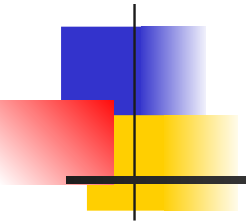
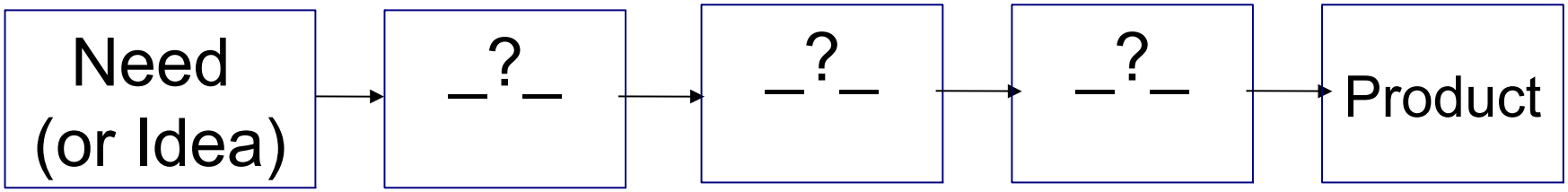


Intro to Software Processes

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:
   - 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, integration testing.

repeat 6-9

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

Activities in Software Development

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

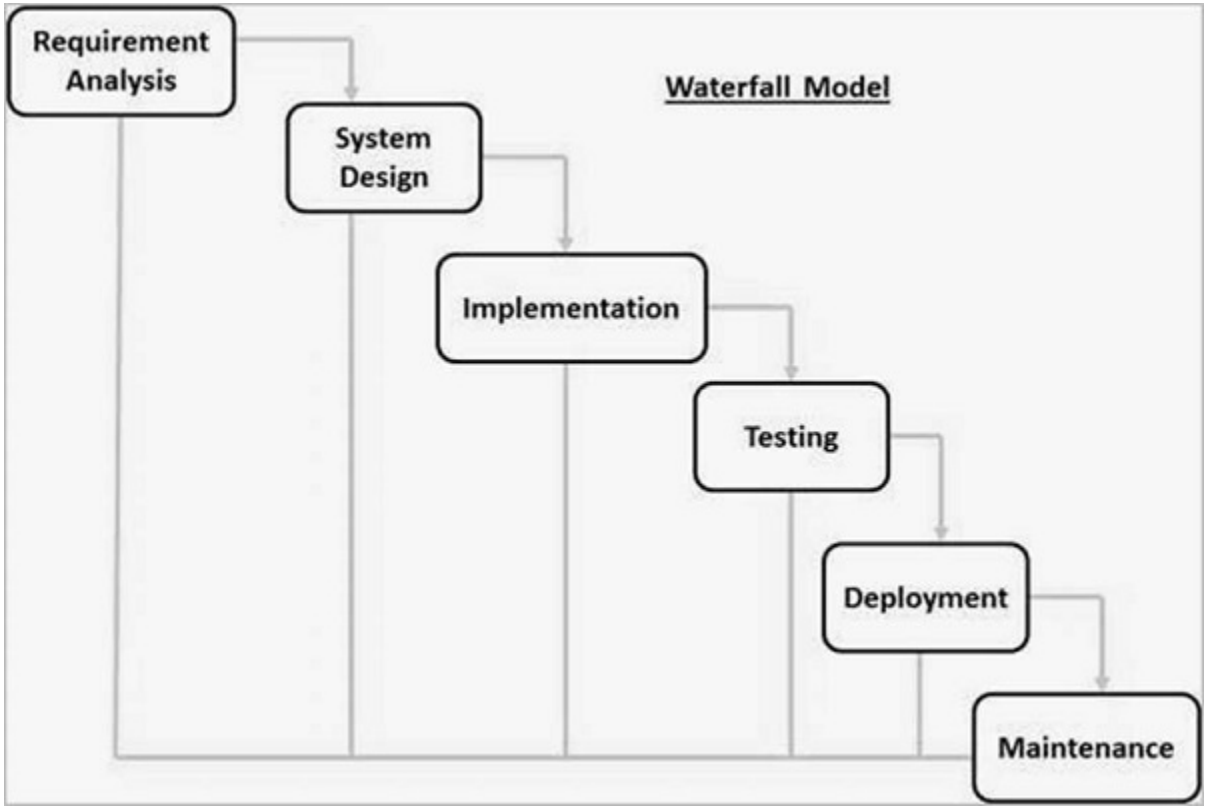
Managing the project involves

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

How about if we do them in order?

This is a logical way to proceed.

Similar to a civil engineering project.



(There are really more steps than this.)

What Could Go Wrong?

Consider these cases

What would be effect on project of each of these?

1. You fail to elicit some requirement(s).

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

3. The framework you chose for the software can't handle requirements.

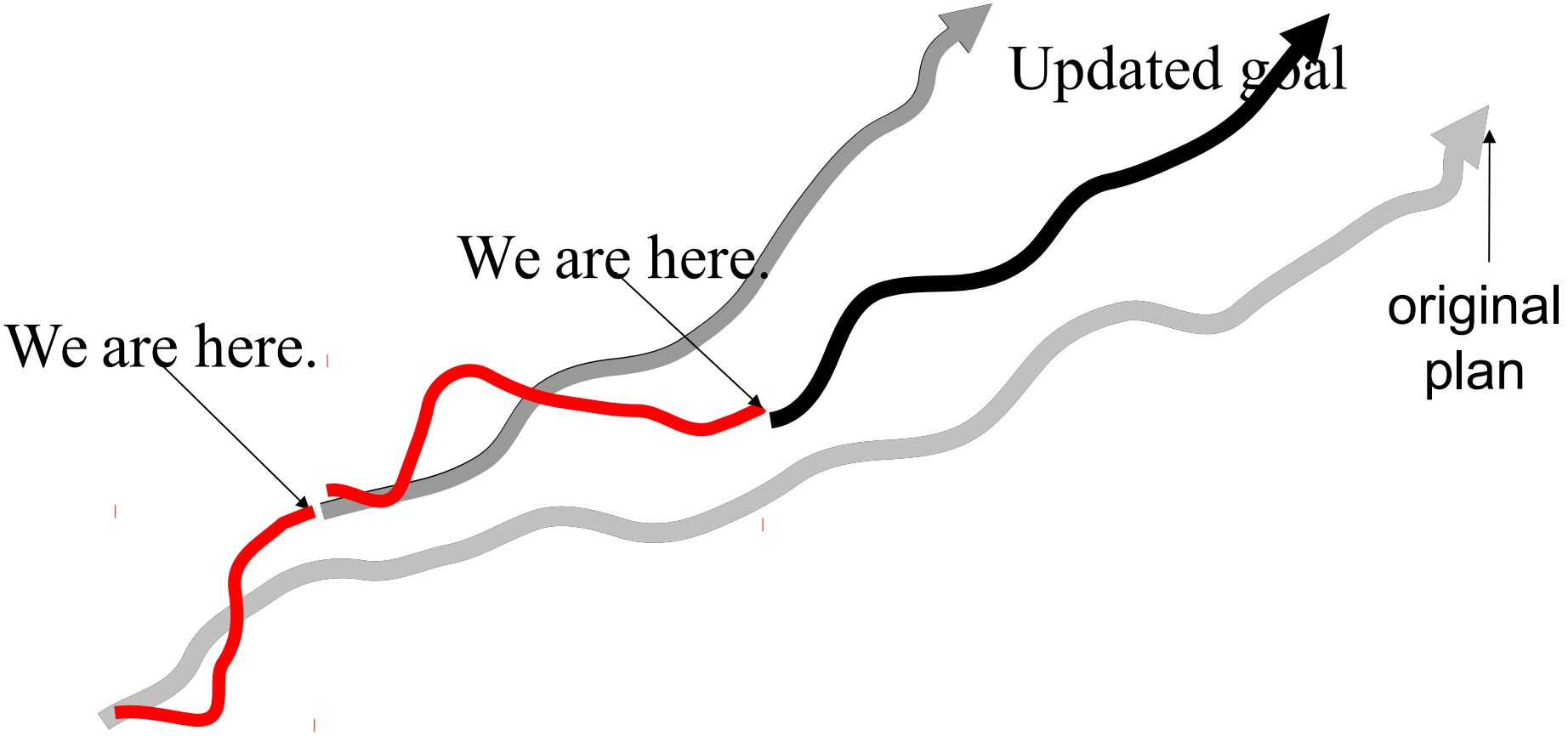
4. Lots of bugs during development, discovered late during test phase.

How to Avoid These Problems?

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

- Feedback
- Testing
- Review
- Analyze results and take corrective action

Using Feedback



Overview of some Process Models

"Model" of how software development (SDLC) 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, doodle ideas on paper
2. start coding
3. run it. Test what I just wrote and fix it.
4. as code grows, I realize that I need to change some parts.
    - 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".

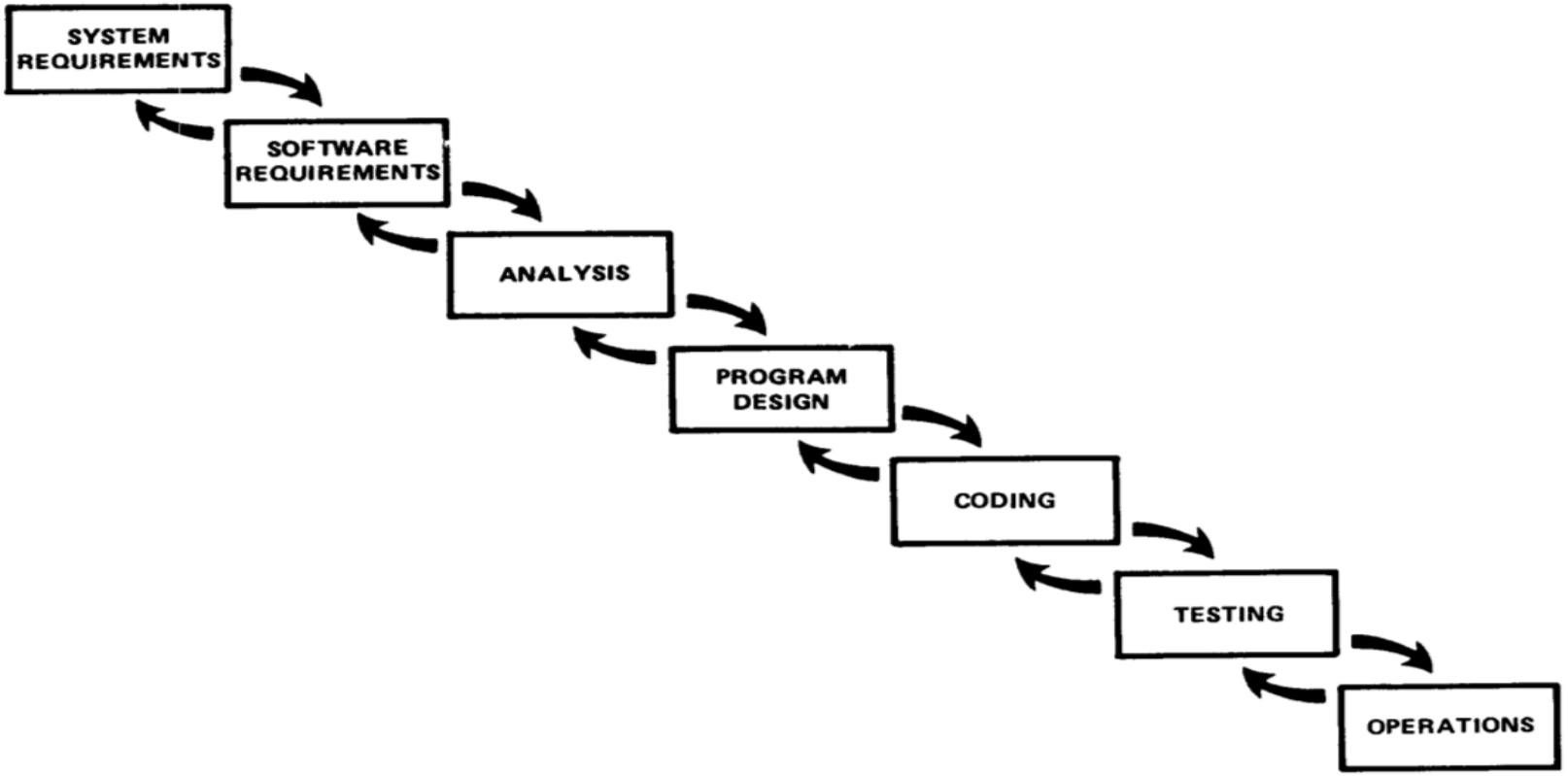
- Lots of rework.

Waterfall Model

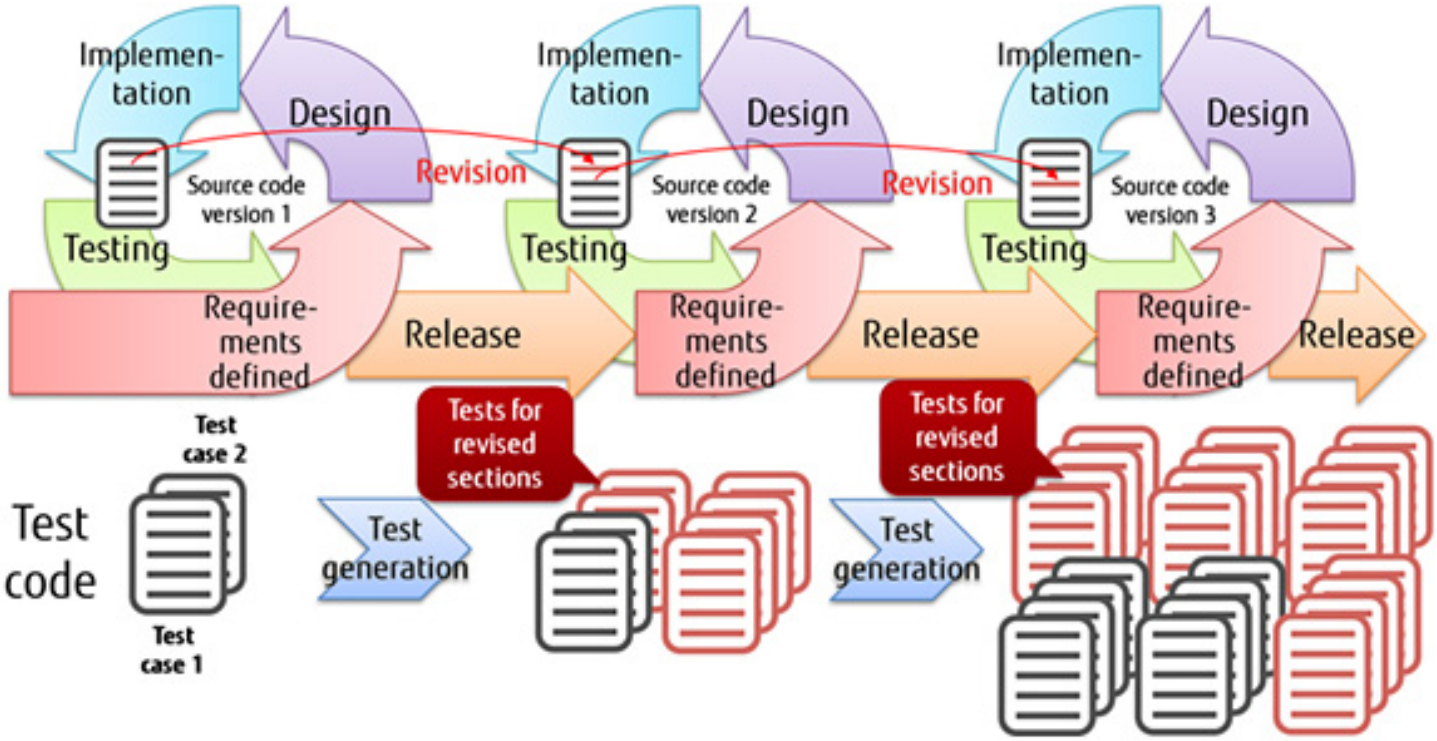
Winston Royce, *Managing the Development of Large Software Systems* (1970)

Still commonly used.

Not as simple as usually portrayed.



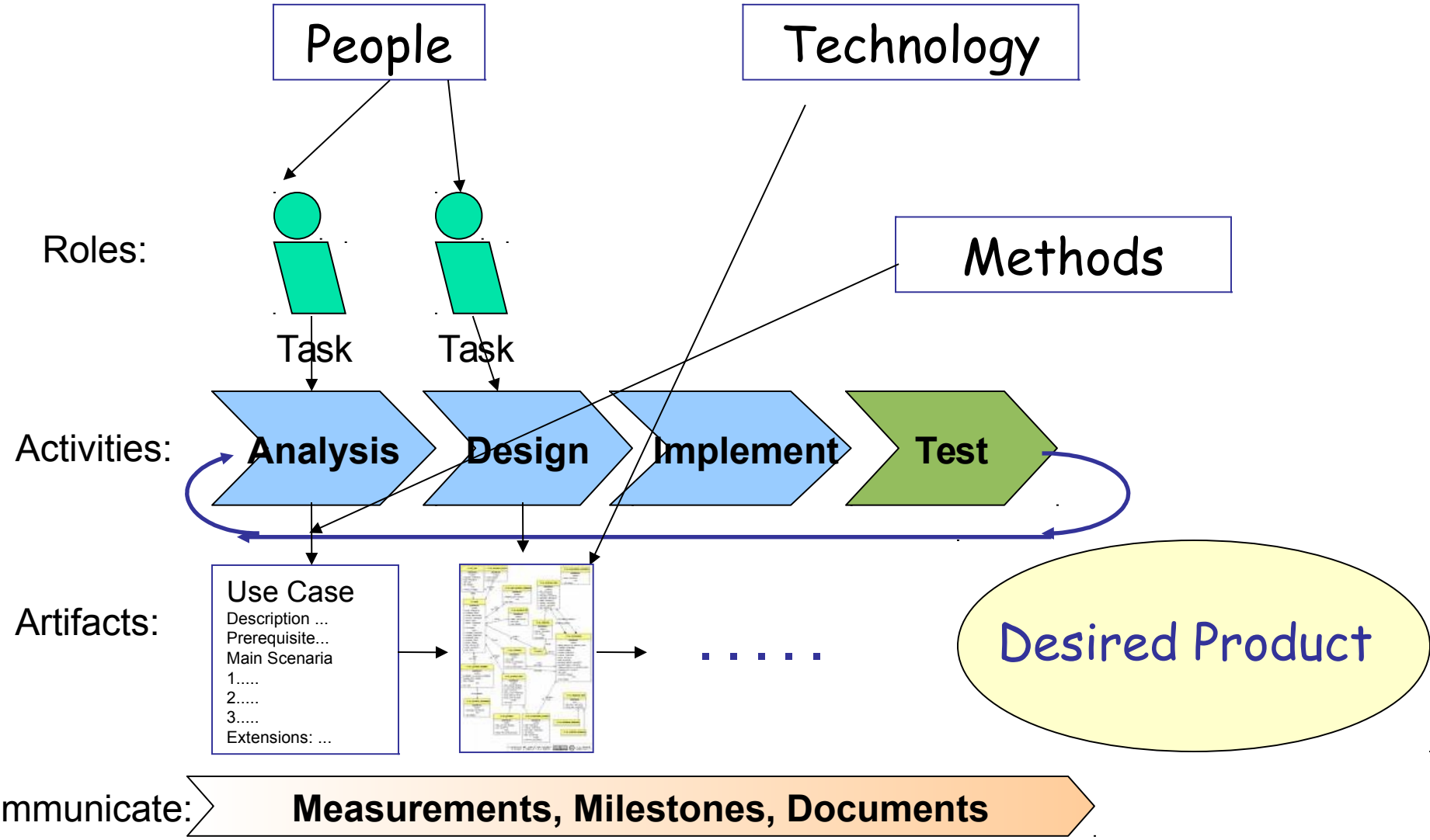
Iterative and Incremental



Incremental - project broken down into increments containing some useful features and/or other products.

Iterative - repeat it until satisfactory product is delivered.

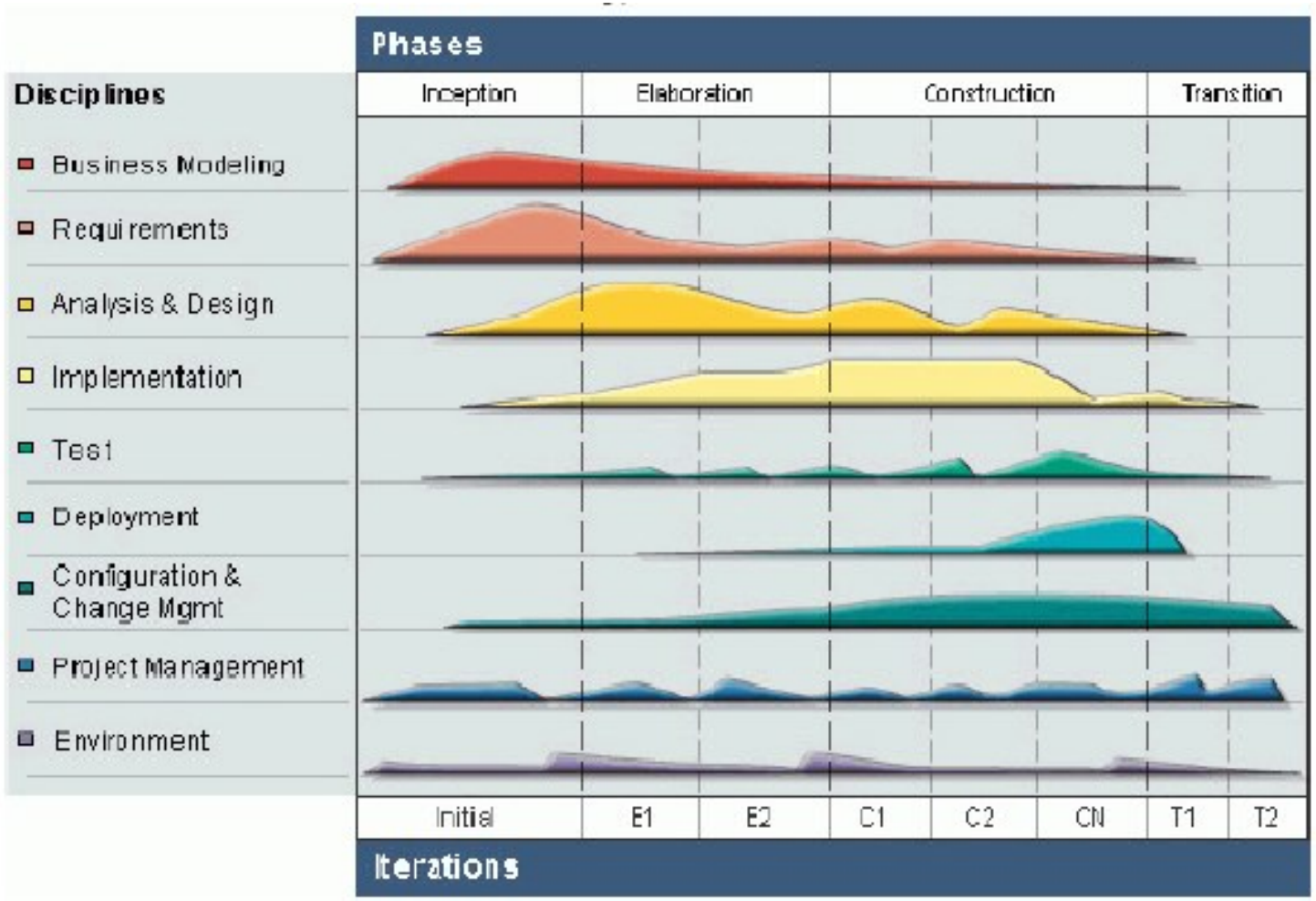
Unified Process Model



UP Disciplines and Workflow

Workflows related to different areas of concern.

Workflow contains activities and tasks.



Unified Process Characteristics

- Characteristics
  - time-boxed iterations
  - plan based
  - architecture centric
  - address risks early
  - implement requirements based on priority or business value
  - accommodate change
- Unified Software Development Process - a framework
  - Rational UP
  - Open-UP
  - Agile UP (Scott Amber)

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

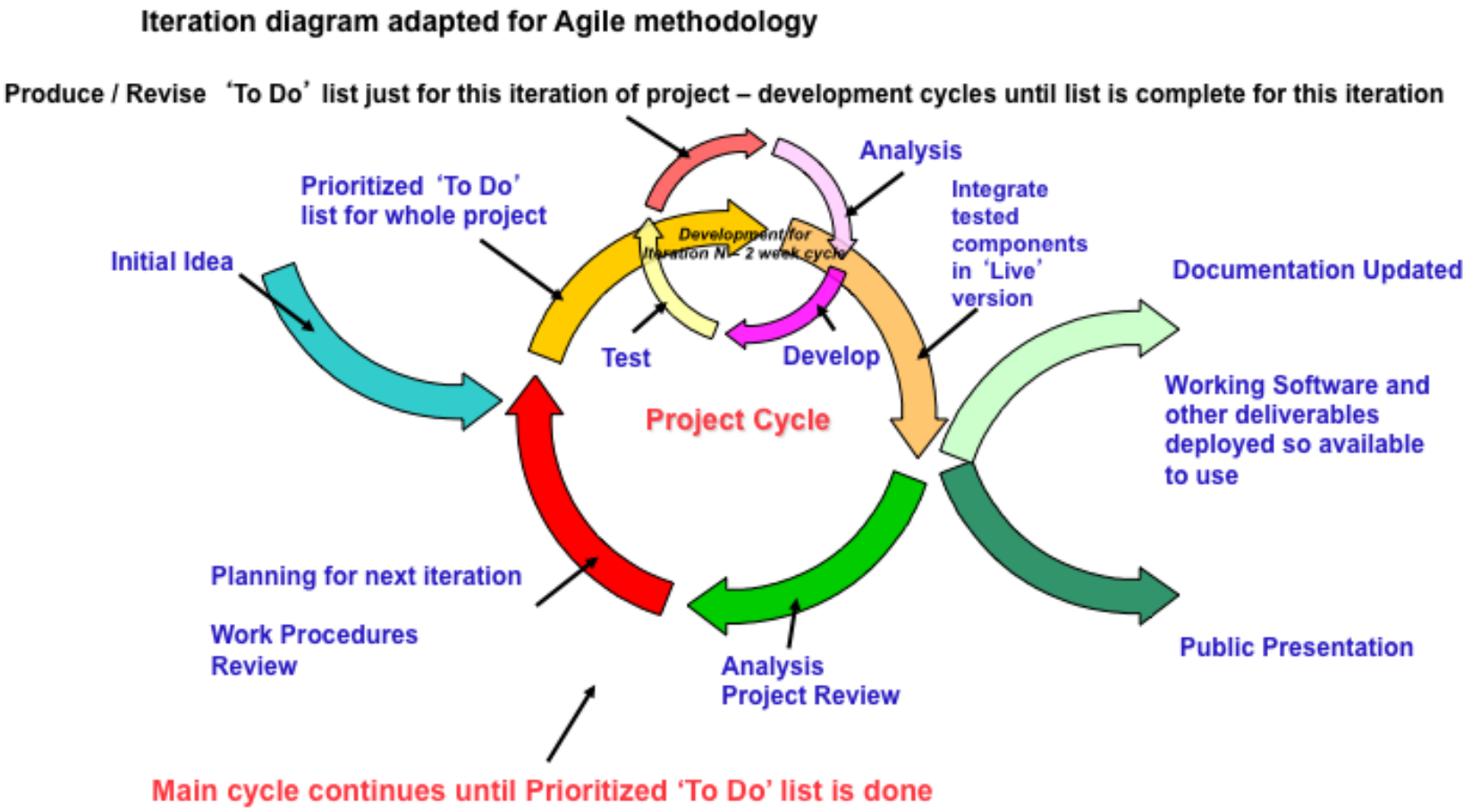
Customer collaboration over contract negotiation

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)
- simplicity of 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
  - small, democratic teams
- 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.

Can you define:

1. Goals for an I.S.P.?
2. Key practices and skills, based on the goals?

The End

(remain slides are unordered)

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.

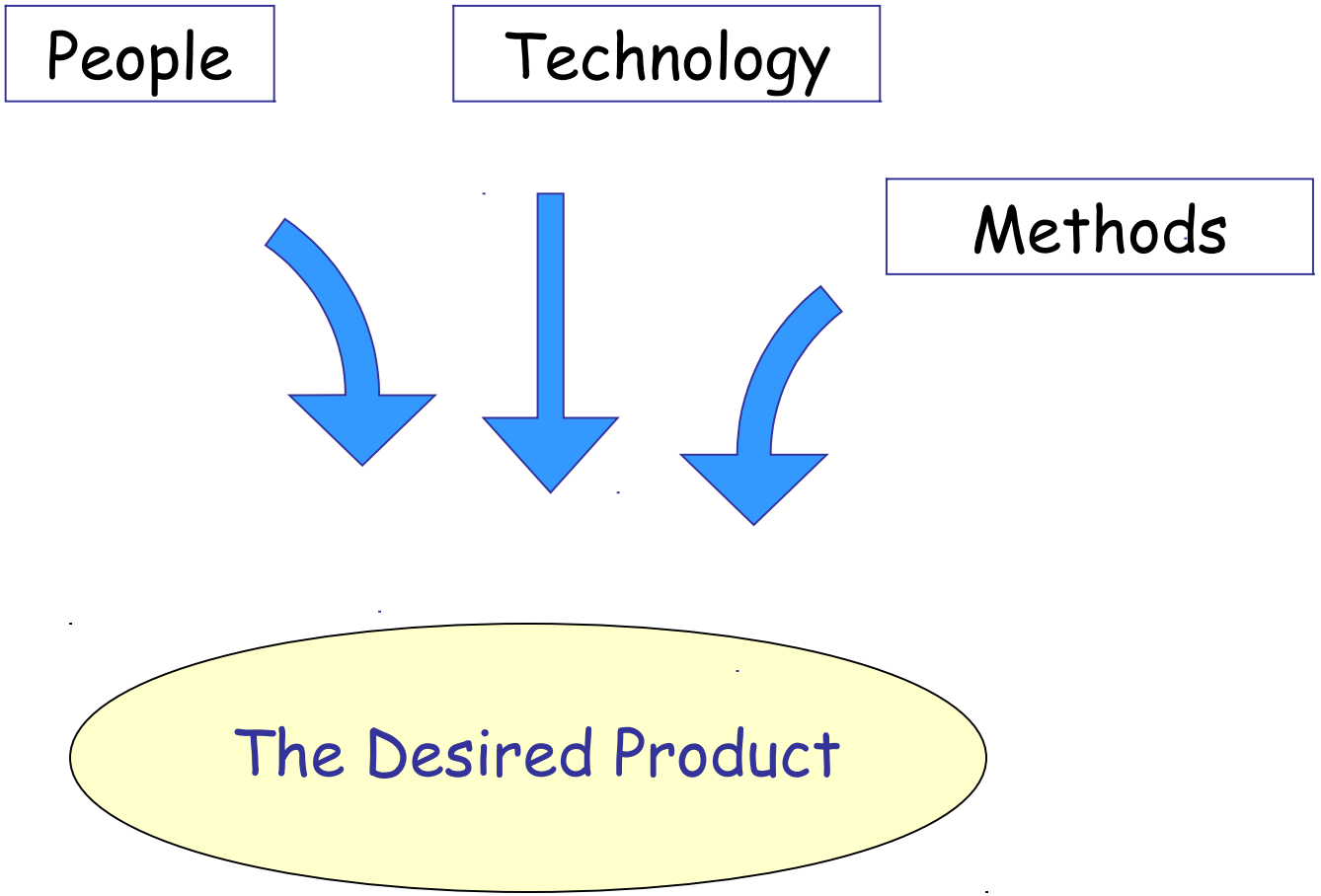
(*separate slides*)

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

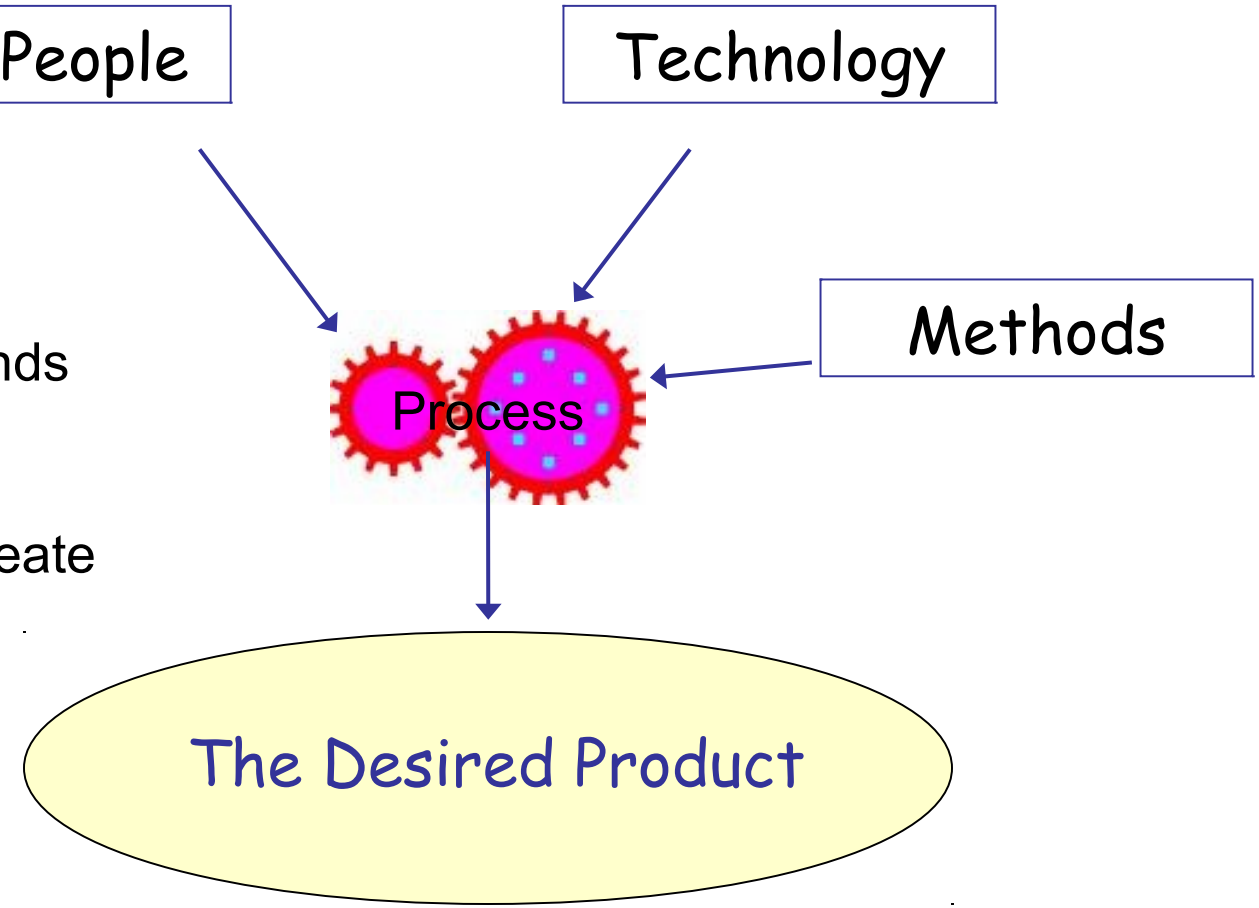
Ref: http://www.acm.org/crossroads/xrds6-4/software.html (2000)

The Role of Process



The Role of Process

"the glue that binds [guides] people, methods, and technology to create a product."



Name some software process models?

"Software process" can be …

Software development process

Software life cycle process

Other "Disciplined" Processes

- Team Software Process (TSP)
- Personal Software Process (PSP)
  - objective is to improve personal productivity through staged sequence of activities
  - measure and analyze: time, defects
  - improvement through reflection and planning

Personal Software Process (PSP)

Objective: provide a disciplined process for SEs to...

- improve estimation and planning skills
- reduce defects in their products
- be able to manage their own schedule & work quality

PSP Approach

Engineers progress through 6 levels:

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

PSP0.1: measure output LOC. Add 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

Humphrey emphasizes use of scripts, forms, and checklists to guide the user.

A useful tool is Process Dashboard (Sourceforge).

- this tool includes the PSP scripts and forms, and generates reports for you.

Classic Mistakes: People

These practices are predictably 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 predictably 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 development project

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*

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.

Approach of this Course

Emphasis on understanding the elements of a process.

- What is there purpose?
- How can they help your projects?
- How to do them practically?

Exercise 2

- What process practices or activities did you use in your ExceedVote project?
- Did you assign roles?

Some Terms

Software Life Cycle

Software Life Cycle Model

Software *Development* Life Cycle [Model]

Software Process

Software Process Model

aka: Defined Software Process

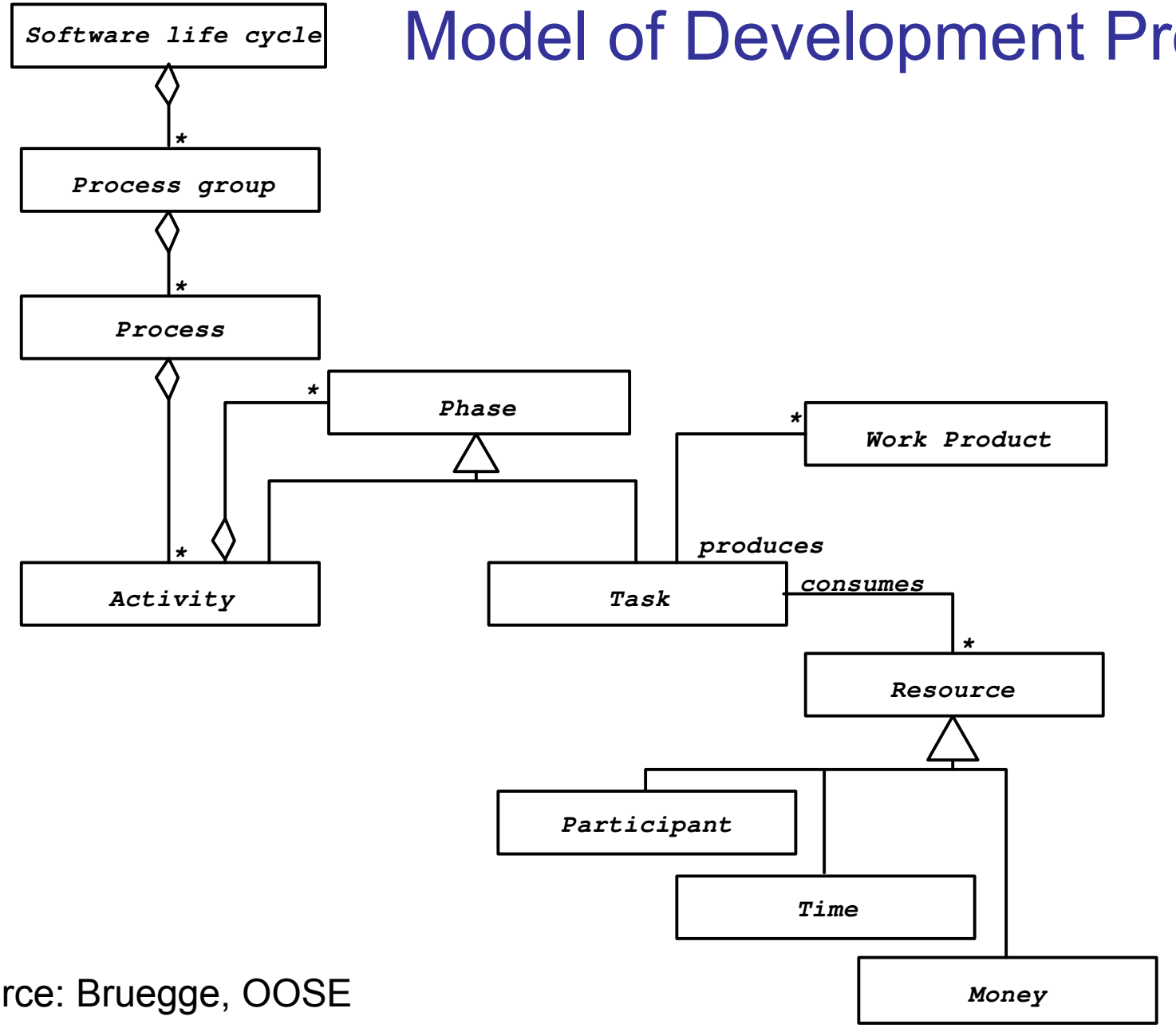
What is a Life Cycle Model

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

Model of Development Process



source: Bruegge, OOSE