COMP 354
Introduction to Software Engineering

Greg Butler

Computer Science and Software Engineering
Concordia University, Montreal, Canada

Office: EV 3.219 Email: gregb@cs.concordia.ca

Winter 2015

Course Web Site:
http://users.encs.concordia.ca/~gregb/home/comp354-w2015.html

Software Process — Introduction

The objective of software development is to efficiently and predictably deliver a software product that meets the needs of the community of its stakeholders.

A process is a set of ordered steps intended to reach an objective.

A software development process (or model) is an approach to **building**, **deploying** and **maintaining** software. The motivation behind defining a process for software development is to manage the size and complexity of software systems.

Software Process — Introduction

No single approach works for all projects (no silver bullet)

Various development paradigms exist

Common principle

Breakdown of a project into manageable components.

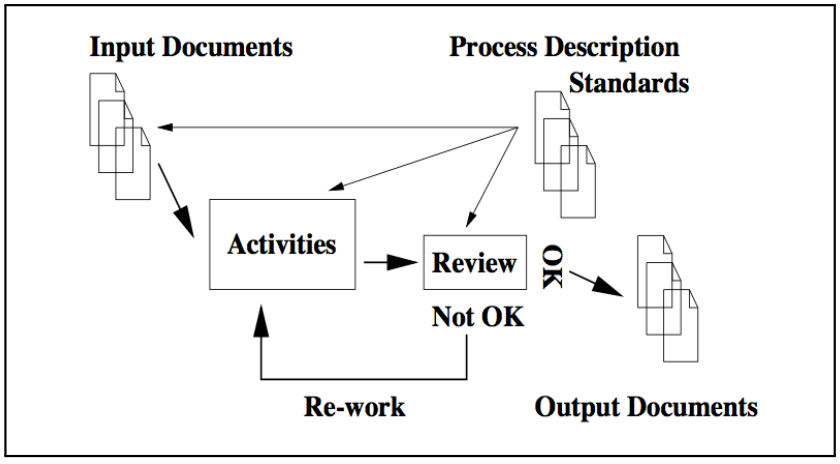
Breakdown by Activity: Linear Process Model

1 month requirements
1 month analysis
2 month design
2 month coding & testing

Breakdown by Functionality: Iterative Process Model

1st iteration has 20% functionality
2nd iteration adds 30%
etc.

How to Think about SE Activity



Products/Deliverables

- what is the aim of the phase, that is, what does it aim to produce

Activities

- what steps are taken in the process of producing the deliverables

Audience

- who will use each deliverable
- how will they use each deliverable
- what information will they extract from each deliverable
- is it easy for them to find/understand this information

How to Review

- what steps can you take to ensure the quality of each deliverable
  - during the production of the deliverable
  - after the production of the deliverable

Typical Activities in Software Development

Requirements

Design

Specification & Modeling

Implementation

Quality Assurance

System Deployment

Maintenance

Documentation

All of these activities must be managed, that involves ...

planning, coordination, scheduling, cost estimation, measurement, reviewing

Requirements Activity

Concerned with ...

Understanding the problem
What do users want?
What do users need?

Describe in user-oriented language

why?

Requirements form contract between the client and the developer

Design Activity

Concerned with ...

How will the software work?
How will users interact with the system?
How will data be stored and managed?
What parts (if any) should be done by hardware?
What components (methods, classes, packages, etc.) will we need?
How will the components be connected?

Bridge between requirements and implementation

Design must meet the requirements
Design must be technically feasible

Specification & Modeling Activity

Precise description of ...

What each component does
How components are structured (static models)
How each component communicates with other components (dynamic models)

Allows many people to work cooperatively on a large project

Implementation Activity

Write the code

Or ...

In some cases, coding may not even be necessary, because suitable code can be found in libraries or COTS (commercial, off-the-shelf software).

Quality Assurance Activity

Reviews

Inspections

Tests

unit tests ensures that components work independently
integration tests ensure that components work together
system tests ensure that the whole system works correctly

System Deployment Activity

Move the software to client sites

Upgrade from previous versions

Populate databases, etc

Ensure that it works on site

Single deployment (customized software)

multiple deployment (shrink-wrap software)

Maintenance Activity

Changes made to the software after delivery

correct errors
provide upgrades and enhancements
adapt to new hardware
improve quality without changing functionality

Up to 80% of total costs

Depends on the availability of up-to-date documentation

Documentation Activity

All tasks must be documented

Documentation is ...

very important

very difficult

Poor documentation may lead to failure

Linear vs. Iterative Development Process

Linear Development

Code and Fix
Waterfall development model

Iterative Development

Evolutionary development model
Spiral development model
Iterative and incremental development

- Rapid development model
- Agile development model
- Cleanroom development model

(UnifiedProcess)

Code and Fix

Used in the earliest days of software development

Two steps

1. Write some code
2. Fix the problems with the code

Advantage

No overhead as it does not entail any activities other than implementation.

Disadvantages

Does not adhere to solid engineering principles (such as planning, analysis, design, quality control)
Code deterioration
Often poor match to users needs

Waterfall development model

Winston R. Royce (1970)

Winston R. Royce (1970) *"Managing the Development of Large Software Systems"*

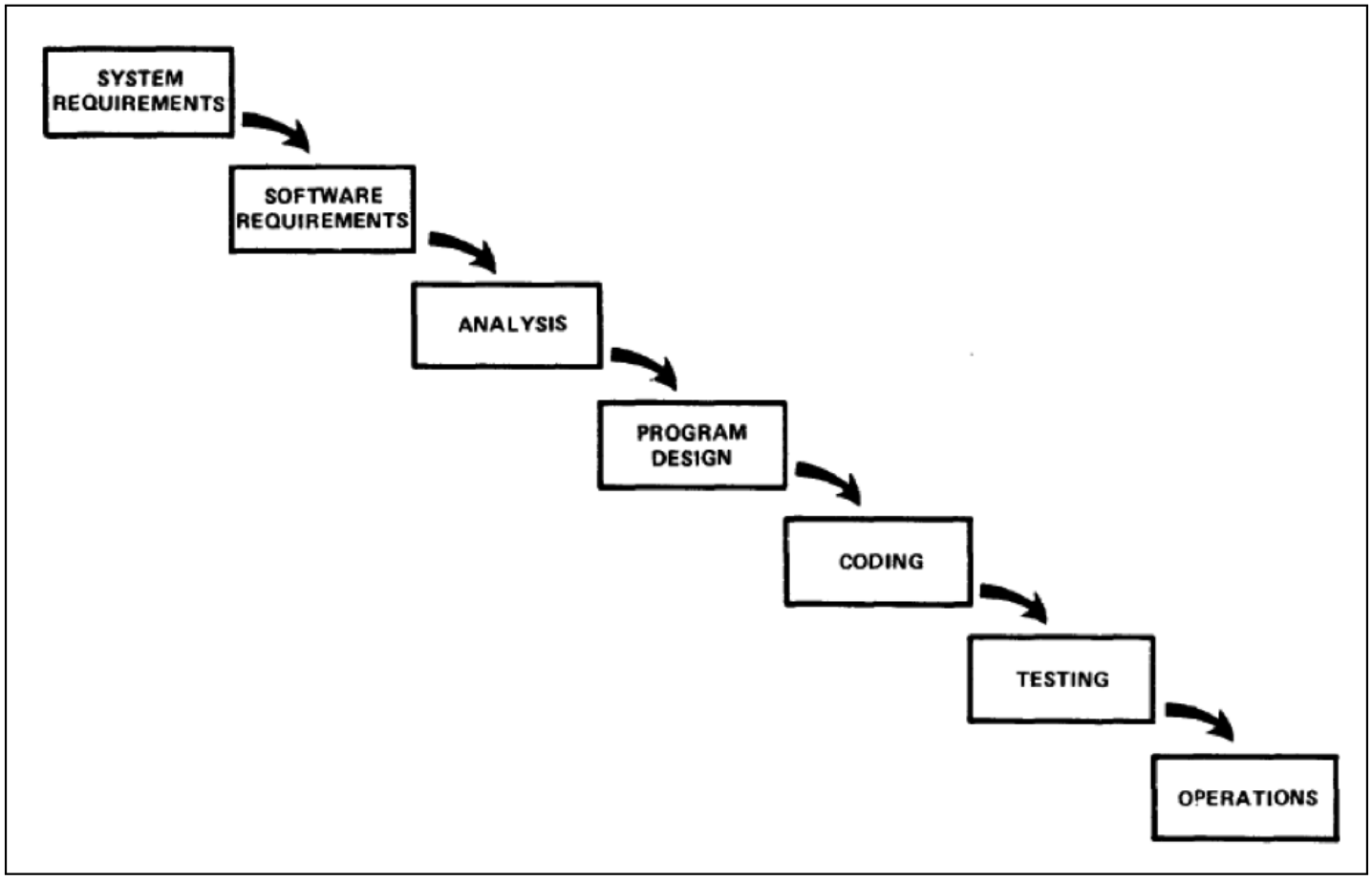
Linear

The model follows a linear path of development

Phases

Development is seen as flowing steadily downwards
(like a waterfall)
through the activities (called *"phases"*)

Waterfall development model



Waterfall development model

Origins in the manufacturing and construction industries

Typically requirements are very well defined
Product is delivered in a single version
Changes are either very costly or simply impossible.

Software system is developed as a whole

... and **not** in incremental releases

Proceeds from one phase to the next in a sequential manner

Move on to a new phase only ....

... once the preceding phase has been fully completed (and perfected)

Document-driven

Each phase produces a document needed by the subsequent phase

Waterfall development model — Strengths

Facilitates strong management control (plan, staff, track).

Documents provide a tangible record of accomplishment

Easy to understand and easy to use

Milestones are well-understood by the team (at the end of each phase)

Provides requirements stability

Applicable if both functional and nonfunctional requirements are well defined and the product is planned to be delivered in a single version.

Waterfall development model — Weaknesses

Limited stakeholder feedback

little opportunity for customer to preview the system.
Only feedback once the system is delivered (*"sign-off"*)

All requirements have to be known upfront

Often not feasible as requirements tend to change throughout development
Lack of flexibility of changing requirements

Risks are not explicitly addressed

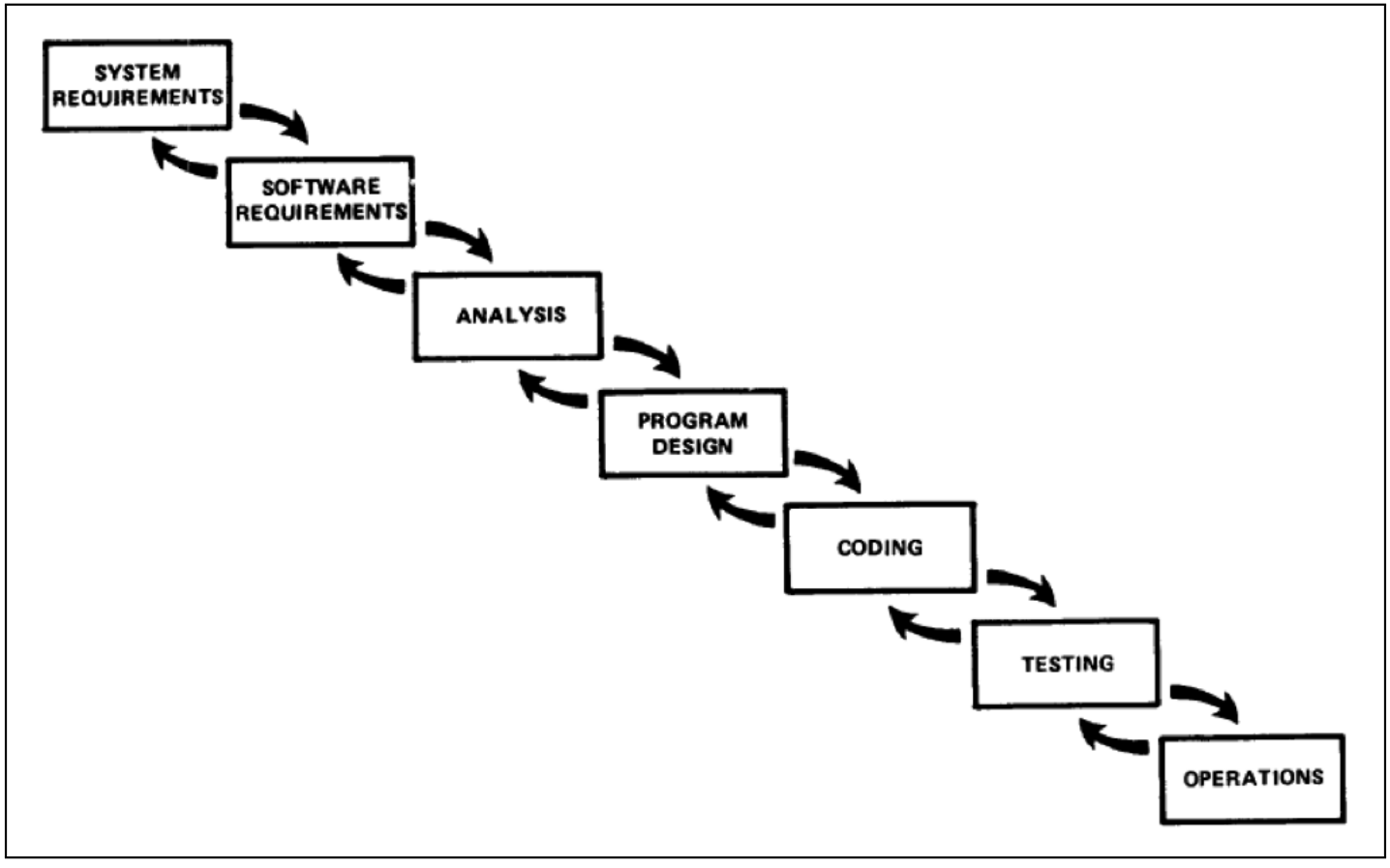
As such, they can be encountered late in the development

Can give a false impression of progress

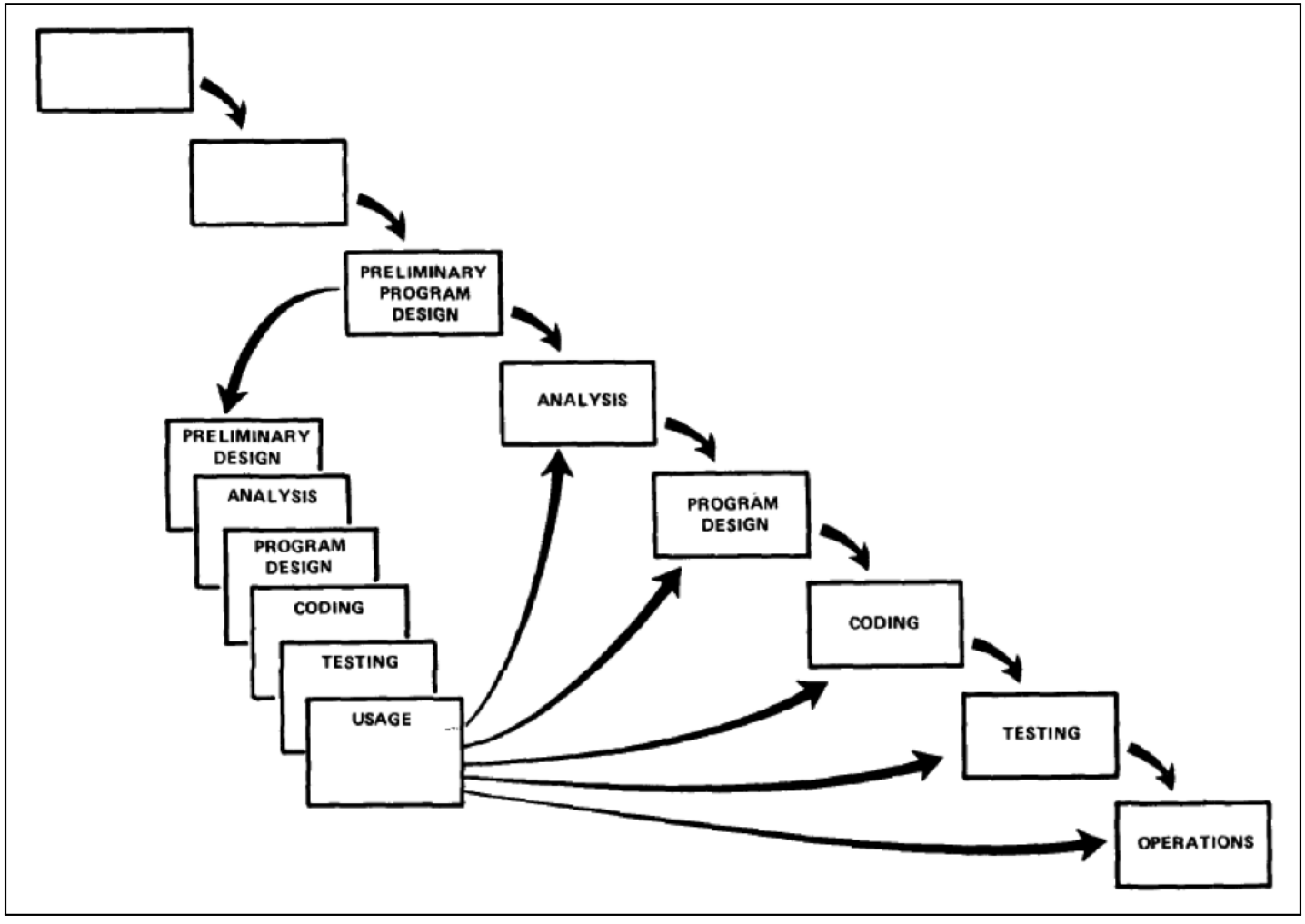
Documents say what people want to hear, not what is actually happening

Integration is one big-bang at the end

Waterfall development model — Add feedback



Waterfall development model — Protype then do again



Spiral development model

Boehm (1986)

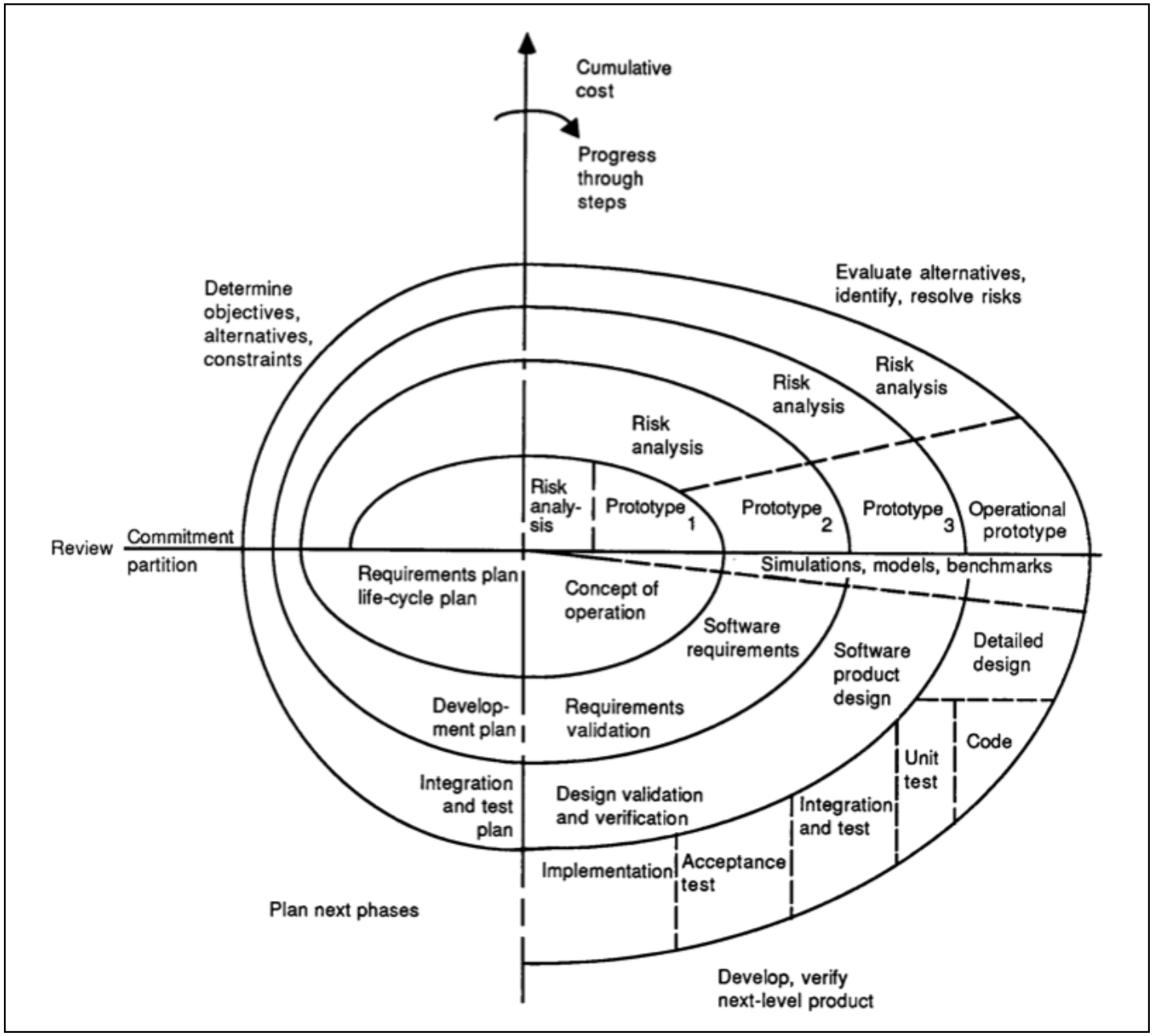
Barry Boehm (1986) *"A Spiral Model of Software Development and Enhancement"*

First to address the importance of iteration

Requirements are defined in as much detail as possible

Each iteration follows the waterfall model
(and is essentially a mini-project)

Spiral development model



Spiral development model

An iteration contains the following activities

1. Determine objectives; Specify constraints: Define the problem addressed by the current iteration.
2. Generate alternatives: Define solution(s). Identify and resolve risks. Risks are addressed in order of priority.
3. Develop and verify product: Work through requirements analysis, architecture, design, implementation and testing, producing a prototype.
4. Plan: Prepare for the next iteration.
5. Review

Spiral development model

Each iteration has a purpose

Examples:

- Feasibility
- Concept
- Top-level requirements specification

Risk-driven nature allows any appropriate mix of:

Specification-oriented, prototype-oriented, simulation-oriented or other approach of software development
Customization of budget, schedule, effort according to risk

Review ensures ...

... that all concerned parties are committed to the next phase

Spiral development model

The spiral model is based on **risks**.

A *risk* is a potentially adverse circumstance which may impair the development process or the quality of products.

In **risk management** we identify risks and respond to them before they endanger the whole project.

1. identify the risk
2. assess their likelihood and potential impact
3. address the risk by devising a plan to deal with it
4. eliminate the risk by implementing the plan

Spiral development model

Distance

The distance from the origin represents the cost accumulated by the project

The angular coordinate indicates the accomplished progress within each iteration

Development involves a number of iterations

each produces a prototype

(whose strengths and weaknesses are evaluated)

until eventually an operational prototype is delivered

Interim prototypes may or may not be deliverables

In its pure form, the model does not have phased releases.

Spiral development model

Strengths

Explicit integration of risk management
Applicable for development and maintenance projects
Early attention on options involving reuse of existing software

Weaknesses

Replying on risk-assessment expertise

- People-dependent
- Often expensive and lengthy

Evolutionary development model

Evolutionary prototype

Robust prototype which will eventually evolve into the operational system

Exploratory (throw-away) prototype

Inexpensive prototype which will eventually be discarded rather than becoming part of the final delivered software

Iterative and incremental development

Main idea

Build a system through repeated development cycles and in relative small portions.

As a result, the system is successively enlarged and refined though multiple iterations.

An iteration represents a complete development cycle and it includes its own treatment of all activities (normally in varying workloads);

Each activity produces a set of artifacts

Outcome of each iteration is tested and integrated into the executable system

Iterative and incremental development

Incremental development

The complete set of features and high-level requirements is elicited, analyzed and allocated to individual small scale projects before full-scale development begins

Allocation done ...

Based on criticality of requirements
Based on stakeholders preference

Executable of each iteration is a production subset of the final system

Made available to stakeholders in order to obtain feedback

Full functionality and the ability to test all requirements ...

... cannot be completed until all increments are complete by which time the system is ready for production.

Iterative and incremental development

Rapid development model

Emphasizes very short development cycles

Agile development model

Light-weight (agile) method (to be discussed on the next slides)

Cleanroom development model

Prevention of defects by deploying formal methods and stepwise refinement (functional decomposition)

Heavy-weight vs Light-weight Development

Heavy-weight

Characteristics

Pre-planned approach
Heavy regulations and control
Low degree of adaptation
Infrequent stakeholder feedback
Heavily documented
Large increments

Applicability

Large projects using large teams
Command controlled environments

Light-weight

Characteristics

Frequent stakeholder implication
Low degree of documentation
Complete teamwork and excellent communication
Small increments

Applicability

Smaller projects using small teams
Collaborative environments

Agile Development Model

Light-weight development model

Advocate short iterations and rapid customer feedback through:

Adding a customer representative to the agile team

Progress review at the end of each iteration by stakeholders and the customer representative

Follow the "Agile Manifesto" (next slides)

Several instances of the agile model exist, perhaps most popular is extreme programming (XP)

Agile Manifesto

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
Working software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.
Simplicity — the art of maximizing the amount of work not done — is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Extreme Programming

programming in pairs

extensive code review

unit testing of all code (actually test-driven development)

avoiding programming of features until they are actually needed

a flat management structure

simplicity and clarity in code

frequent communication with stakeholders.

The Unified Process

Created by Rational (IBM) in 1997

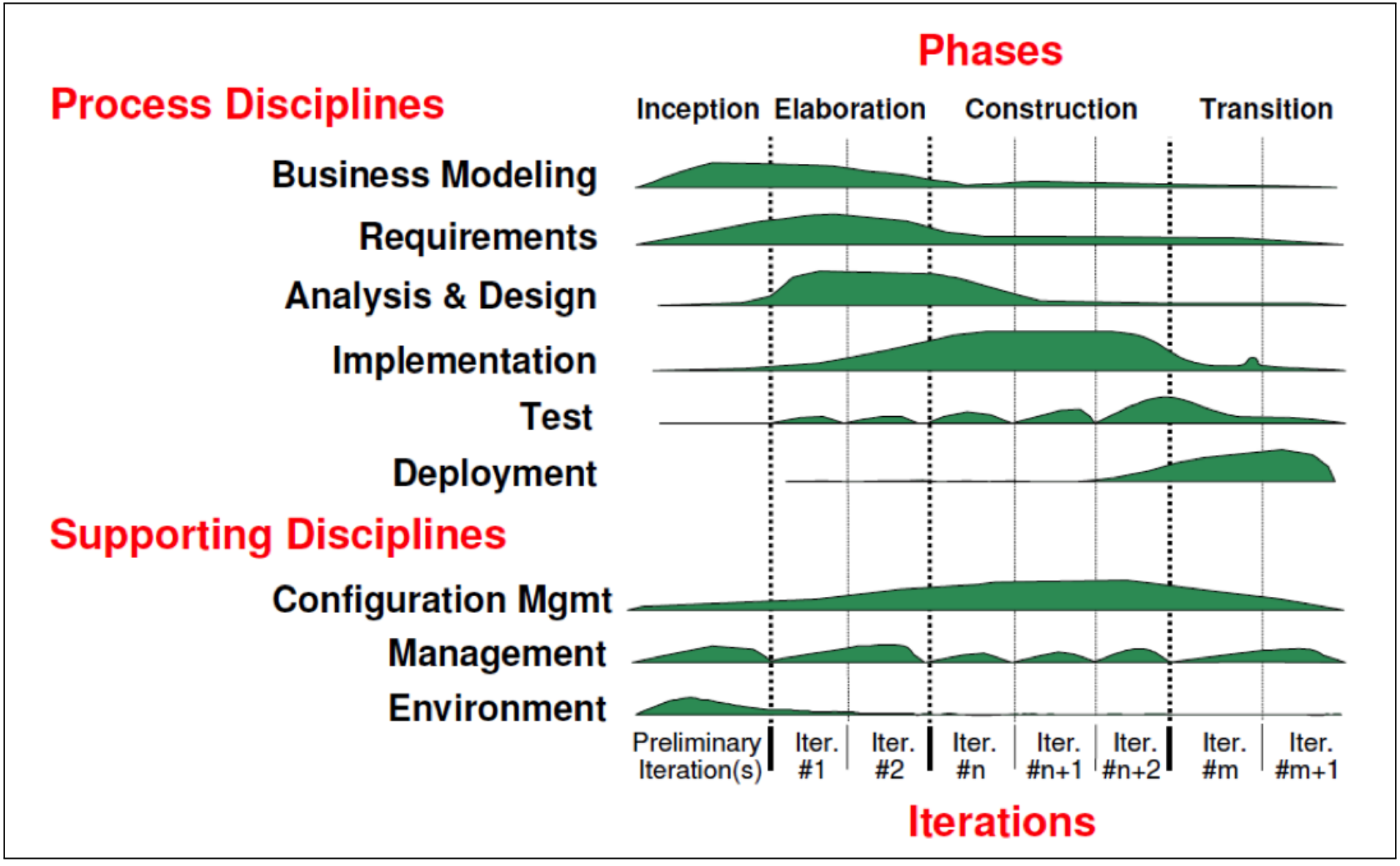
Technically not a process

... but a framework from which an iterative and incremental development process can be derived

Defines 4 sequential phases

Inception	Defines the scope of the project
Elaboration	Implements core architecture and critical (high risk, high value) use cases. So as to "drive-down" the risk of the entire project.
Construction	Implements secondary (non-critical) use cases. Alpha-testing
Transition	Deploys the system into "production". Beta-testing

The Unified Process



The Unified Process — Key Principles

Risk driven

Major risks "mitigated" during inception.
All risks mitigated by end of elaboration.

Use case driven

Use cases capture the functional requirements
use cases define the contents of the iterations
Each iteration realizes one or many use cases

Architecture-centric

Elaboration constructs a working architecture.