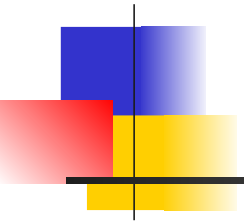
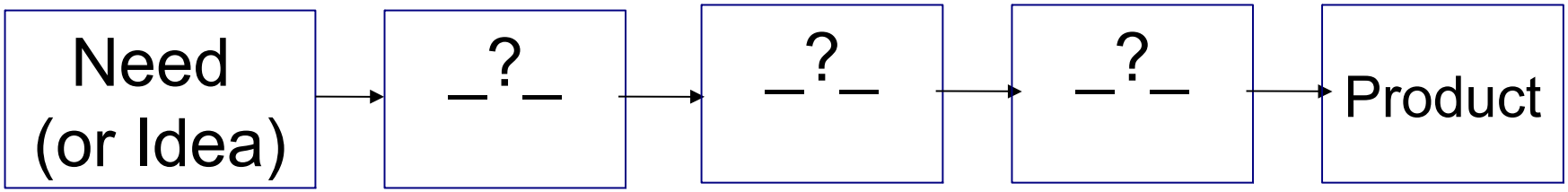


Intro to Software Processes

Goal of Software Development

Produce a software product that fulfills a need or realizes an idea.

What are the Steps?



What are the major steps or activities you would need to do?

List as many as you can.

Software Development Steps

1. Elicit Requirements
2. Vision of the solution
   - Develop a business case
3. Analysis (of requirements)
4. Specification
5. Project Planning
6. Design
   - architectural design
   - detail design
7. Implementation.
8. Test and review.
9. Integration & testing.

repeat 6-9

10. Acceptance testing.
11. Deployment.
12. Migration.
13. Maintenance.
14. Enhancement, improve.
15. End-of-life

Activities

Creating software involves

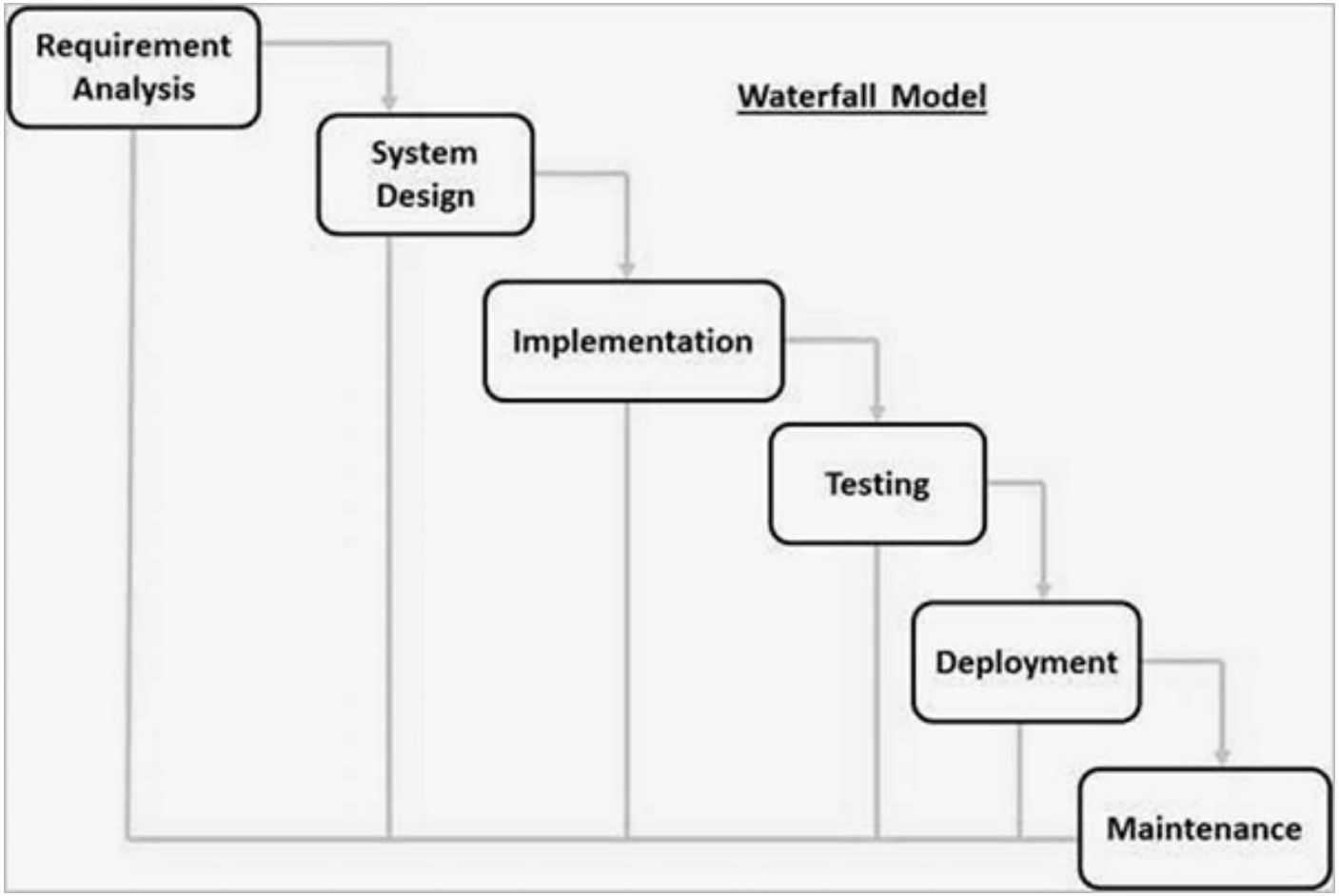
- elicit requirements
- specification
- high-level design
- prototyping (maybe)
- detailed (software) design
- construction & testing
- review
- validation
- documentation

Managing the project involves

- planning
- obtaining resources
- measuring
- tracking progress
- review
- analyze results, take action as needed

What if we do them in order?

Similar to a civil engineering project.



(There are really more steps than this.)

What Could Go Wrong?

Common Problems

What would be effect on project of each of these?

1. You miss some requirement(s).

2. You misunderstand some requirements, so the design is not what the customer wants.

3. The framework or design you chose can't handle the requirements.

4. Lots of bugs during development, discovered late during testing.

Common Project Outcomes (failures)

1. Project is late and over-budget.
2. Software does not do what the customer requires.
3. Excessive defects (bugs).

Causes of Project Failure

1. Poor communication.
2. Unrealistic schedule or budget. Forced deadlines.
3. Unclear requirements.
4. Excessive change in requirements.
5. Unwillingness to accept change.
6. Not monitoring *actual* project progress regularly.
7. Insufficient developer skills.

How to Avoid These Problems?

Frequently "check" progress to see if you are on the right track.

- Feedback
- Testing
- Compare actual versus planned progress
- Analyze results and take corrective action

Process

Process -

a series of actions to achieve a particular result

Software process - a sequence of actions for producing software.

Process

Process -

a series of actions to achieve a particular result

Software process - a sequence of actions for producing software.

Do You Have a Software Process?

What is your software process?

(discussion)

What did you do (activities, tasks) in...

- Exceed Camp project?
- Programming 2 project?

Do You Have a Software Process?

Yes!

Everyone who develops software uses a process.

"Never thought about it" ...

- Process is *implicit* or *informal*.

... "*its different for each project*"

- *ad hoc process*

Overview of some Process Models

"Model" of how software development should go.

A "model" is an abstraction of something else.

So, some details are missing and it may be imprecise.

Code and Fix

- The most common software development process
- *Ad hoc* (chaos). Little or no planning and design.

Code and Fix

1. think about the problem, write some ideas on paper
2. start coding
3. run it. Test what I just wrote and fix it.
4. add more features to implement.
    - modify the code
    - goto step 2.

Why "Code and Fix" is Inefficient

- No clear statement of what the result should be

- No design or "its in my head".

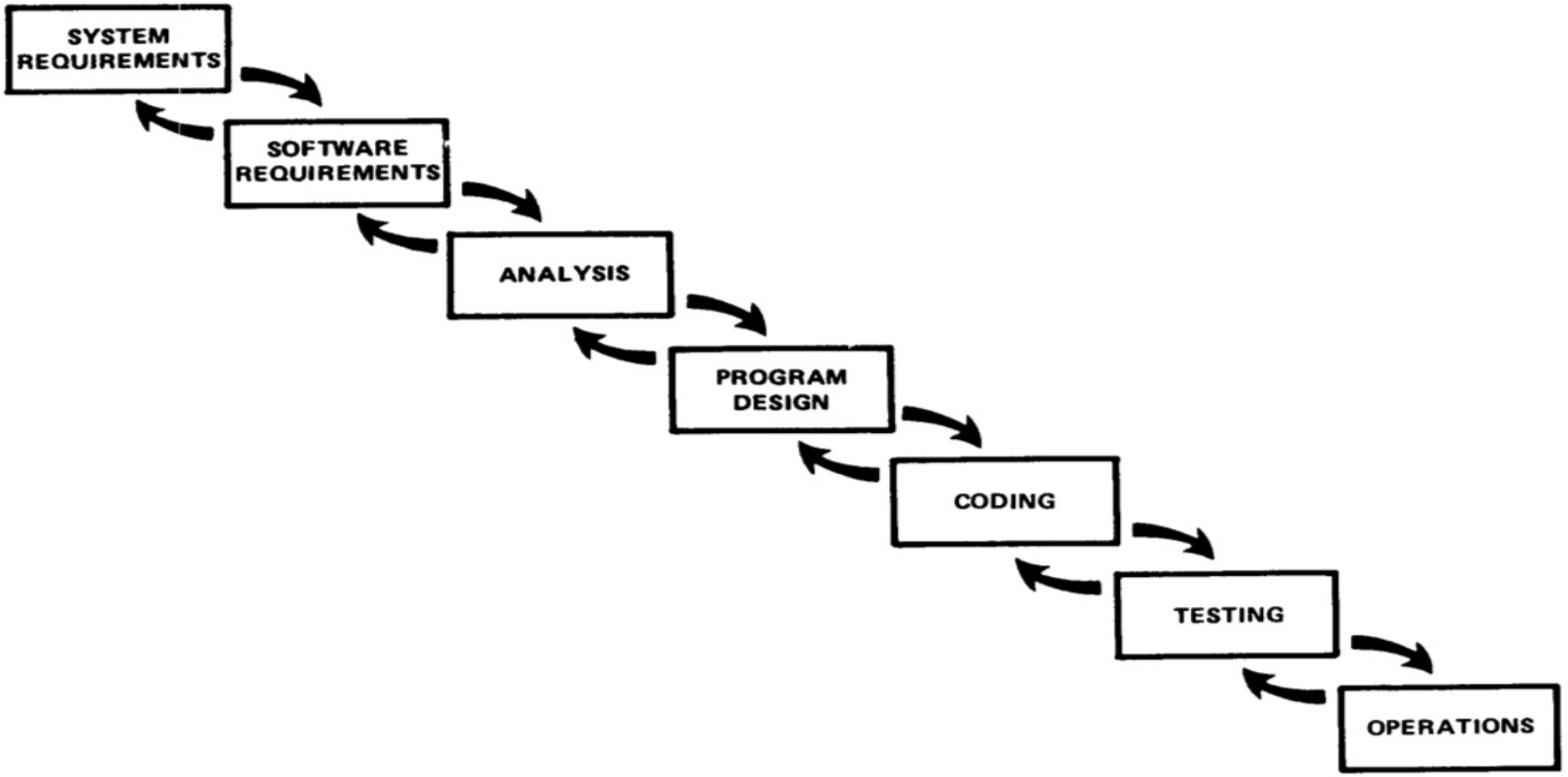
- Lots of rework.

Waterfall Model

Winston Royce, *Managing the Development of Large Software Systems* (1970)

Still commonly used.

Not as simple as usually portrayed.



Project Phases = Software Activities

In Waterfall, major activities are *phases* of project...

- Requirements phase
- Analysis phase
- Design phase
- Construction phase

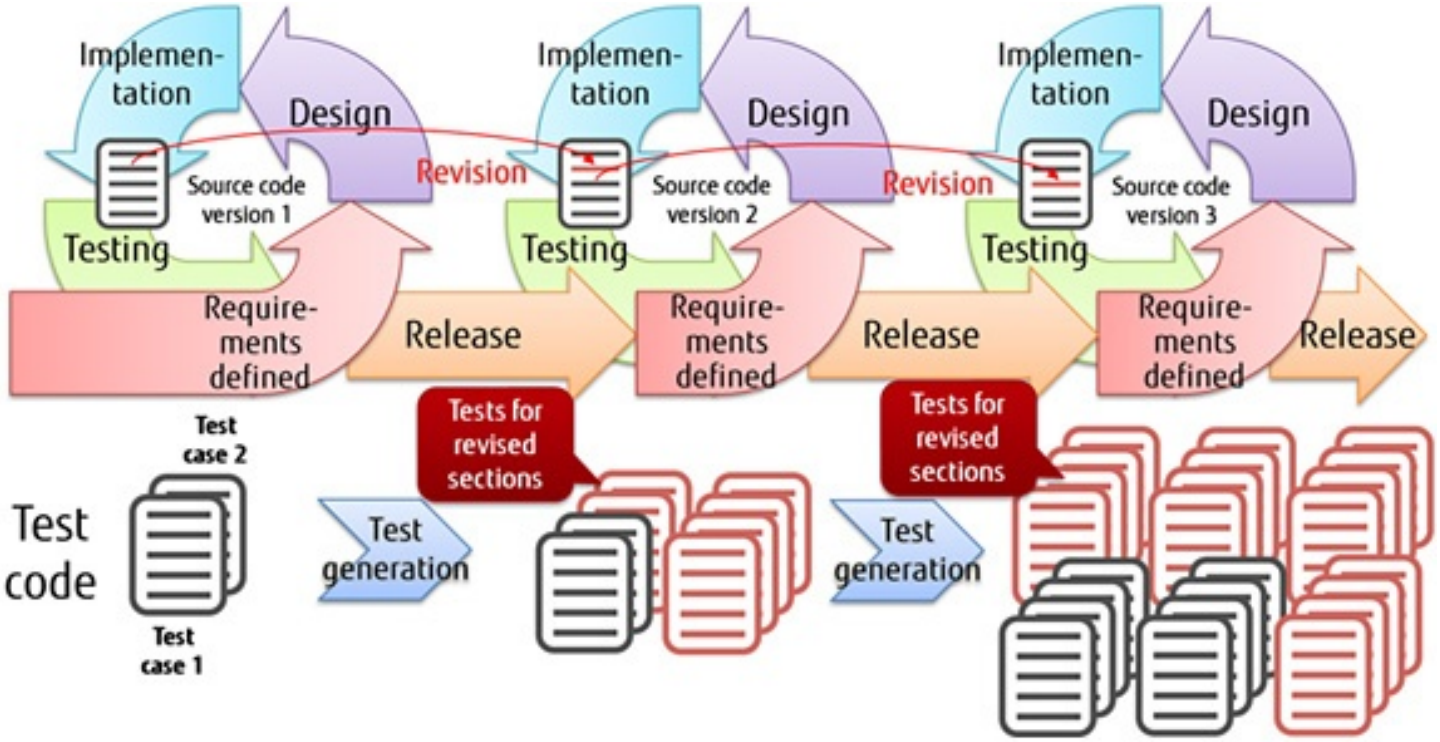
...

Iterative and Incremental

Let's not try to totally finish one activity (requirements, design) before starting others.

Iterate over each one.

Activities $\neq$ Phases



Iterative and Incremental

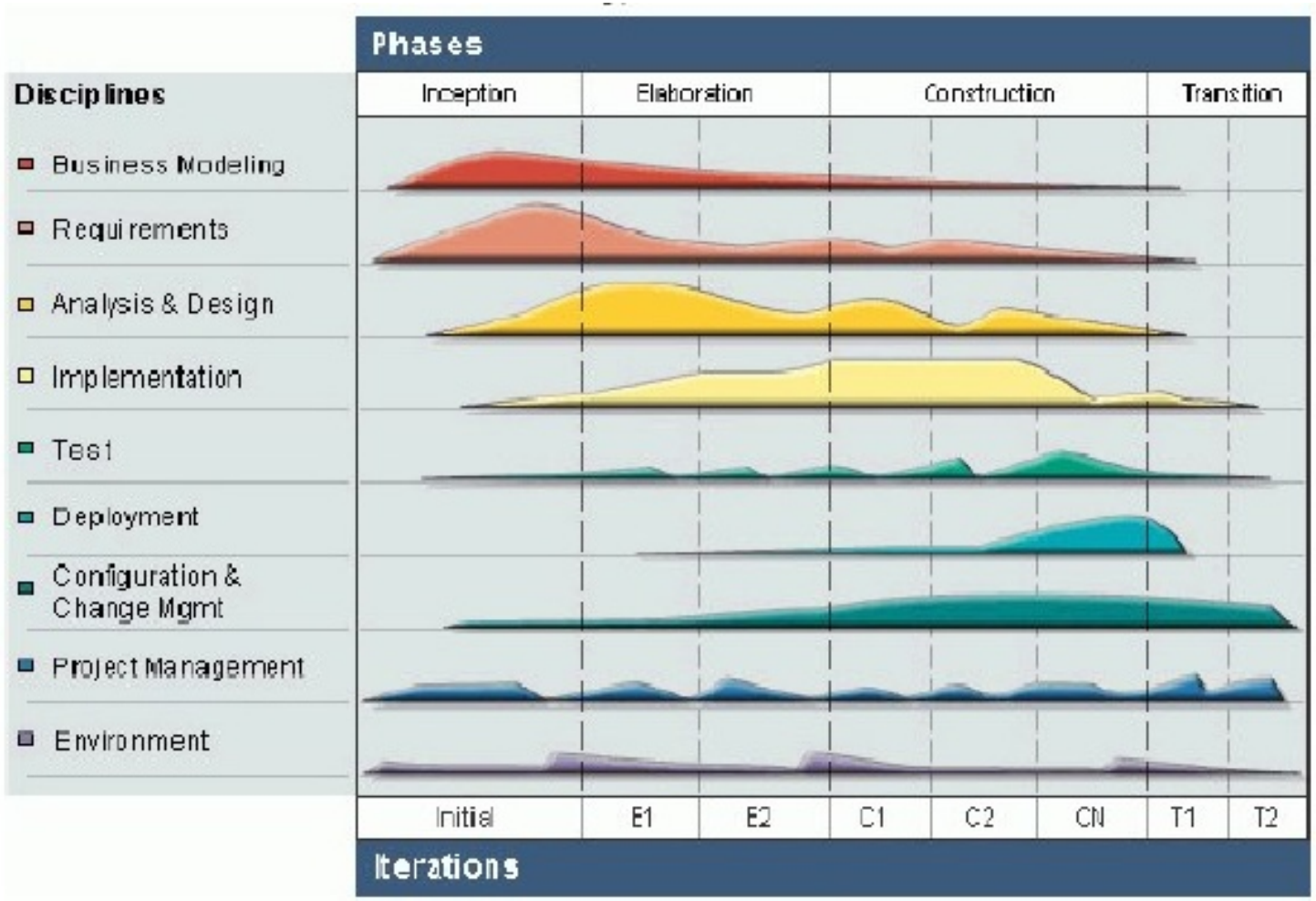
Incremental - project broken down into increments. Each increment adds some useful features and produces a usable result.

Iterative - add increments until satisfactory product is delivered.

UP Model

Workflows (disciplines) for different kinds of activities.

Phases: major divisions of project. May have many iterations.

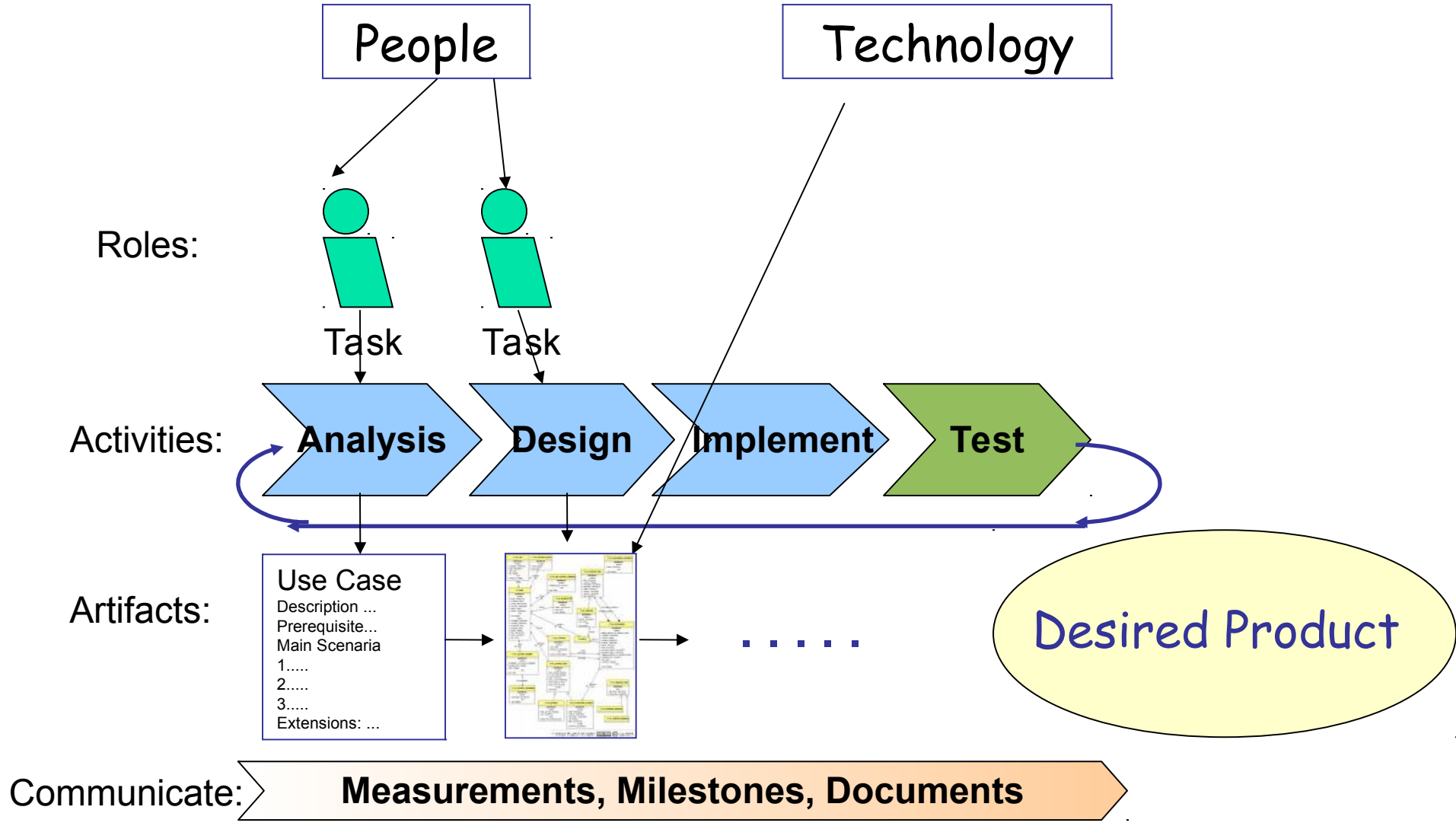


Unified Process

- Characteristics:
  - time-boxed iterations
  - plan based (but adapts to change)
  - "architecture centric"
  - address risks early
  - implement requirements based on priority or business value
  - accommodate change
- UP is a "framework" for a process -- you can tailor it as you like.

UP is covered in *Software Spec and Design* course.

Unified Process Model



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

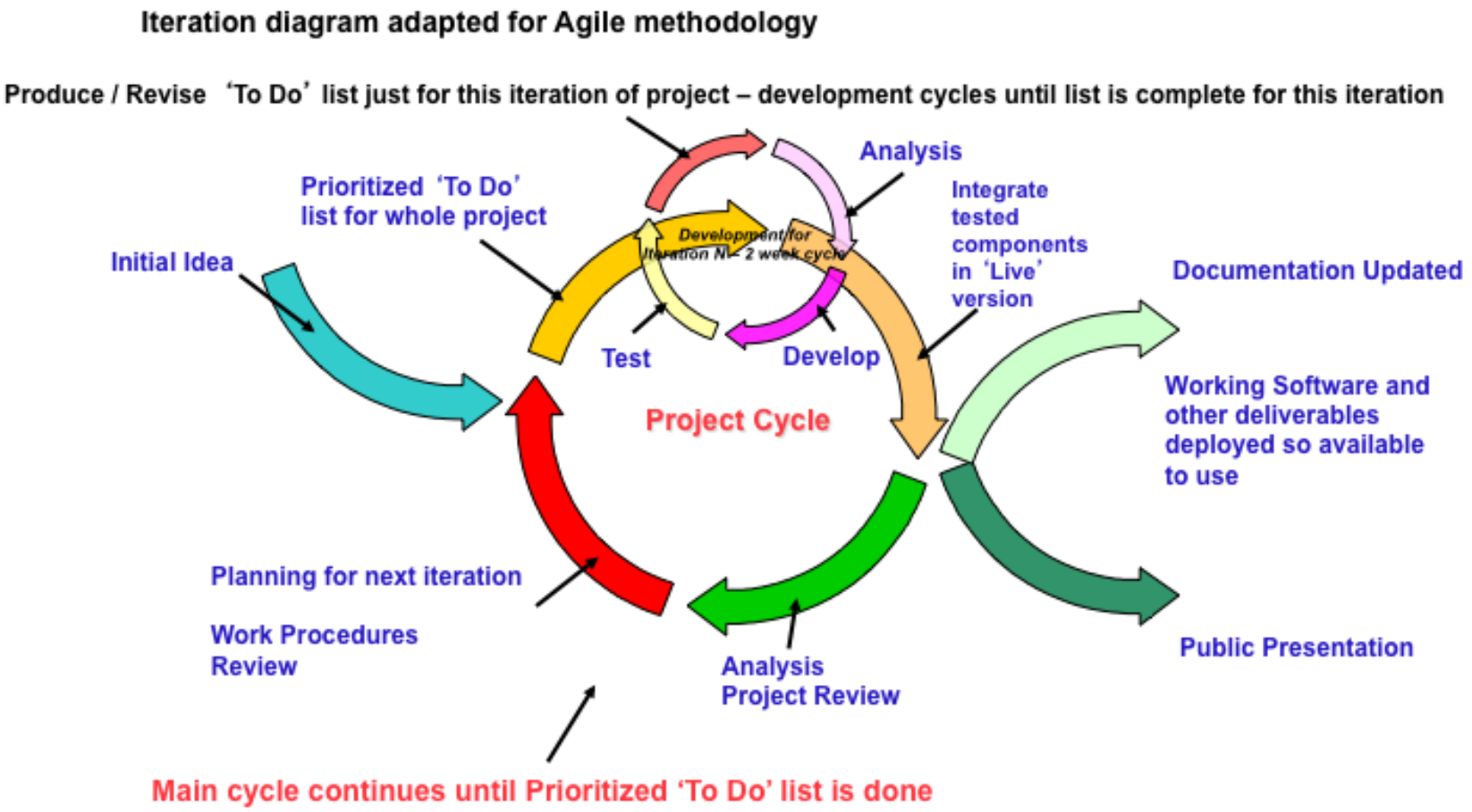
Customer collaboration over contract negotiation

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.

Agile Characteristics

See: *12 Agile Principles and Practices*



Agile Process Characteristics

- create customer "value" at each iteration
- welcome evolving requirements
- working software as primary measure of progress
- lack of up-front architecture design (YAGNI)
- simple design (XP: "*do the simplest thing that* ...")
- small, self-organizing teams at one site (preferred)
- frequent customer feedback
- shared understanding instead of *comprehensive* documents … but they *do* write documents

...

Some Agile Processes

- eXtreme Programming
  - Kent Beck: Chrysler
- Scrum - called "more a management technique"
  - iterative development in "Sprints"
  - daily stand-up meeting
  - no advise about requirements,
- Crystal
  - a family of methods to address different types of projects
- Synchronize and Stabilize (Microsoft process)

What About Individual Process?

Many software processes. *So what?*

... This is a course about individual process.

The End

(remain slides are unnecessary supplement)

Why a *Defined* Process?

This is to try to convince you that a defined software process is a good thing.

- Let's look at why software projects fail or succeed.

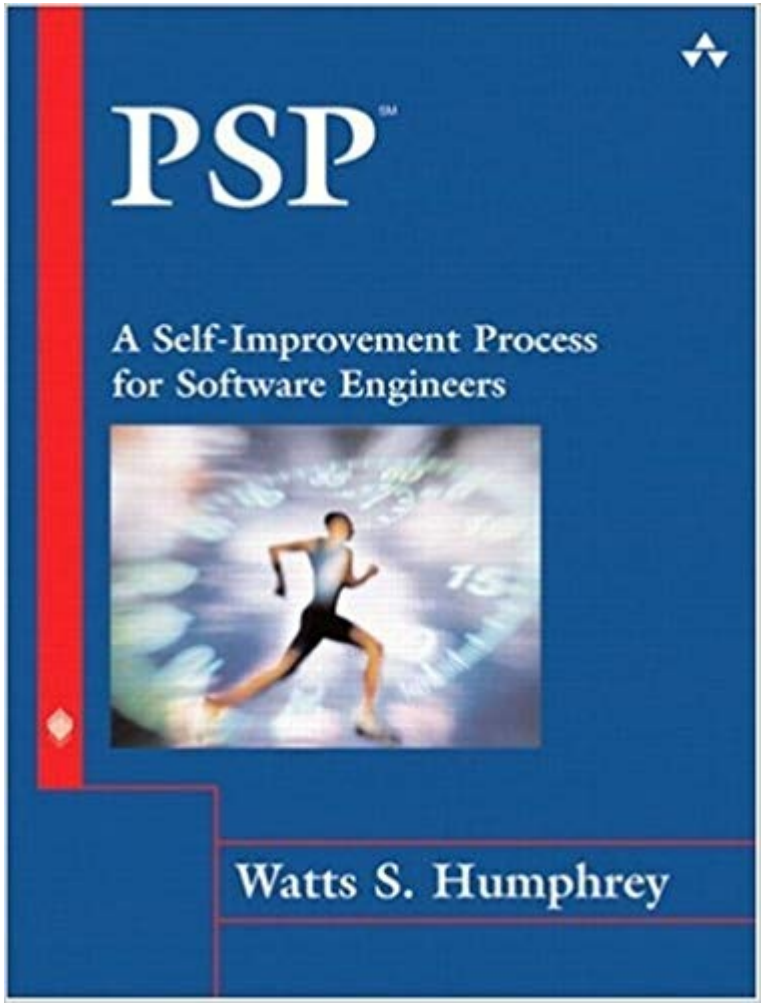
(*separate slides*)

Benefits of a *Defined* Process

- Saves time
  - less time spent on planning, estimates, decisions
- Predictable - enables planning & scheduling
- Repeatable
- Trackable - observed progress compared to plan
- Enables Quality Assurance
- Provides a basis for improvement
  - revise process based on experience

Ref: http://www.acm.org/crossroads/xrds6-4/software.html (2000)

Original Syllabus of the Course



Step-by-step practices to build a personal process for:

planning

defect tracking

estimation

measuring quality & efficiency

evaluation

process improvement

Personal Software Process (PSP)

Objective: provide a disciplined process for SEs to manage their own work

- improve estimation and planning skills
- reduce defects in their products
- manage their own schedule & work quality
- improve their own software process

PSP in an 8-part course

Clear instructions of what to do at each part.

At each step you follow a process and write some code.

You add to your "process" during the course.

PSP progress through levels

PSP0: [baseline] measure time you spend on planning, design, coding, test, and *post mortem* (retrospective)

PSP0.1: measure output LOC. Add a coding standard and process improvement proposal (PIP).

PSP 1.0: Estimate program size using level 0 data. Make a test plan.

PSP 1.1: Add planning. Estimate time from program size.

PSP 2.0: Add design & code review. Emphasis on defect removal and prevention.

PSP 2.1: Add design specification.

PSP 3: Apply an iterative process to PSP2.1.

PSP Tools and Support

PSP emphasizes use of scripts, forms, and checklists to guide the user. These are included in course.

A useful tool is Process Dashboard (Sourceforge).

- tool includes the PSP scripts and forms, and generates reports for you.
- performs time tracking. Automates some reporting.

Classic Mistakes: People

These practices are likely to fail

- Reliance on Heros (super-programmers)
- Poor work environment: noisy, distractions
- Adding people to a late project

"*adding developers to a late project makes it later*"

Classic Mistakes: Process

These practices are likely to fail

- Unrealistic schedule
- Poor planning
- Avoiding difficult decisions

Classic Mistakes: Product

Requirements often lead to failure...

- Vague or unrealistic requirements
- Feature creep, or *requirements churn*
- Unnecessary features ("Gold plating")
- Treating R&D project as a development project

Classic Mistakes: Technology

- Silver bullet thinking: expecting technology to provide improve productivity or make the problem easy
- Using unfamiliar or unstable technology

"*Let's use Lavarel because we want to learn it*"

- Lack of version control

Related:

- *Don't keep a "personal copy"* outside of project VC.

Problem of Teaching Software Process

1. We learn on *small, one-semester* projects.
2. Projects often succeed based on heroic effort or super-programmers.
3. Programs aren't deployed or supported.
4. We are still learning, so process seems awkward.

Problem:

our process doesn't scale to larger projects.

benefits of many practices aren't visible.

What is a Life Cycle Model

A *model* of the phases that software goes through

Serves as a guide for management of software development

Models omit some real-life complexity and detail

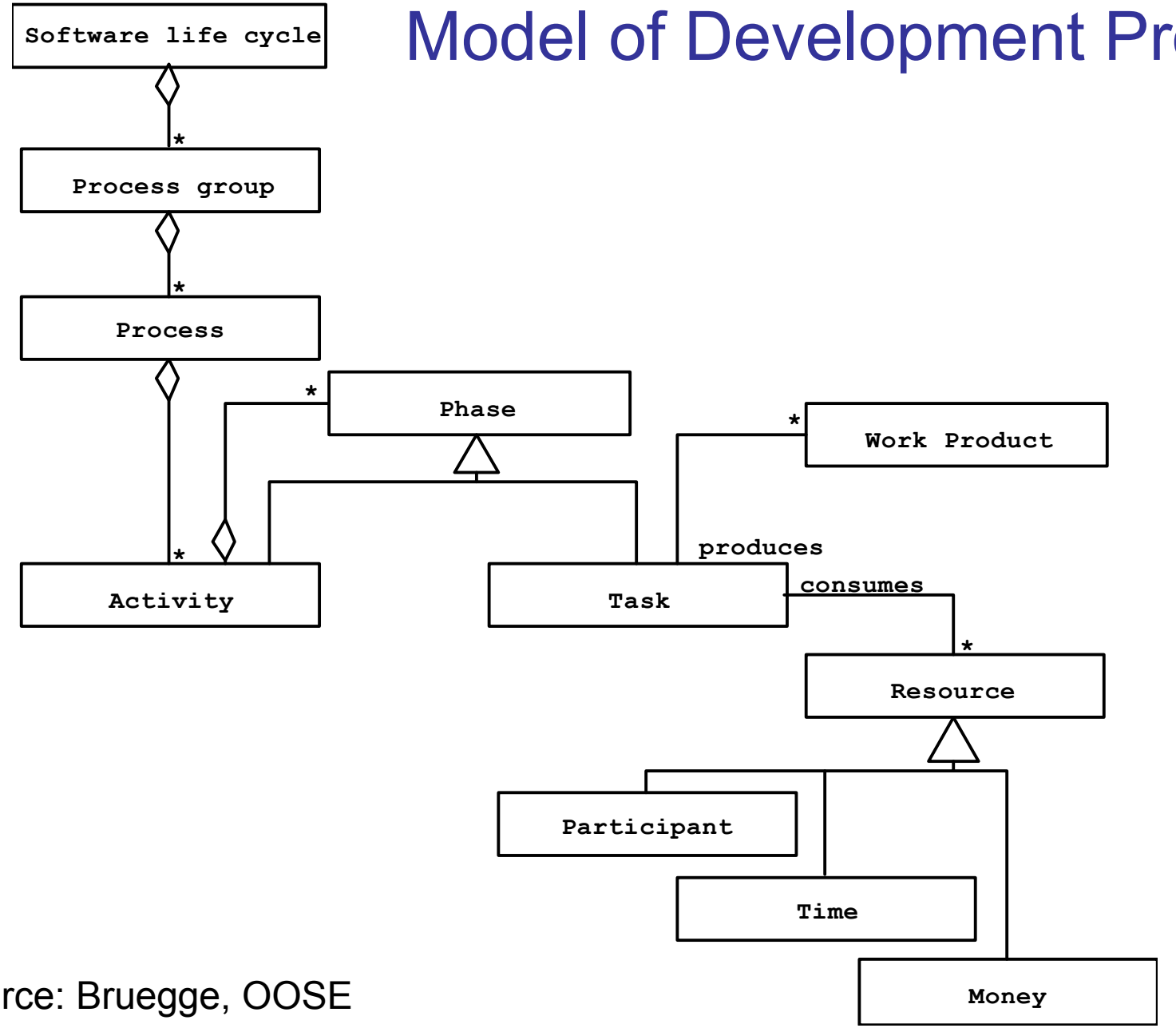
Typical Lifecycle Model

- Inception of idea
  - vision
  - business case
- Analysis
- Design
- Implementation
- Acceptance
- Deployment
  - transition
  - training & education

- Maintenance and Support
- End of Support
- Retirement
  - end of useful life
  - transition or archiving

GREEN = part of the *process, not life cycle*

Model of Development Process



source: Bruegge, OOSE