Lifecycle Models: Waterfall / Spiral / EVO

Dror Feitelson

Basic Seminar on Software Engineering
Hebrew University
2011

Lifecycle

- The sequence of actions that must be performed in order to build a software system
- Ideally thought to be a linear sequence: plan, design, build, test, deliverT
  - This is the waterfall model
- Realistically an iterative process
  - Including agile development and the Unified Process

Royce 1970

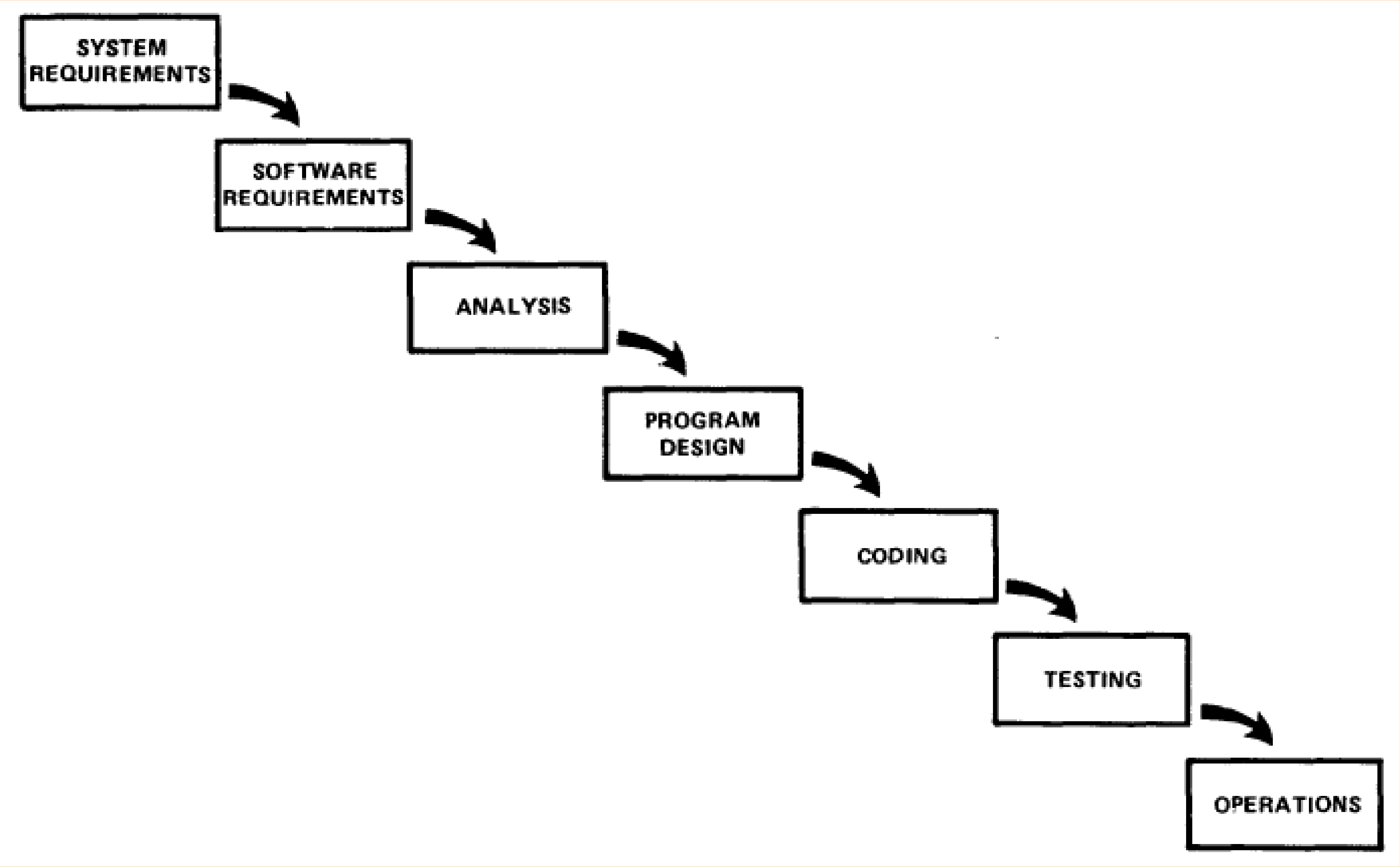
Dr. Winston W. Royce, “Managing the development of large software systems”.

Proc. IEEE WESCON, Aug 1970.

Reprinted 9th *Intl. Conf. Softw. Eng.*, 1987.

- Universally cited as the reference for the waterfall model
  - ➔ But, the word “waterfall” is not mentioned
  - ➔ And the model looks more like a cascade
- Moreover, the paper is actually against the waterfall model

The Basic Waterfall Model



SYSTEM REQUIREMENTS

SOFTWARE REQUIREMENTS

ANALYSIS

PROGRAM DESIGN

CODING

TESTING

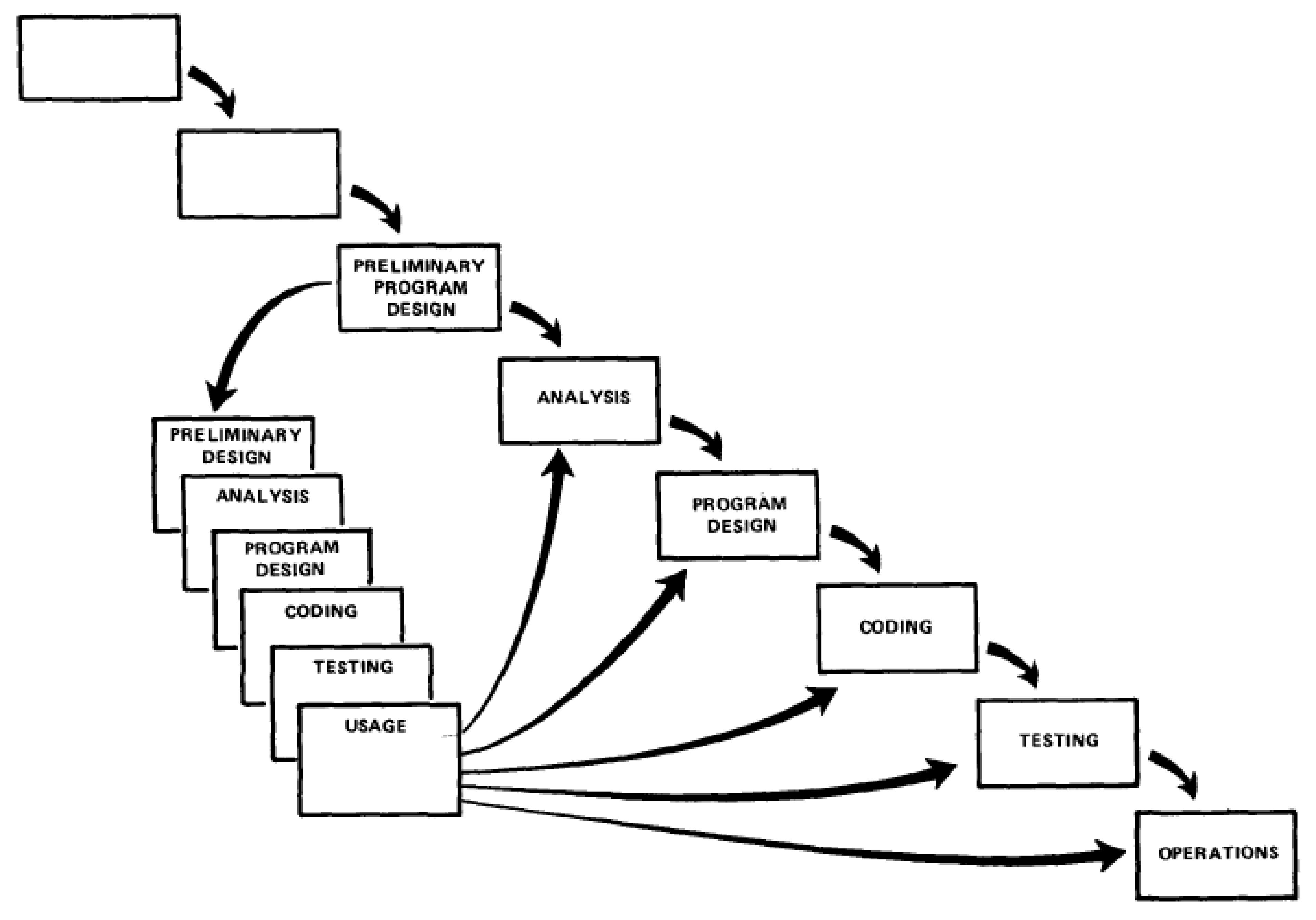
OPERATIONS

Problems

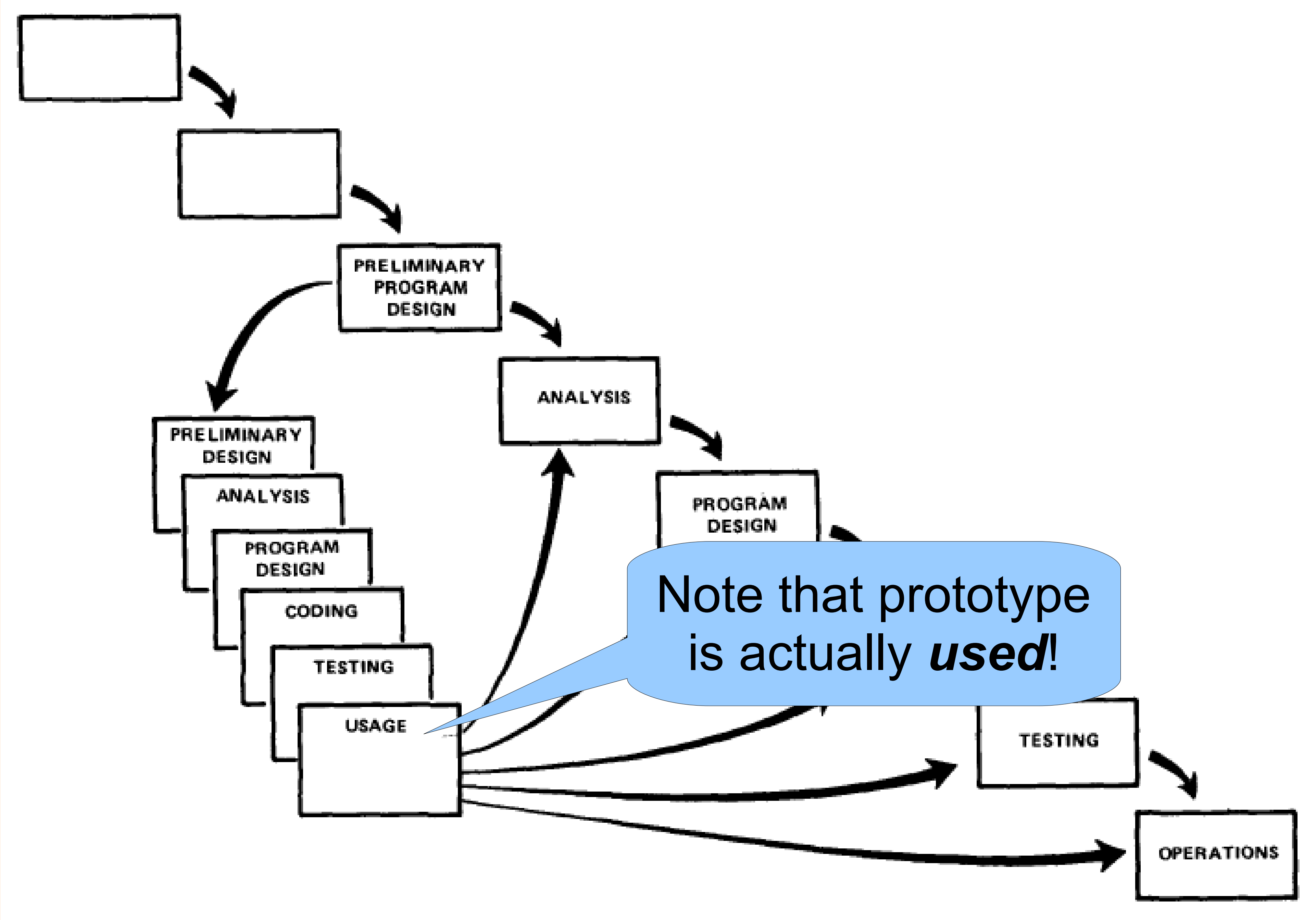
- Doing everything in a single sequence is unrealistic
- A better model involves iteration between successive steps
- However, testing comes too late and may uncover problems in the initial design
- The solution: do it twice

(Same advice as Fred Brooks in *The Mythical Man-Month*, but referring to a full-scale system)

Using a Prototype



Using a Prototype



Note that prototype is actually ***used***!

Additional Emphases

- Need to plan and control the testing
- Need to involve the client in key points
- Create multiple documents (requirements, specification, design, test plan, manual) and keep them up to date
  - "Write an overview document that is understandable, informative, and current. Each and every worker must have an elemental understanding of the system."
  - "If the documentation is in serious default my first recommendation is simple: replace project management."

The Frustration

This paper is very insightful and foreshadows several modern ideas.

So why is the waterfall model still being used?

(Or is it?)

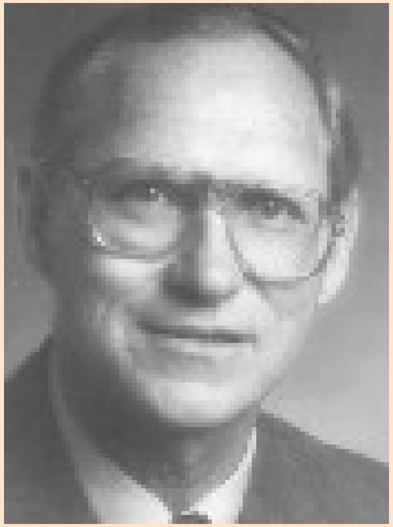
Documentation and Design

- The waterfall is document heavy
  - Including design documents
- Jack Reeves: the software is the design
  - Meaning the document, not the process: still need to think before you code
  - But the code embodies the design better than any other document
  - Actually building from the design is trivial and mechanized, unlike in other fields
  - Programmers must be creative designers, they are not assembly workers

Software as Design

- Software is incredibly cheap to build
- Software is incredibly expensive to design; everything (planning, designing, coding, testing) is part of the design process
- Creating a design or changing it is easy and cheap, leading to highly complex designs
- Testing and debugging are actually design validation
- Real advances depend on advances in programming techniques

Barry Boehm



Barry W. Boehm, “A spiral model of software development and enhancement”. *Computer* **21(5)**, pp. 61-72 May 1988.

- Prof. Software engineering, Univ. Southern California
- Worked at General Dynamics, Rand, TRW
- Director of DARPA Information Science and Technology Office 1989-1992
- Fellow of ACM, IEEE
- COCOMO cost model, Spiral model, ...

The Basic Force

- Code-driven development
  - “Code-and-fix” approach
  - No design leads to poor code and frustrated clients
- Document-driven development
  - Waterfall model
  - Requirement for fully developed documents unrealistic
- Risk-driven development
  - Support iterative development
  - Decide how to proceed by reducing risk of failure

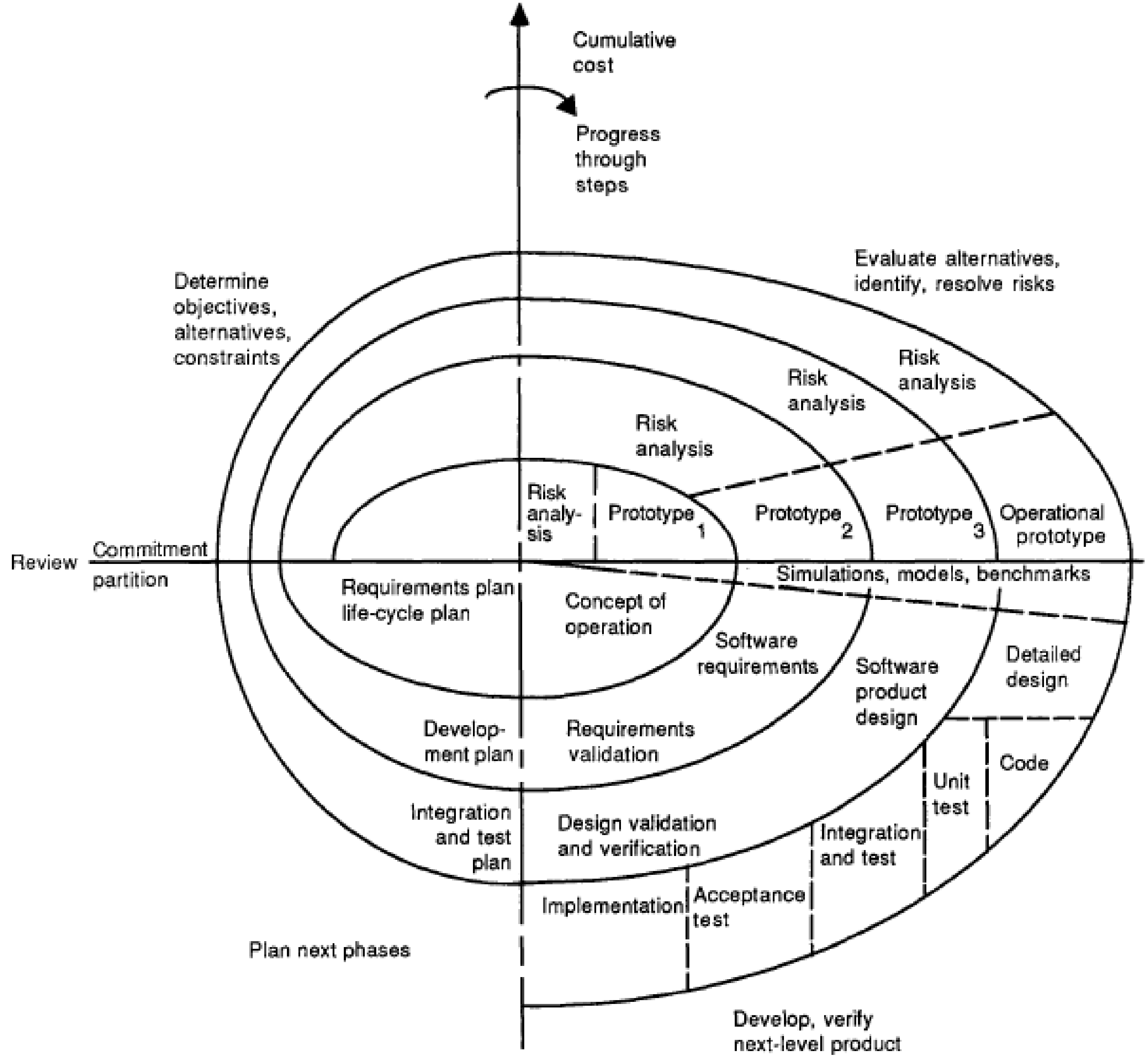
The Spiral Model

- Several rounds development: System concept, Requirements, design
- In each round, mitigate risks
  - Define objectives of part you are doing
  - Map alternatives for implementation
  - Recognize constraints on these alternatives
  - Use prototyping, analysis, etc. to gain necessary knowledge and reduce risk
  - Plan the next step
- At the end, perform sequence of coding, testing, and integration

The Spiral Model

- Several rounds development: System concept, Requirements, design
- In each round, mitigate risks
  - Define objectives of part you are doing
  - Map alternatives for implem
  - Recognize constraints on th
  - Use prototyping, analysis, e knowledge and reduce risk
  - Plan the next step
- At the end, perform sequence of coding, testing, and integration

What you actually do depends on the biggest remaining risk


Cumulative cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analy-sis
Prototype 1
Prototype 2
Prototype 3
Operational prototype
Review
Commitment partition
Simulations, models, benchmarks
Requirements plan life-cycle plan
Concept of operation
Software requirements
Software product design
Detailed design
Develop-ment plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration and test
Acceptance test
Implementation
Plan next phases
Develop, verify next-level product

Using the Spiral

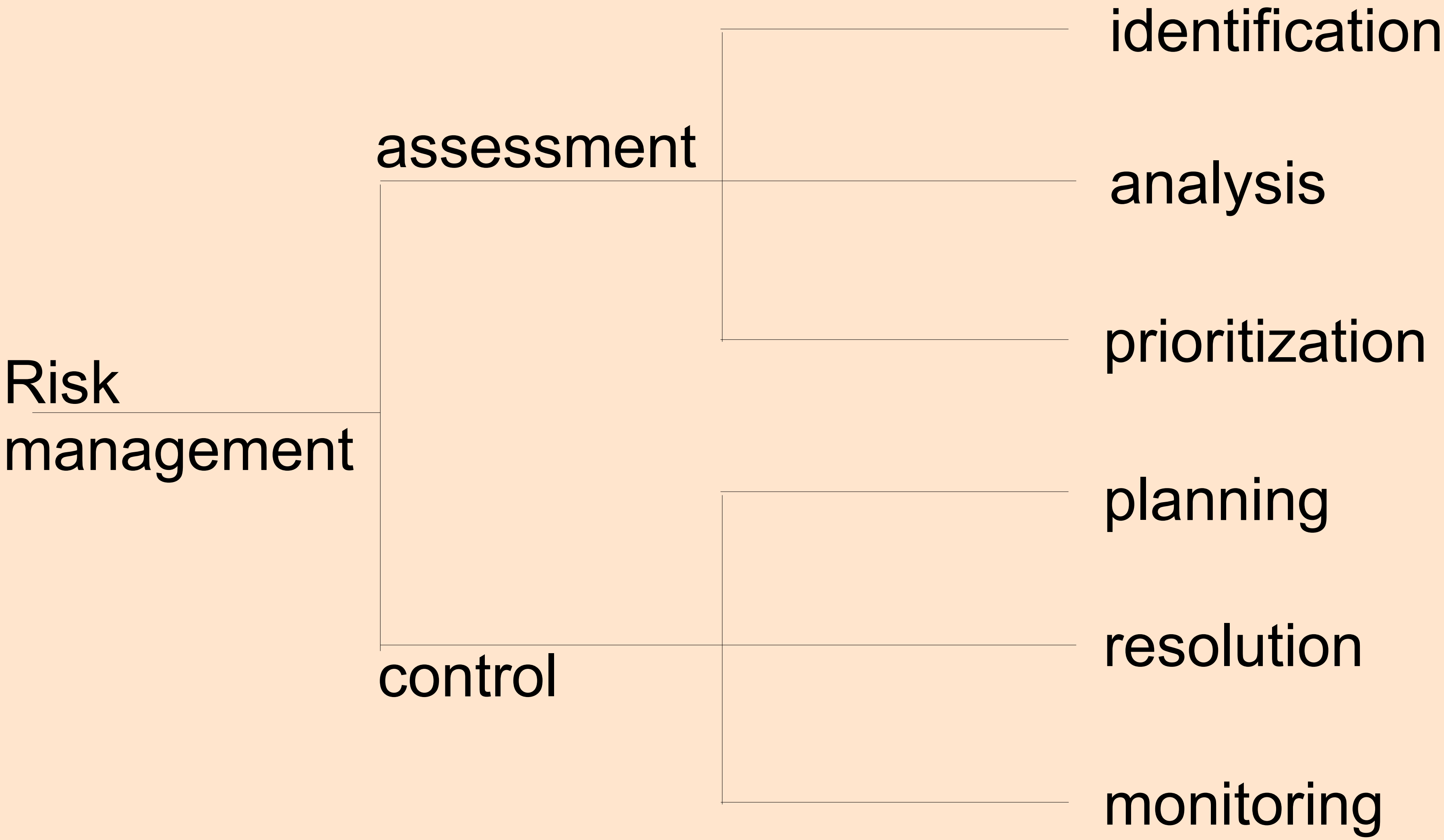
- Start with hypothesis that something can be done
- Round 1: concept and lifecycle plan
- Round 2: top level requirements
- Additional rounds: preliminary design, detailed design
- May go back and redo previous round if needed
- If the evaluation at some stage shows that it won't work then stop

Risks

- Developing software is fraught with uncertainty
- Uncertainty implies risk
- This needs to be quantified:

  RiskExposure = Probability x Loss
- Can be used to chose between alternatives: select the one where the expected loss is smaller

Risk Management



Risk management

assessment

identification

analysis

prioritization

control

planning

resolution

monitoring

Milestones

- In waterfall model there are many milestones
  - This is too rigid and sequential
- But there are three really important ones:
  - Life-cycle objectives
  - Life-cycle architecture
  - Initial operational capability

(these foreshadow the unified process)

Milestones

- In waterfall model there are many milestones
  - This is too rigid and seque
- But there are three really
  - Life-cycle objectives
  - Life-cycle architecture
  - Initial operational capability

(these foreshadow the unified process)

Make sure we know what we want to do, and that it can be done

Milestones

- In waterfall model there are many milestones
  - This is too rigid and seque
- But there are three really
  - Life-cycle objectives
  - Life-cycle architecture
  - Initial operational capability

(these foreshadow the unified process)

Make sure we

Elaborate on how things will be built

Milestones

- In waterfall model there are many milestones
  - This is too rigid and seque
- But there are three really
  - Life-cycle objectives
  - Life-cycle architecture
  - Initial operational capability

(these foreshadow the unified process)

Make sure we

Elaborate on

Prepare for the transition to the client in terms of site and training

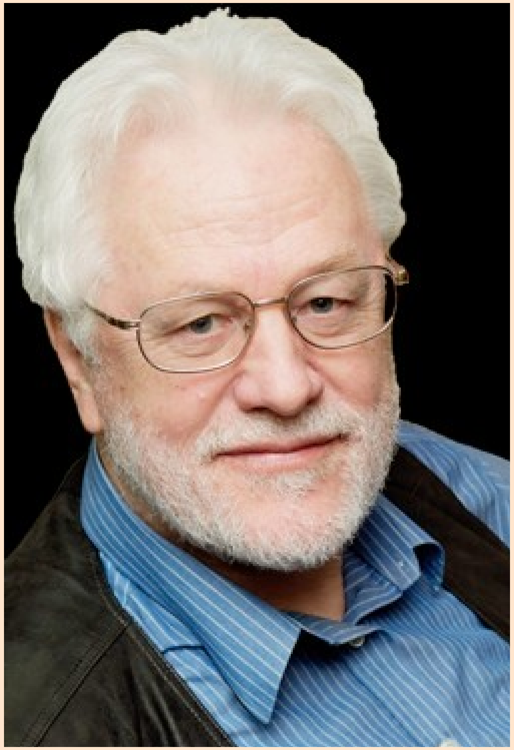
Milestones

- Milestones are not (necessarily) documents!
  - ➔ Not a fully specified spec or architecture, but a framework that will evolve
  - ➔ For example, important interfaces must be specified precisely, but user interfaces can be a prototype
  - ➔ Articulation of feasibility and rationale are important
  - ➔ Agreement of stakeholders is crucial

Conceptual Development with Time

- Spiral model (1988): in an example round 0 is about deciding that the project is worth doing
- Risk management (1991): one of the risks is that the project is plain wrong
- Anchoring (1996): the first anchor point is agreement among stakeholders that the project can and should be done

Tom Gilb



Principles of Software Engineering Management, Addison-Wesley, 1988

- Early work on iterative and incremental development
- EVO: evolutionary software delivery
- Early work on software metrics
- Early work on inspections
- Independent consultant with his son

Requirements

- Building software is a learning process
- We don't know what the client wants
- Regrettably, the client doesn't know either
- But he'll know it when he sees it
- So we need to create something for him to see
- Hence iterative and incremental development

Engineering

- Requirements is not only what the system should do
- It is also how well it should be done
  - What resource expenses are acceptable
  - What performance level is needed
- Skillful, knowledgeable professionals are needed in order to design and architect a solution
  - satisfying use-cases is not enough

Methodology

- Identify critical stakeholders
- Find what value they are looking for
- Identify solutions
- Develop
- Deliver value early
- Iterate and learn

Evolutionary Delivery

- Lead time to first working and useful system is short
- Real users doing real work brought into the loop
  - Testing in realistic conditions
  - Prioritization of subsequent development
- System and its environment co-evolve
- Respond to changes
  - Can't freeze the world anyway, so make it a feature
- Exploit new technology as it becomes available

Main Comparison

Sequential plans:

- Freeze requirements
- Testing of complete product
- Big bang delivery
- All-or-nothing risks large-scale failures

Iterative / evolutionary:

- Incremental learning of what is needed
- Experience in the field with partial solution
- Incremental delivery
- Hard to fail bigtime

Summary

- Royce: plan ahead and document
- Boehm: iterate and reduce biggest risk each time
- Gilb: iterate and deliver maximal value each time
- Agile: iterate to make progress each time
- Old school: requirement must be met, so compromise schedule and overrun budget if needed
- New school: do the most useful thing within time and money constraints