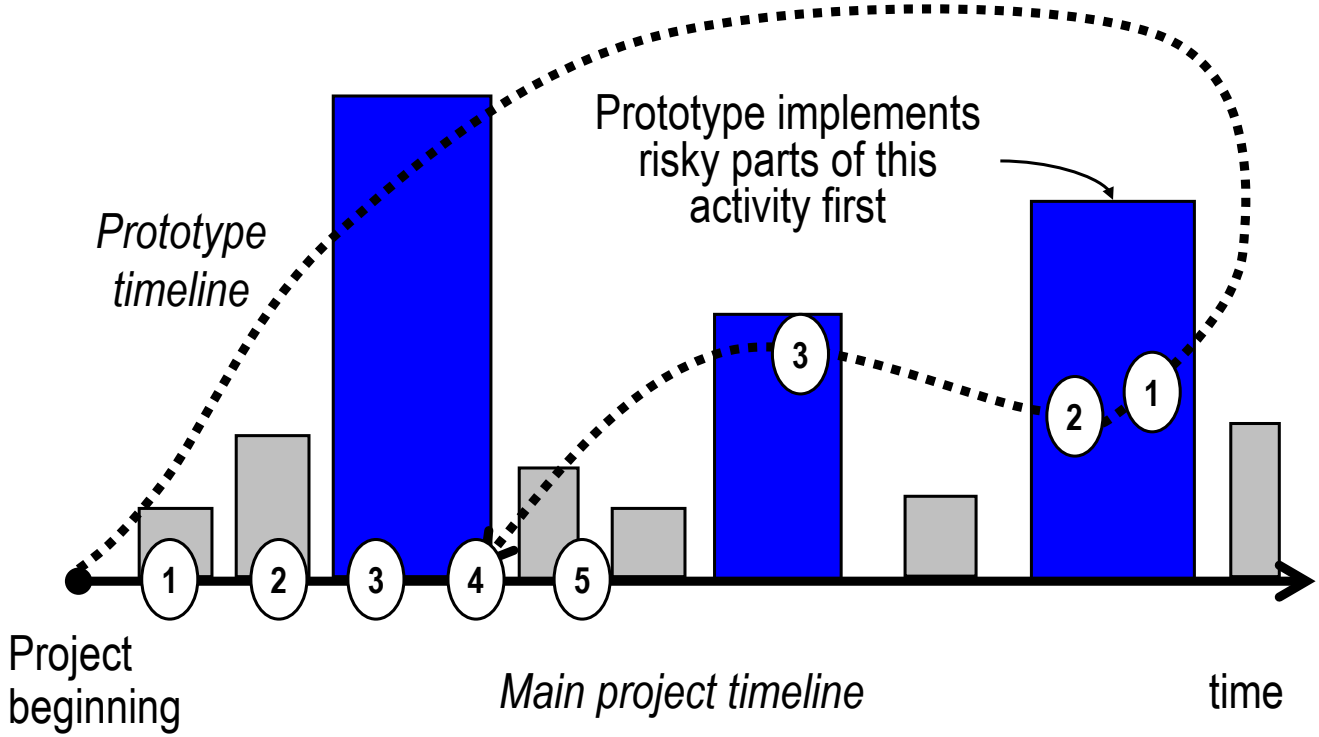


# Release Types

- Proof of concept
- Feasibility study
- Prototype
- “Internal” release
- “External” release

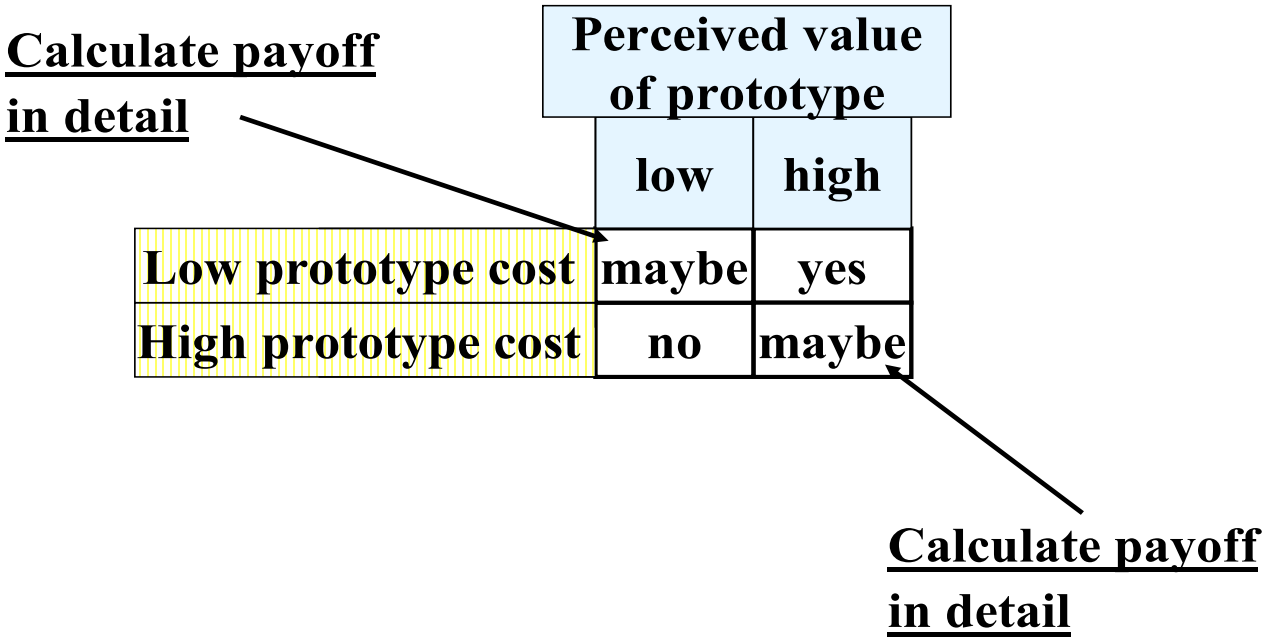
# Prototyping

## Prototype Rationale

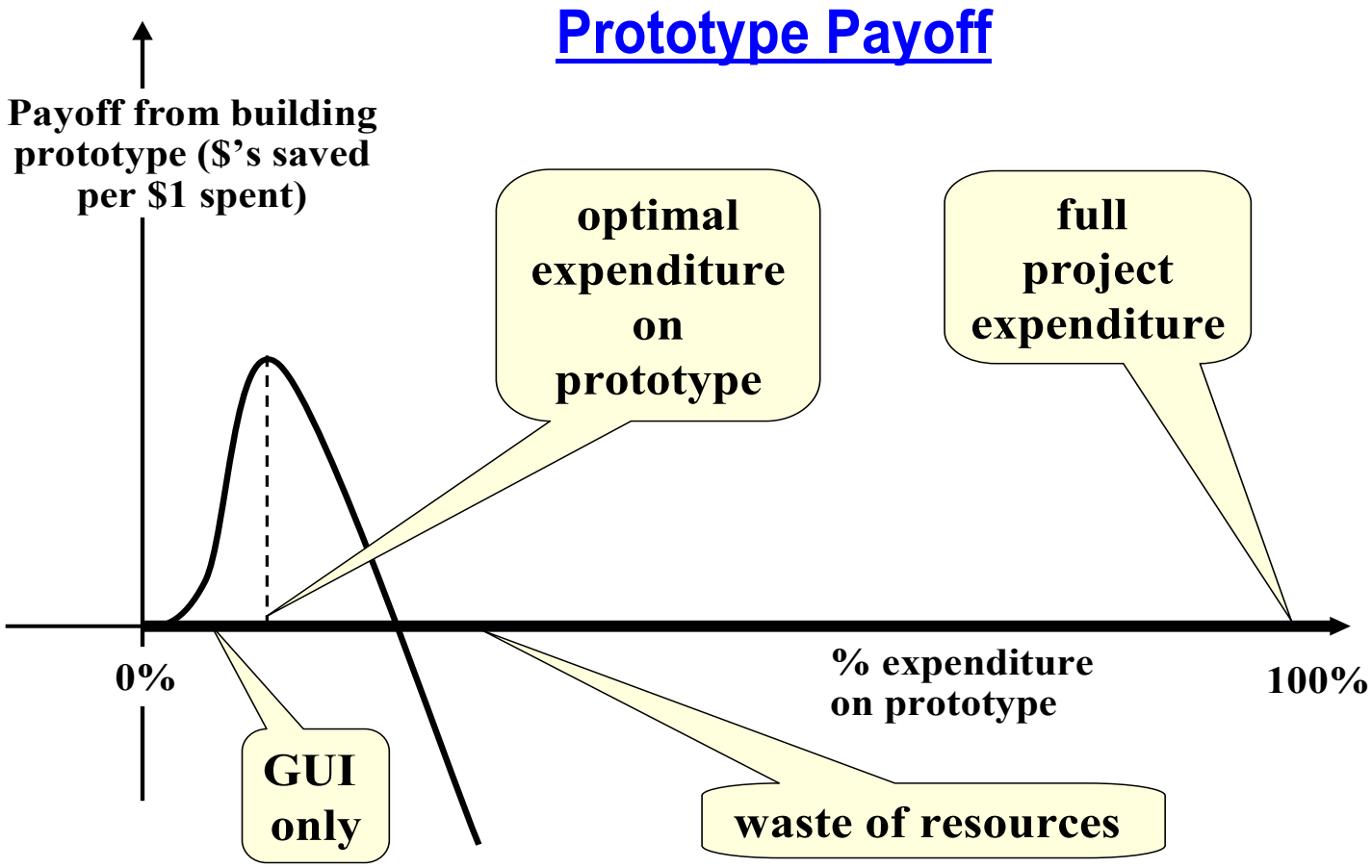


# Prototyping (cont.)

## Prototype Payoff: First Cut



# Throw-Away Prototyping



# Prototyping Costs

## Prototype Payoff Calculations for E-commerce Clothing Application

| | Esti-mated cost | Gross Benefit excluding code re-use | | Percentage of prototype code reused in application | Net Payoff | | |
|---|---|---|---|---|---|---|---|
| | | min | max | | min | max | average |
| Prototype feature | *B* | *D* | *E* | *C* | *D-(1-C)B* | *E-(1-C)B* | |
| | | | | | | | |
| 1. GUI screenshots | $10,000 | $10,000 | $80,000 | 50% | $5,000 | $75,000 | $40,000 |
| | | | | | | | |
| 2. Transaction | $50,000 | $10,000 | $300,000 | 80% | $0 | $290,000 | $145,000 |
| security | | | | | | | |
| 3. Complete | $80,000 | $10,000 | $400,000 | 50% | -$30,000 | $200,000 | $85,000 |
| transaction | | | | | | | |
| 4. Customer | $120,000 | $20,000 | $140,000 | 30% | -$64,000 | $56,000 | -$4,000 |
| tries on clothing | | | | | | | |

# Iterative and Incremental Advantages

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services are released first and receive the most testing
- Paper: How Microsoft Builds Software
  - Synchronize and Stabilize

# Iterative and Incremental

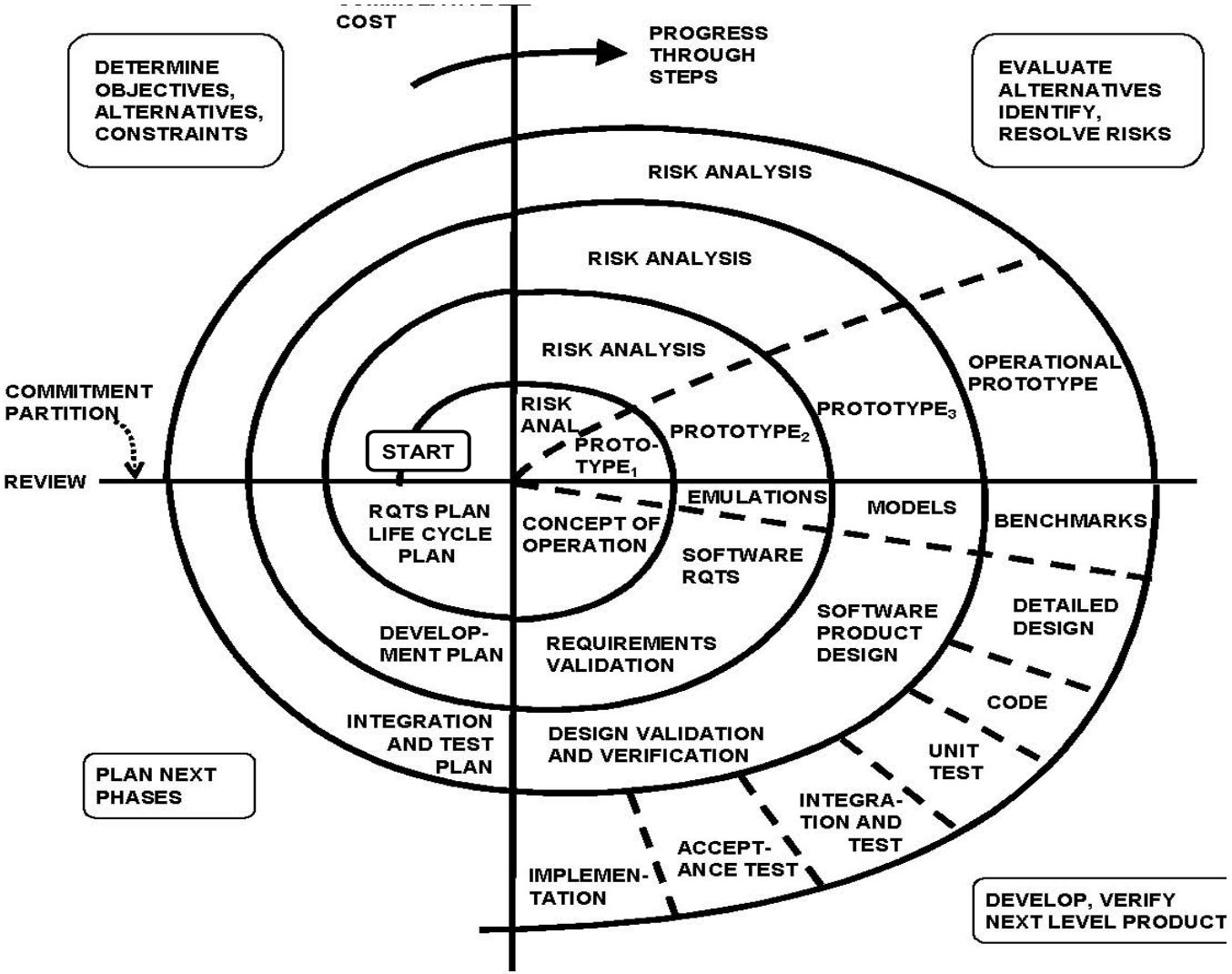
- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems.

# Spiral Model

- Barry Boehm, TRW Defense Systems,1988
- One of the earliest and best known iterative and incremental processes
- Risk-driven process
- Project starts at the center, and each cycle of the spiral represents one iteration
- Goal of each cycle is to increase the degree of system definition and implementation, while decreasing the degree of risk.

# Spiral Model

## The Spiral Model



# Spiral Model - Iteration Steps

1. Identify critical objectives and constraints
2. Evaluate project and process alternatives
3. Identify risks
4. Resolve (cost-effectively) a subset of risks using analysis, emulation, benchmarks, models and prototypes
5. Develop project deliverables including requirements, design, implementation and test
6. Plan for next and future cycles – update project plan including schedule, cost and number of remaining iterations
7. Stakeholder review of iteration deliverables and their commitment to proceed based on their objective being met.

# Spiral Model - Advantages

- ***Risks are managed early and throughout the process*** – risks are reduced before they become problematic

- ***Software evolves as the project progresses*** - errors and unattractive alternatives eliminated early.

- ***Planning is built into the process*** – each cycle includes a planning step to help monitor and keep a project on track.

# Spiral Model - Disadvantages

- ***Complicated to use*** - risk analysis requires highly specific expertise. There is inevitably some overlap between iterations.

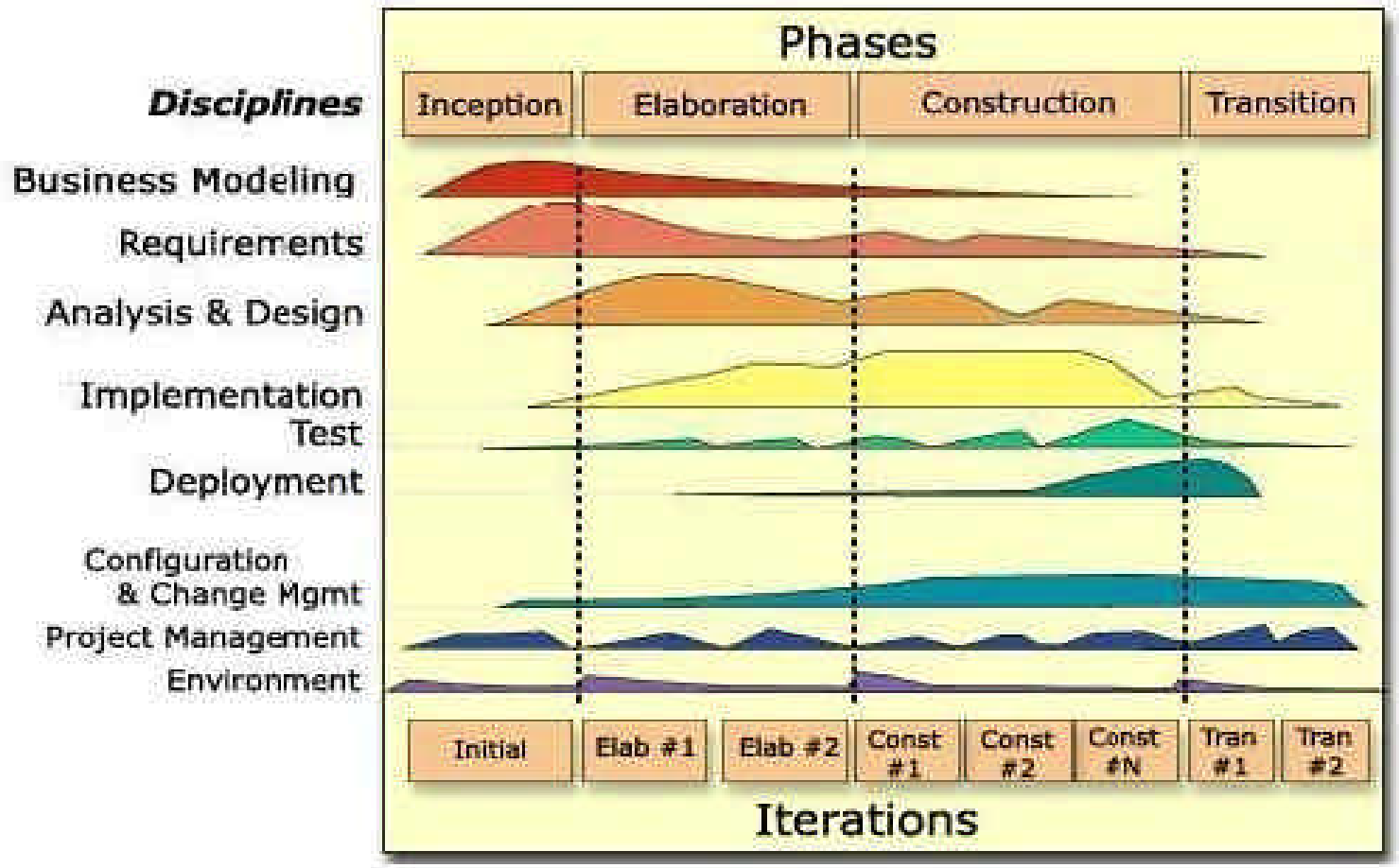
- ***May be overkill for small projects*** – complication may not be necessary for smaller projects. Does not make sense if the cost of risk analysis is a major part of the overall project cost.

# Unified Process

- Developed by Jacobson, Rumbaugh and Booch
- Major elaboration and refinement of Spiral
- Use-case driven
- Commercial product: Rational Unified Process (RUP).

# Unified Process

## Unified Process




http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

# Inception

- Establish feasibility
- Make business case
- Establish product vision and scope
- Estimate cost and schedule, including major milestones
- Assess critical risks
- Build one or more prototypes.

# Elaboration

- Specify requirements in greater detail
- Architectural baseline
- Iterative implementation of core architecture
- Refine risk assessment and resolve highest risk items
- Refine project plan, including detailed plan for beginning Construction iterations.

# Construction

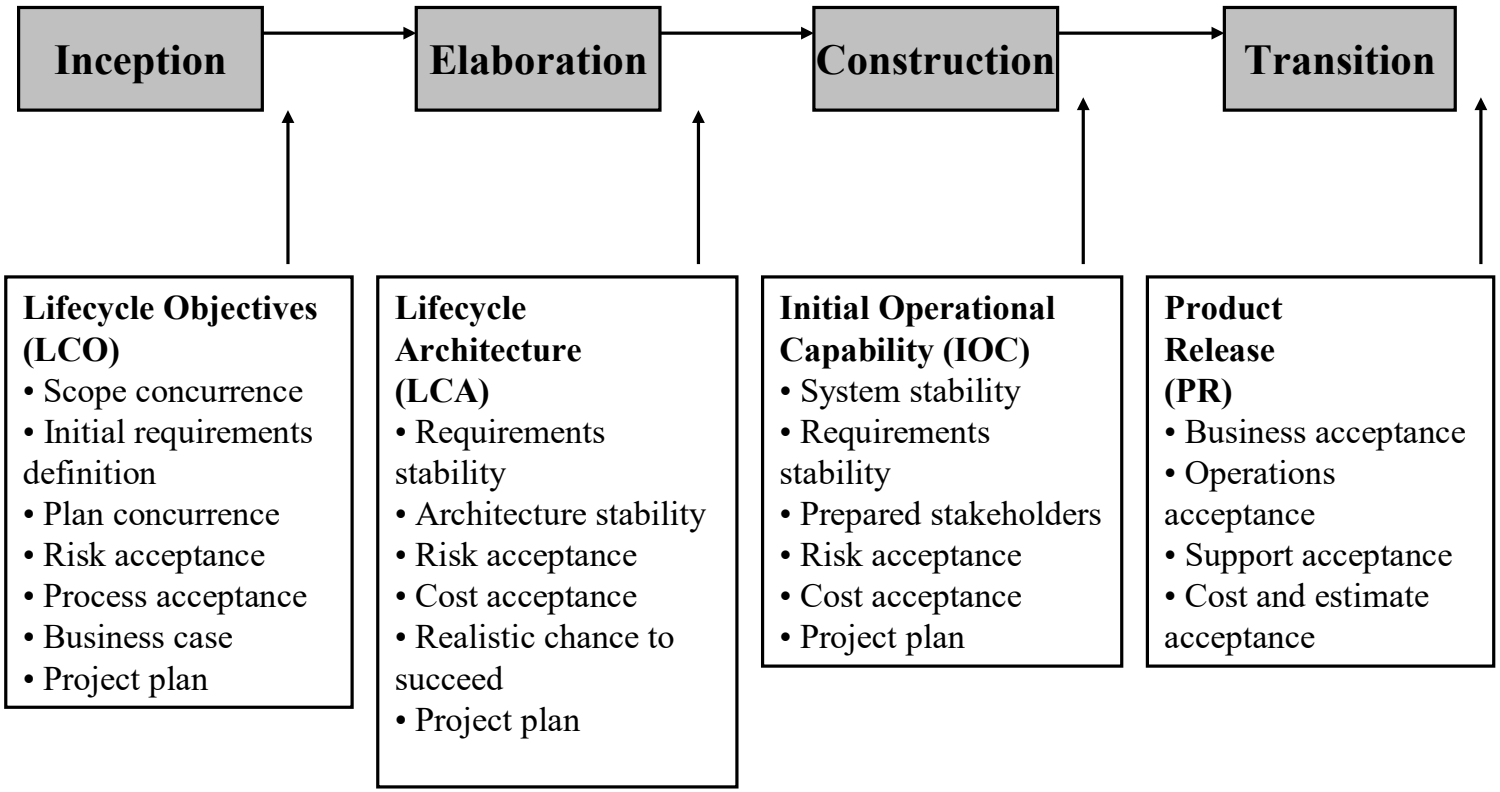
- Complete remaining requirements
- Iterative implementation of remaining design
- Thorough testing and preparation of system for deployment.

# Transition

- Beta tests
- Correct defects
- Create user manuals
- Deliver the system for production
- Training of end users, customers and support
- Conduct lessons learned.

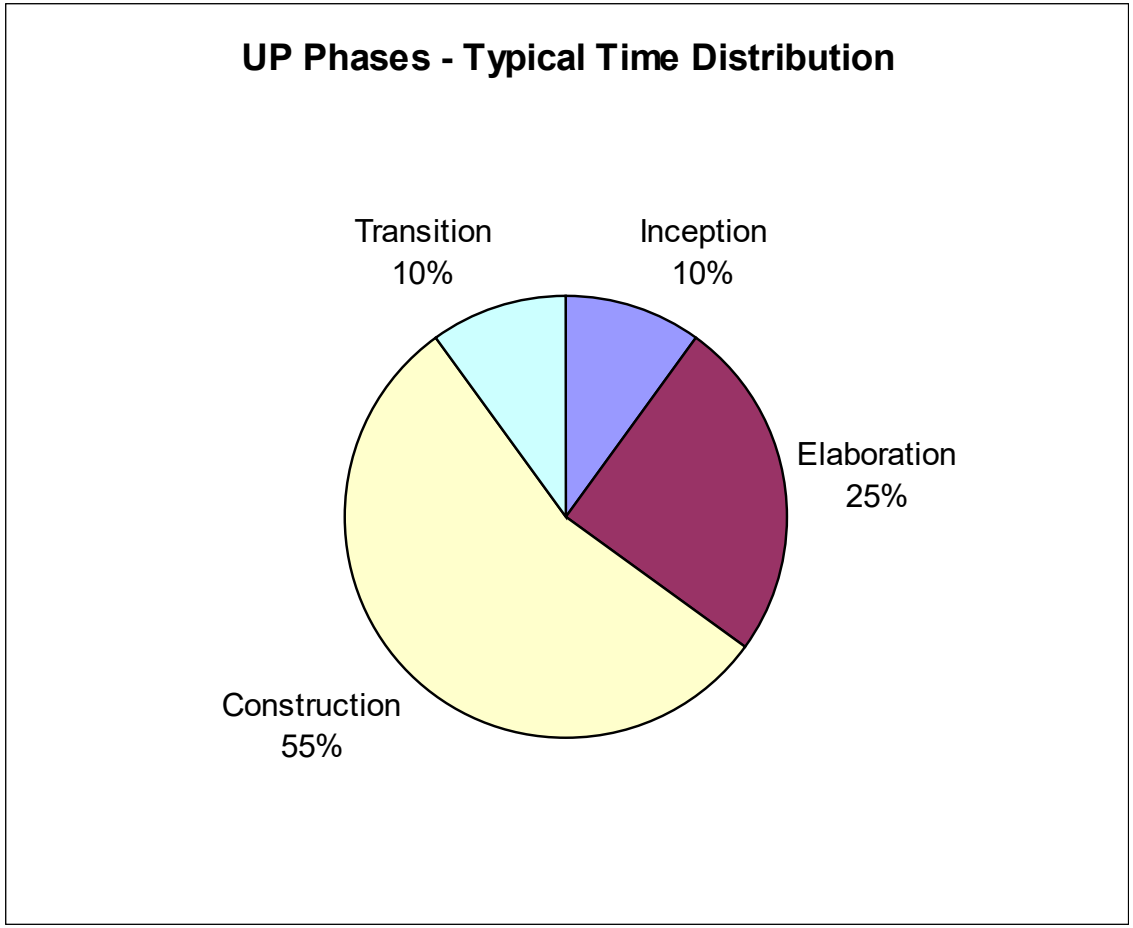
# Unified Process Milestones

## Unified Process Milestones




Adapted from http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

# Phases Times




Adapted from: Ambler, S.W., *A Manager's Introduction to the Rational Unified Process (RUP)*

# Agile Philosophy

- What is Agile?
- https://www.youtube.com/watch?v=Z9QbYZh1YXY&t=13s

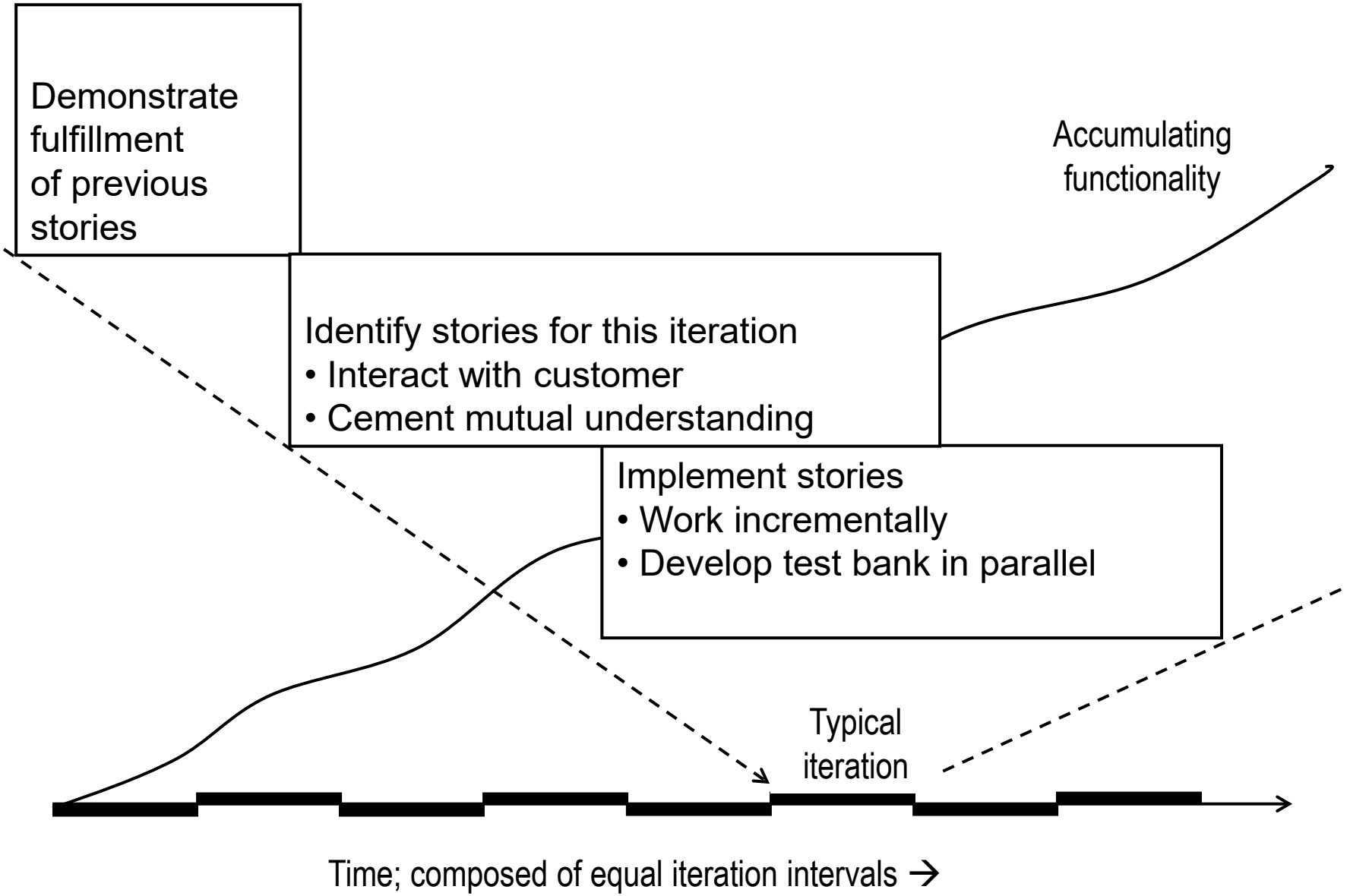
# Agile Processes

## The Agile Manifesto:

**Agile processes 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

# The Agile Process



# Extreme and Pair Programming

- New approach to development based on the development and delivery of very small increments of functionality
- Relies on constant code improvement, user involvement in the development team and pairwise programming
- Programmers work in pairs in just one computer.

# Open Source

## Reasons for Making a Project Open Source *1*

☺Leveraging large number of resources

☺Professional satisfaction

☺ To enable tailoring and integration

☺ Academic and research

☺ To gain extensive testing

☺ To maintain more stably

Richard Stallman
Originator of open source development
http://stallman.org/

# Open Source (cont.)

## Reasons for Making a Project Open Source 2

☺To damage a competitor's product

☺To gain market knowledge

☺ To support a core business

☺ To support services

# Open Source (cont.)

## Reasons Against Open Source

☹ Documentation inconsistent or poor

☹No guarantee that developers will appear

☹No management control

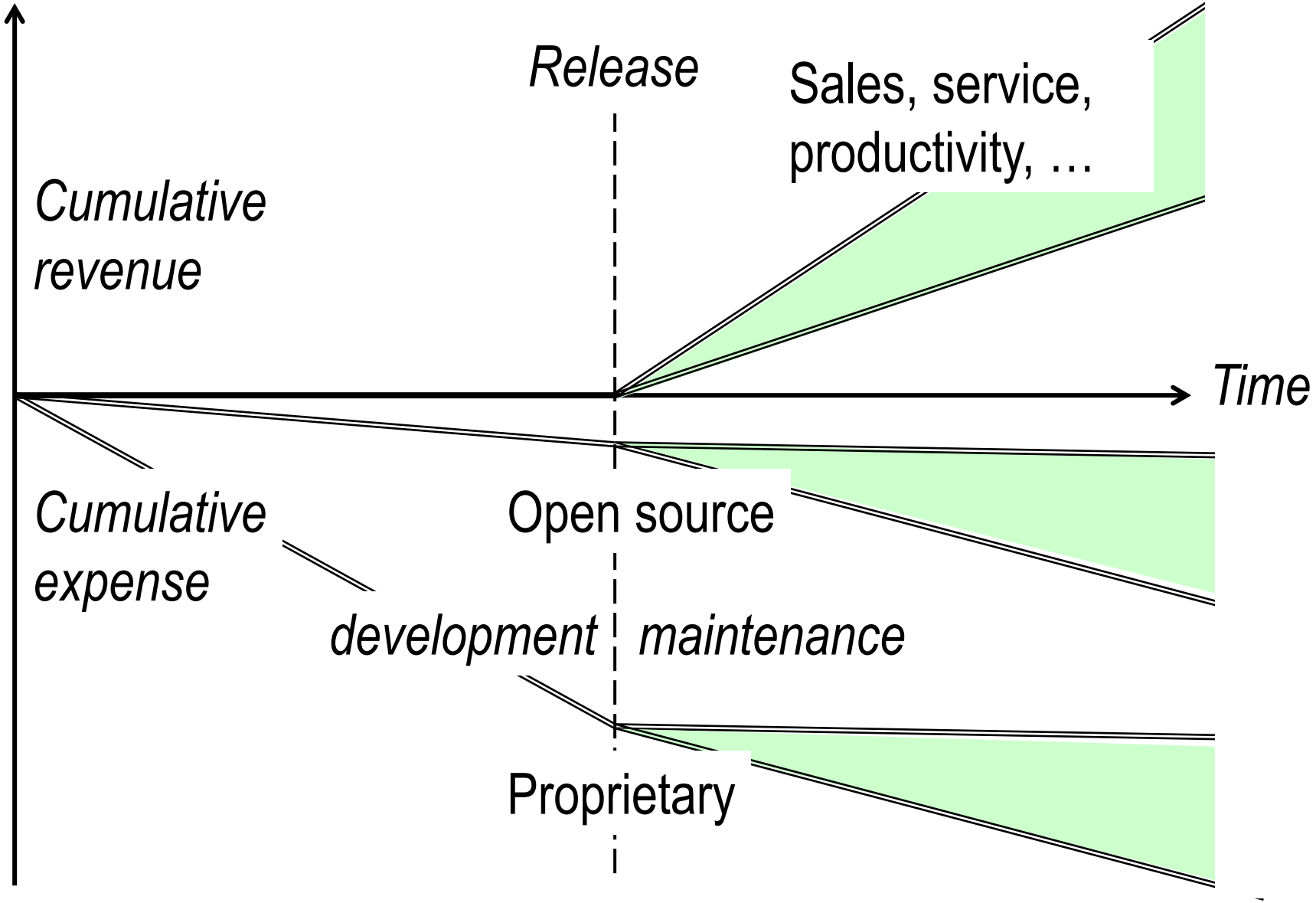
☹No control over requirements

☹Visibility to competitors

Bill Gates
http://www.microsoft.com/billgates/default.asp

# Open Source / Proprietary Processes

## Need Strong Players: Eclipse, Apache, MySQL



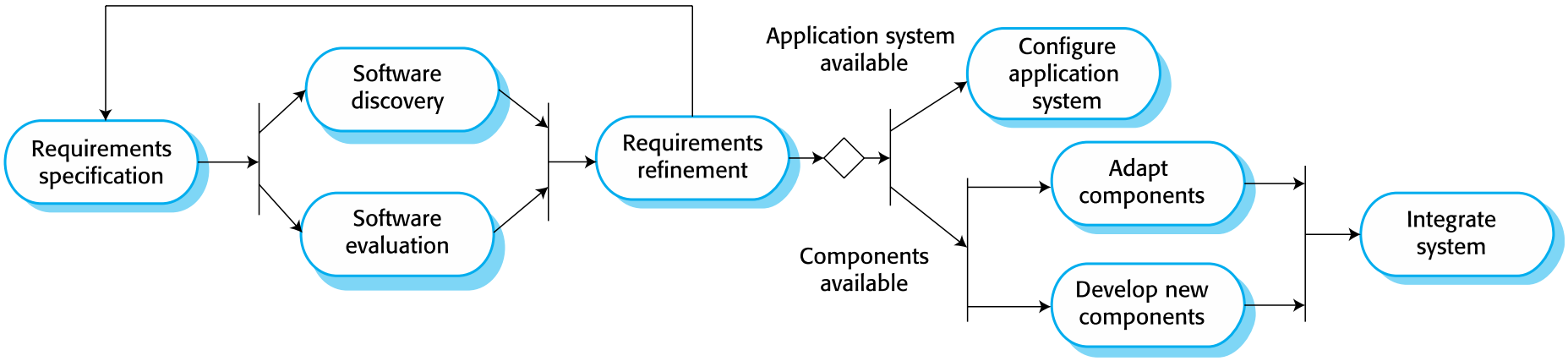
# No One Size Fits All

- X Model paper
- Y Model paper
- V Model
- Software Product Line Model
- Microsoft Model
- Apple Model (?)
- Paper: A New Software Engineering
- Your Model: Contribution-1

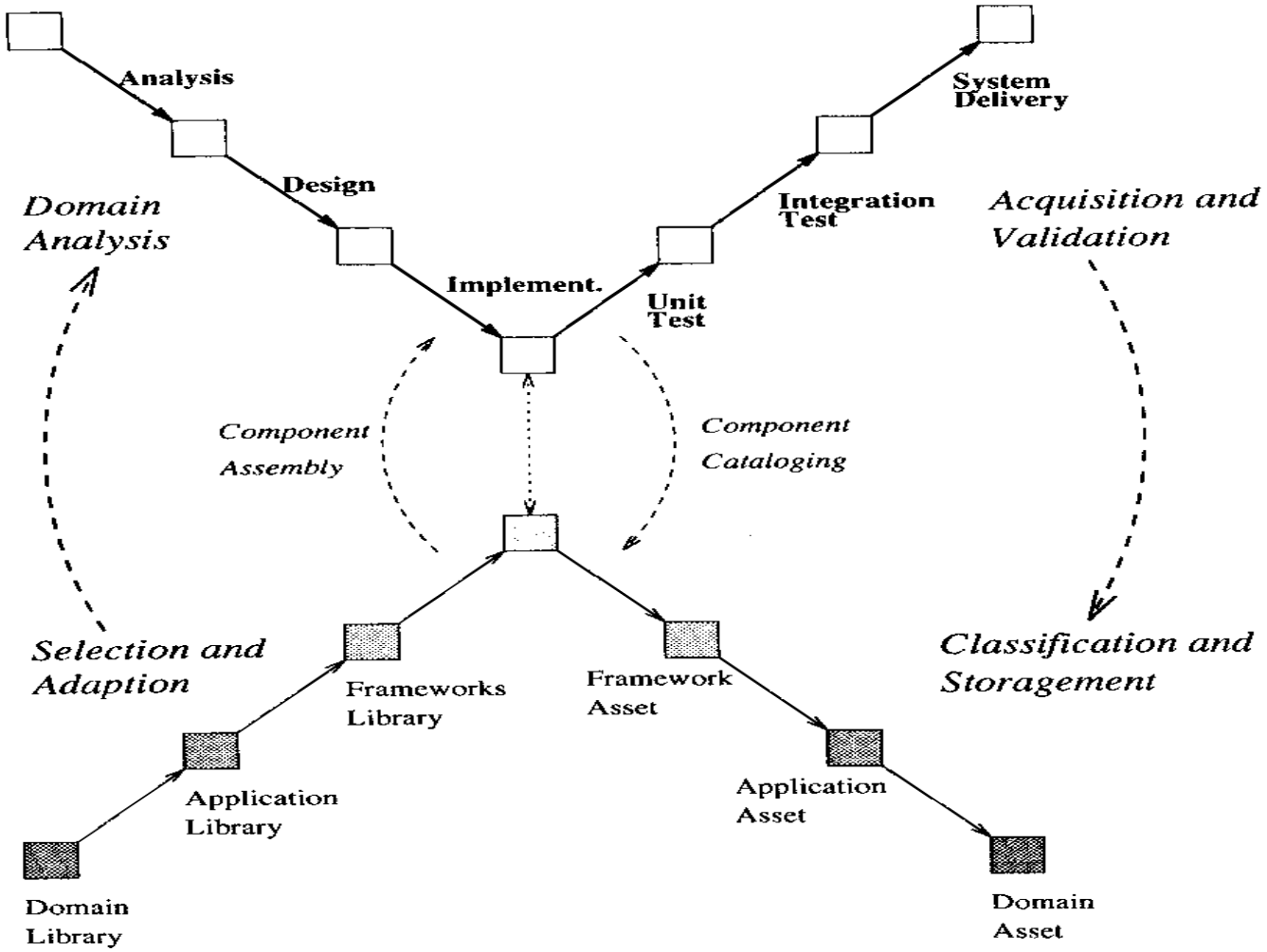
# Software Reusability

- Based on software reuse where systems are integrated from existing components (APIs) or application systems (sometimes called COTS - Commercial-off-the-shelf systems).
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system
  - Github, SourceForge, Google Code, etc.

# Reuse-Oriented Software Engineering



# The X Model with Reusability



# Summary: Software Process Models

- Blog: https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

# Automated Process Support (CASE)

- Computer-aided software engineering (CASE) is software to support software development and evolution processes
- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated software configuration management
  - Effective software project management
  - Automated process throughout the SDLC.

# CASE Technology

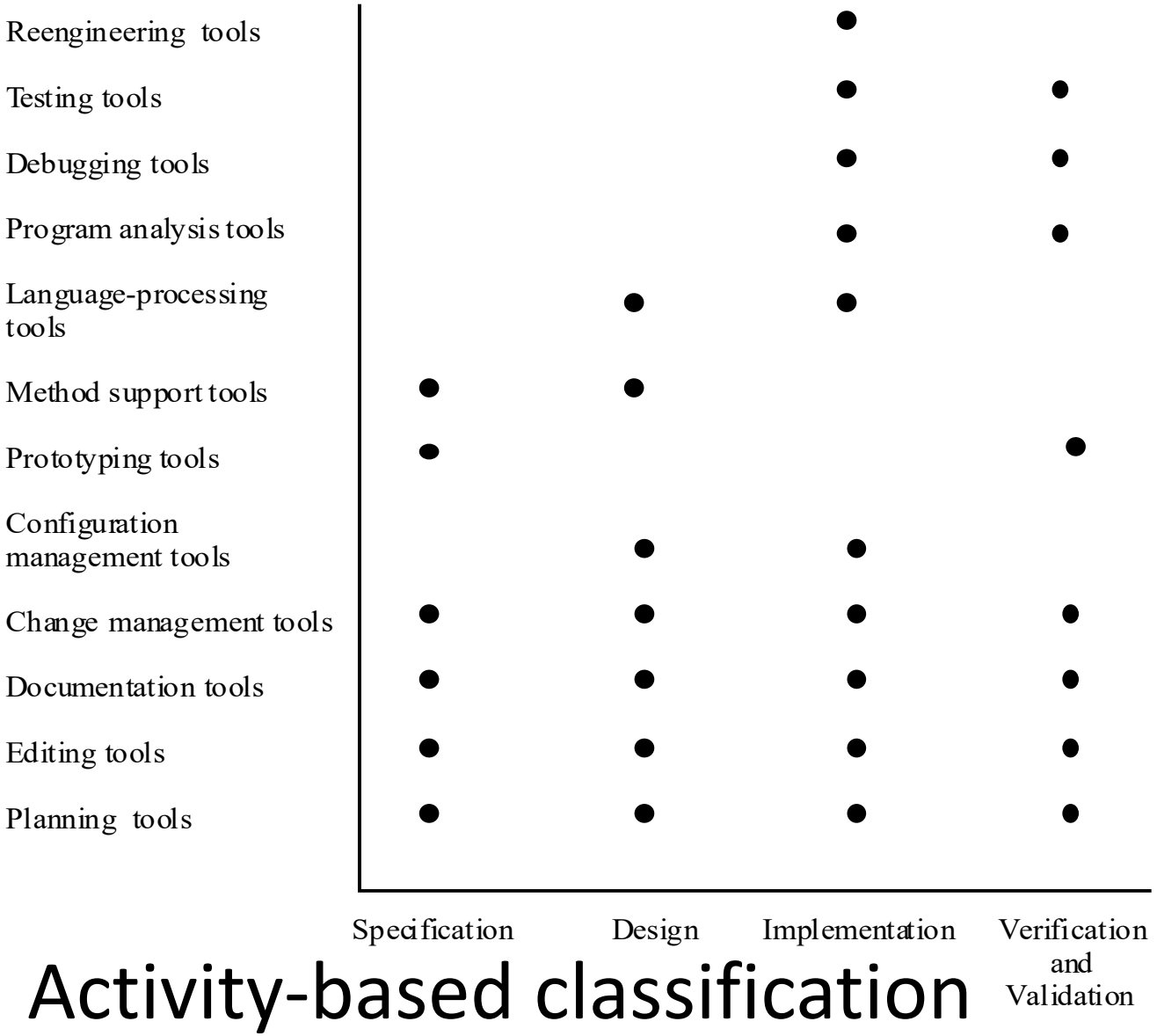
- CASE technology has led to significant improvements in the software process though not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automatable
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions.

# CASE Classification

- Classification helps us understand the different types of CASE tools and their support for process activities
- Functional perspective
  - Tools are classified according to their specific function, i.e. configuration management
- Process perspective
  - Tools are classified according to process activities that are supported, i.e. static testing
- Integration perspective
  - Tools are classified according to their organisation into integrated units, i.e. stand-alone or the whole software process.

# Functional Tool Classification

| Toolt ype | Examples |
|---|---|
| Pla nning tools | PERT tools, estim at ion tools, spreadsheet s |
| Edit ing tools | Te xt ed tors, diagram e editors, word proce ss ors |
| Change ma na gement tools | Requiremen ts trac eab ili ty tod s, chan ge control sys te ms |
| Configuration managemen t tools | Ver si onm anag ements yst ems, sys te m bui lding tools |
| Prototyping tools | Ver yh igh-le vel la nguag es, usern in terface gen era tors |
| Method-su pport tools | Design ed tors, data dic tio narie s, co de gene rators |
| La nguag e-proce ssi ng tools | Com pile rs, inter pr ete rs |
| Program ana lysis tools | Cro ss refe rence gener ato rs, s ta tic anal ys ers, dynamic ana lysers |
| Te st in gt ools | Te st data gene rators, file co mp ar ators |
| Deb ugging tools | Inter act ive de buggings yst ems |
| Doc um en ta tion tools | Page l ayout progra ms, i ma ge edito rs |
| Re-e ngine ering tools | Cro ssr efer enc es yst ems, program re-stru ct urings yst ems |

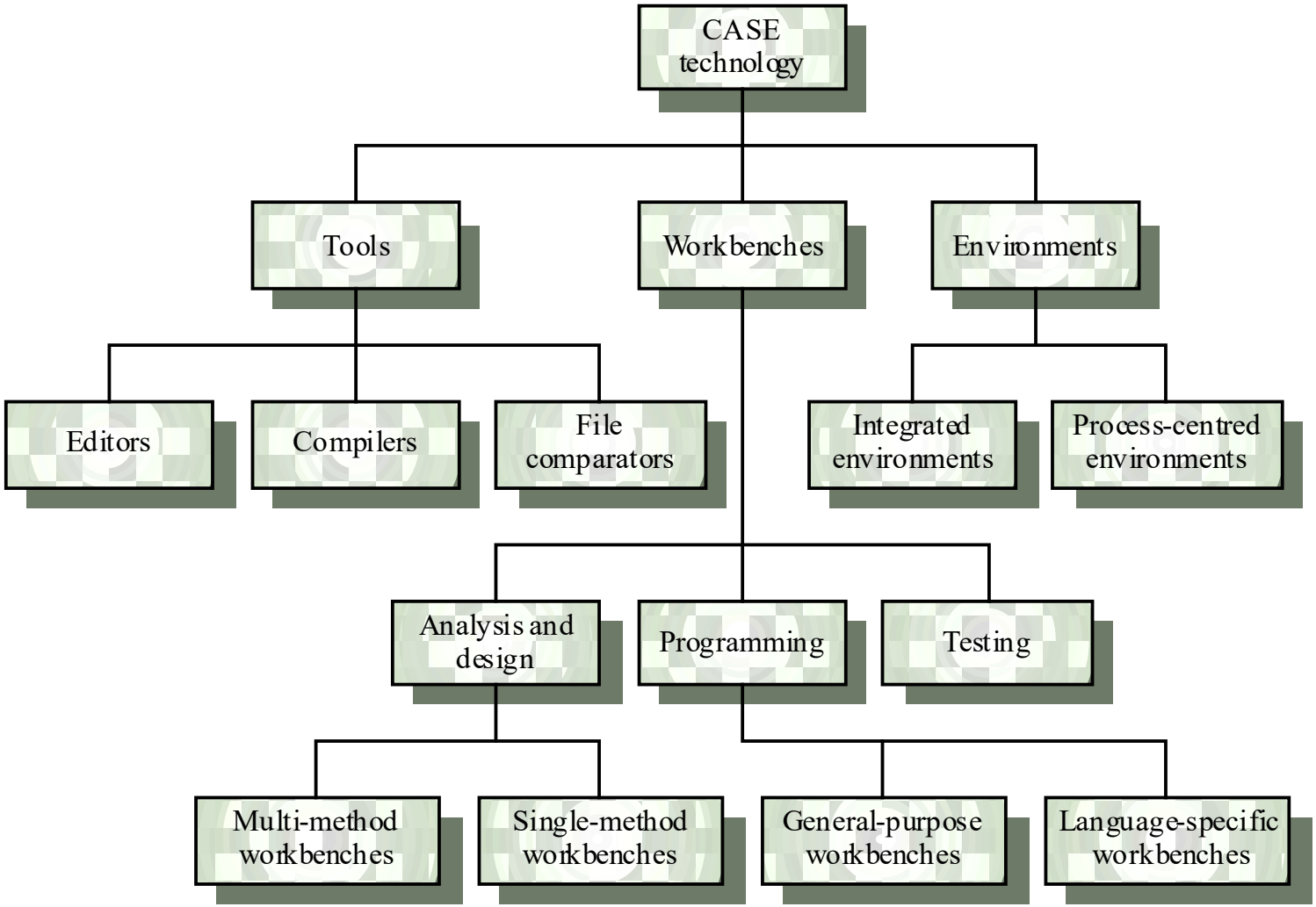



# Activity-based classification

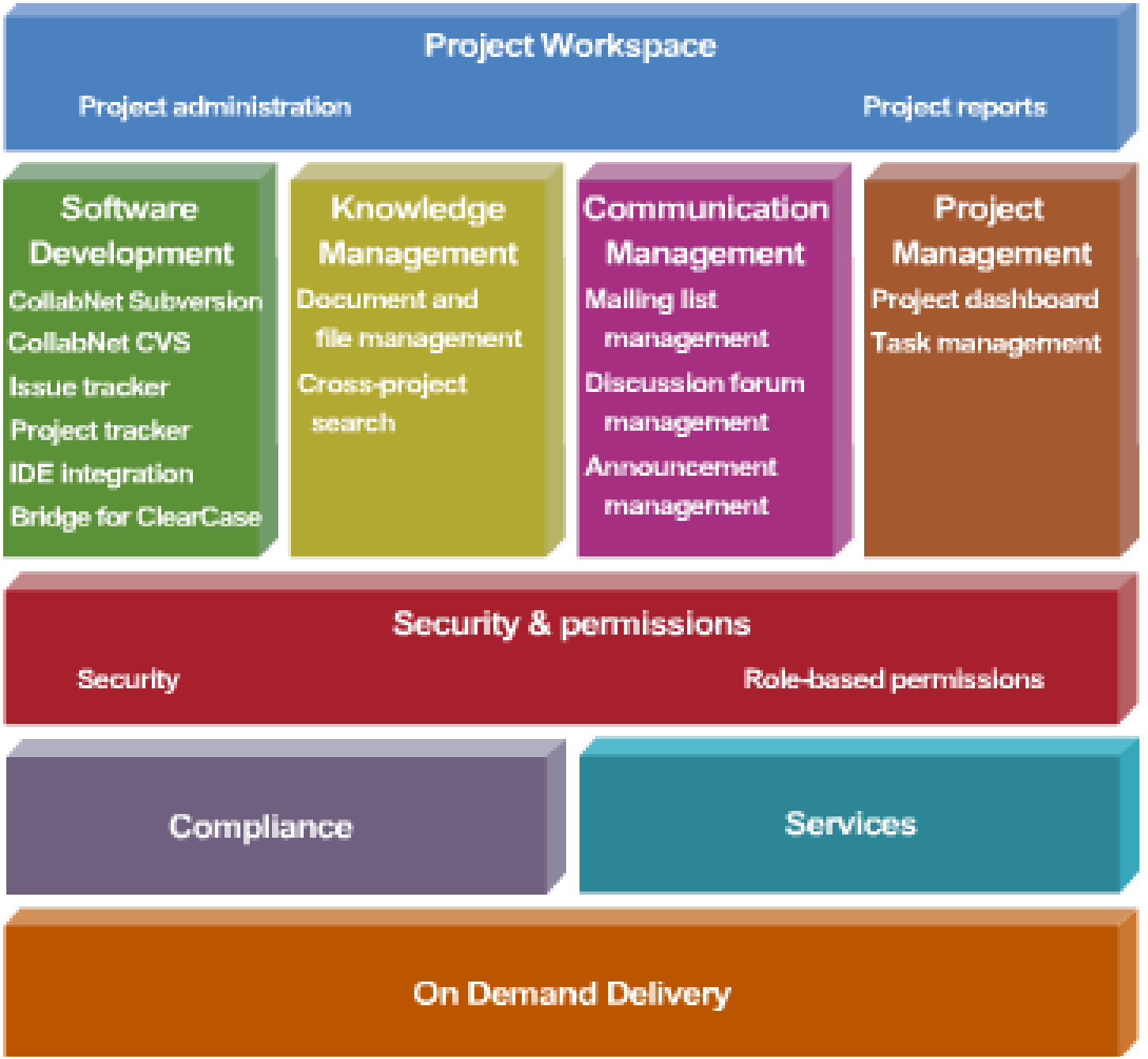
# CASE Integration

- Tools
  - Support individual process tasks such as design consistency checking, text editing, etc.
- Workbenches
  - Support a process phase such as specification or design. Normally include a number of integrated tools
- Environments
  - Support all or a substantial part of an entire software process. Normally include several integrated workbenches.

# Tools, Workbenches, Environments



Example: Collabnet Enterprise Edition for Open Source Software (OSS) Developers




http://www.collab.net/products/index.html

# The Bottom Line

- The Essence of Engineering is Pragmatism.
- It is a matter of choosing and picking the right tool for the right job.
- So, it is futile to try one universal method.

# Initial Student Team Meeting*: General Issues*

1. Set agenda and time limits.
2. Choose the team leader.
3. Get everyone's commitment to required time and work ethics
   - Define an expected average number of hours per week
   - Gather dates of planned absences
4. Take a realistic census of team skills
   - Common problem: inflated programming skill claims
5. Begin forming a vision of the application
6. Decide how team will collaborate and communicate
7. Take meeting minutes with concrete action items.

# Collaboration Plan

1. Meetings: Team will meet each Monday from ... to ... in …

   *Caveat: do not replace face-to-face meeting with remote meetings unless remote meetings are clearly effective.*

2. Meeting alternative: Team members should keep Fridays open from ... to ... in case an additional meeting is required.

3. Standards: Word processor, spreadsheet, compiler, ….

4. E-mail: Post e-mails?; require acknowledgement?

   *Caveat: e-mail is poor for intensive collaboration*

6. Collaboration: Tools for group collaboration and discussion –

   e.g. Slack, Yahoo Groups, Wiki tool, Google tools, …

7. Other tools: Microsoft Project (scheduling), Group calendar, …

# Communication Precepts

1. Listen to all with concentration
   - Don’t pre-judge
2. Give all team members a turn
   - See the value in every idea
3. Don’t make assumptions
   - Ask questions to clarify
4. When in doubt, talk.