

Chapter 4

■ Process Models

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman and Bruce R. Maxim

Slides copyright © 1996, 2001, 2005, 2009, 2014 by Roger S. Pressman

For non-profit educational use only

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach, 8/e*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

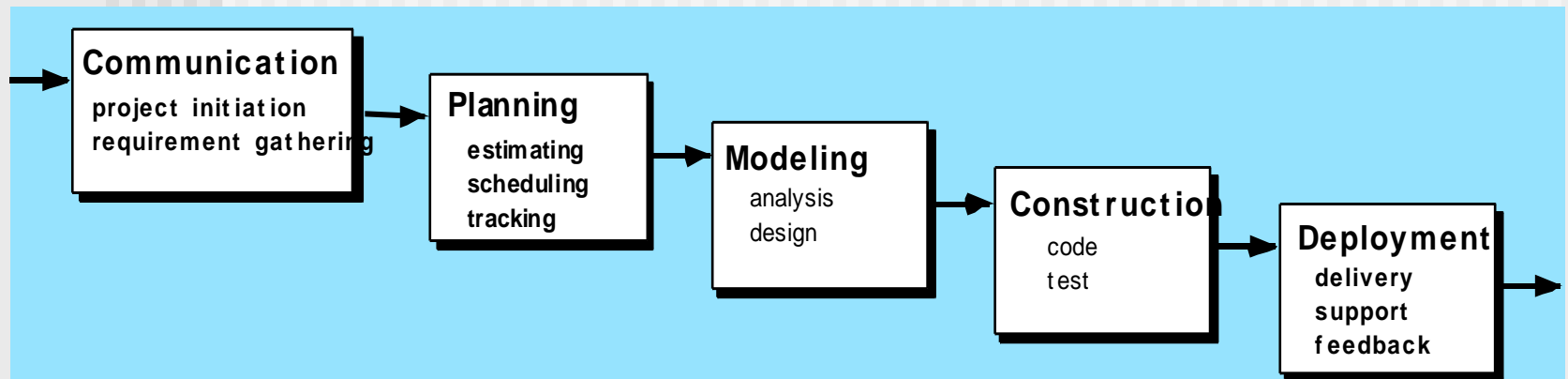
Prescriptive Models

- Prescriptive process models advocate an orderly approach to software engineering

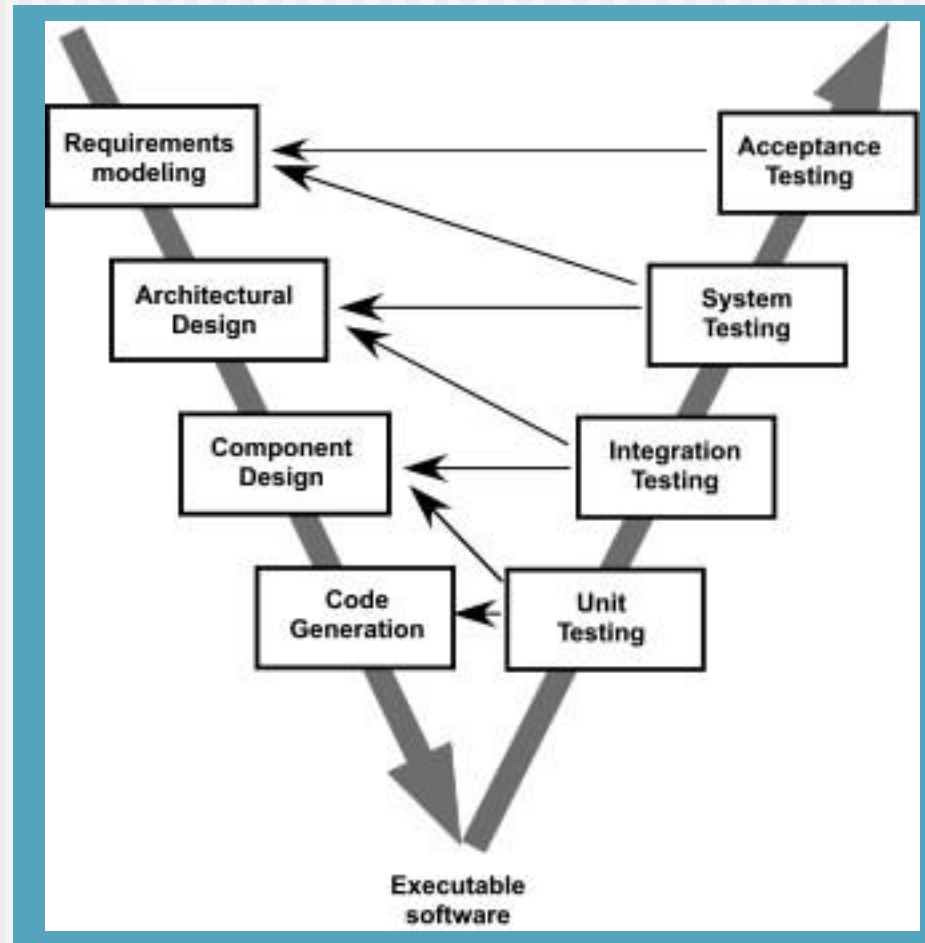
That leads to a few questions ...

- If prescriptive process models strive for structure and order, **are they inappropriate for a software world that thrives on change?**
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, **do we make it impossible to achieve coordination and coherence in software work?**

The Waterfall Model

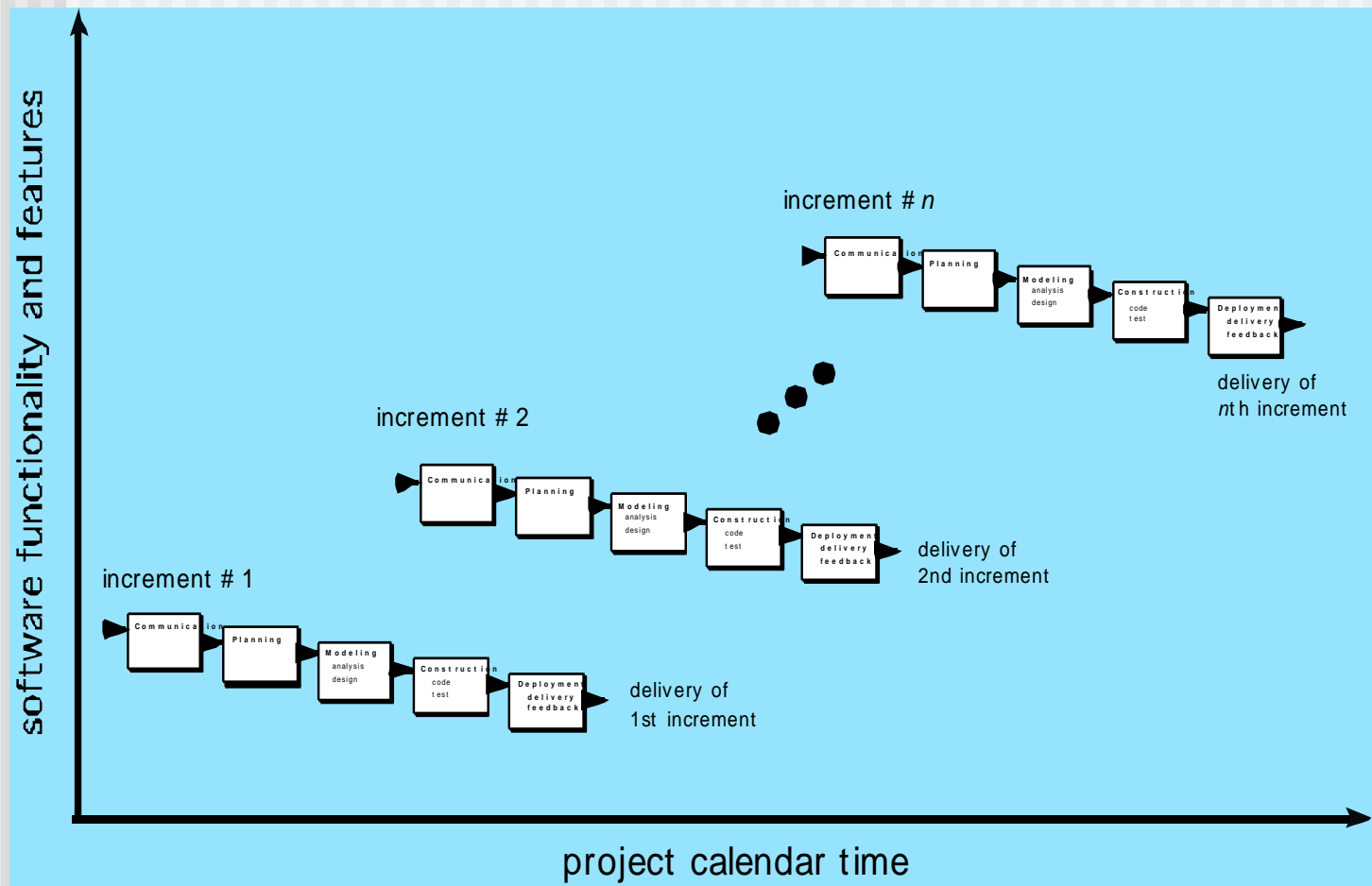


The V-Model



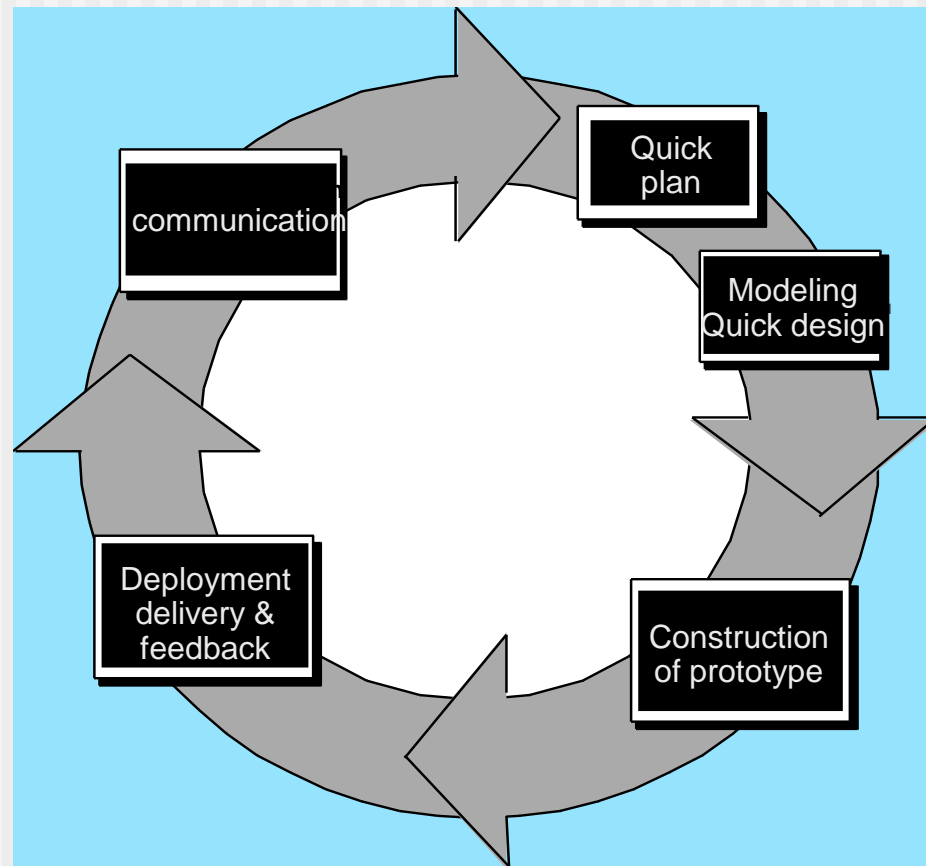
These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 8/e (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

The Incremental Model

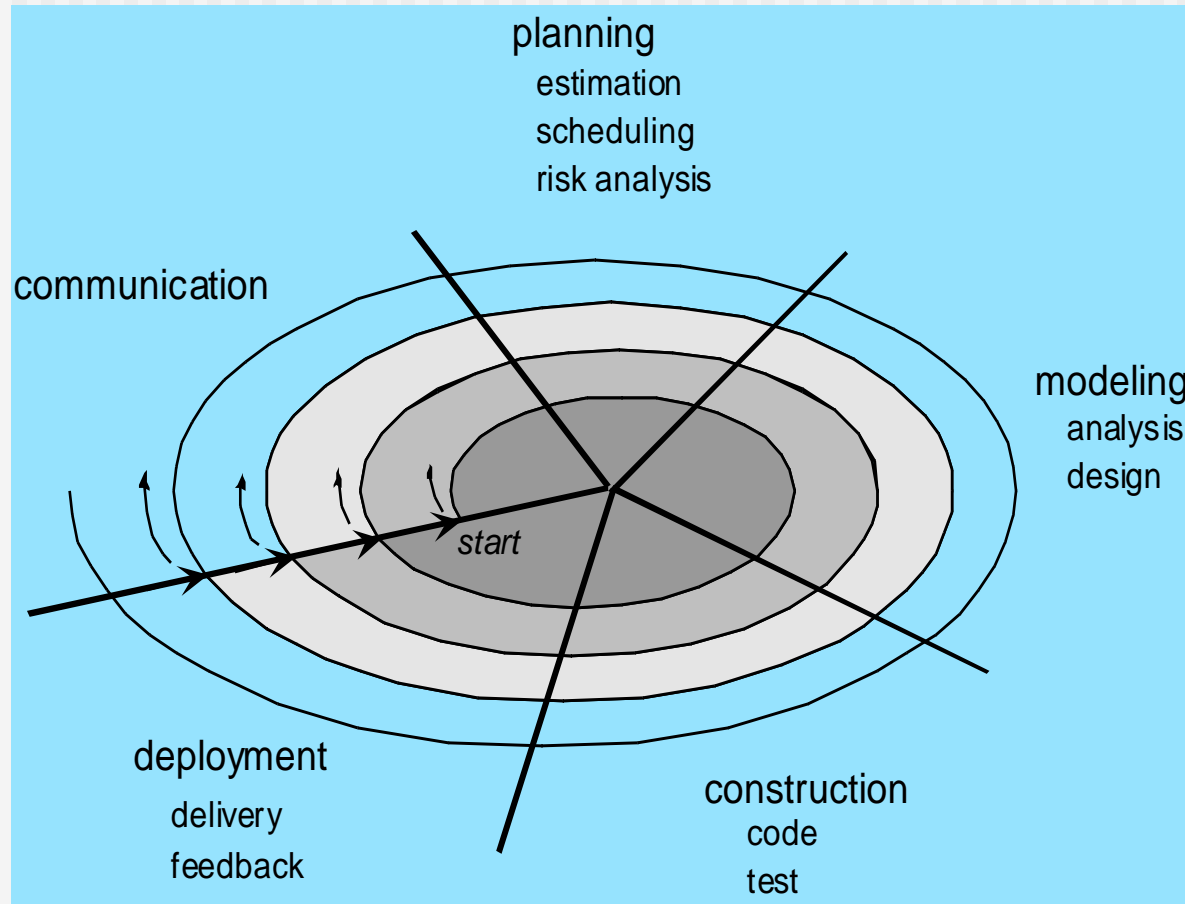


These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 8/e (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

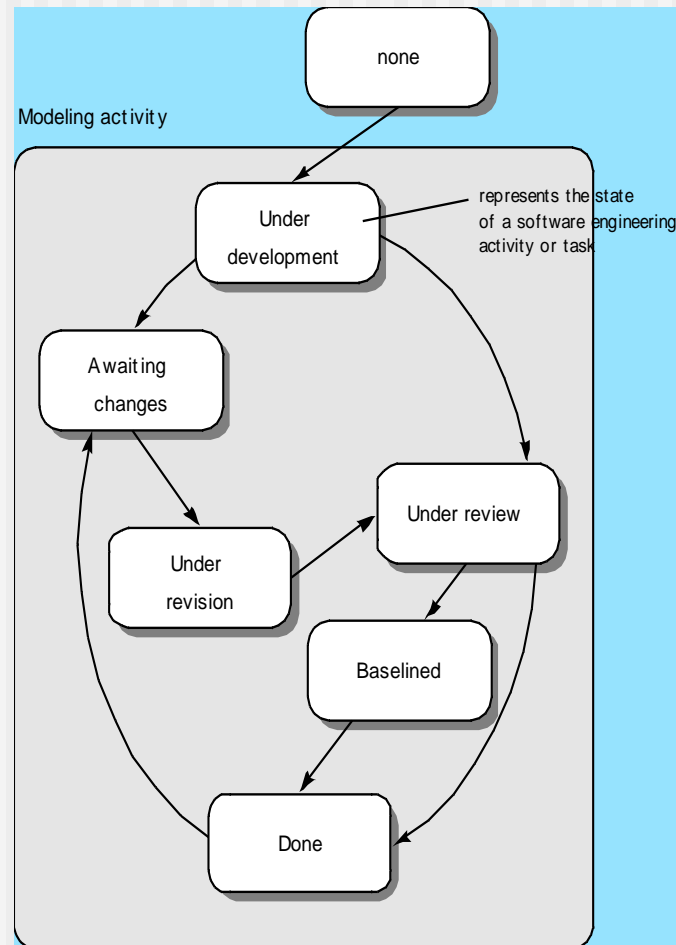
Evolutionary Models: Prototyping



Evolutionary Models: The Spiral



Evolutionary Models: Concurrent

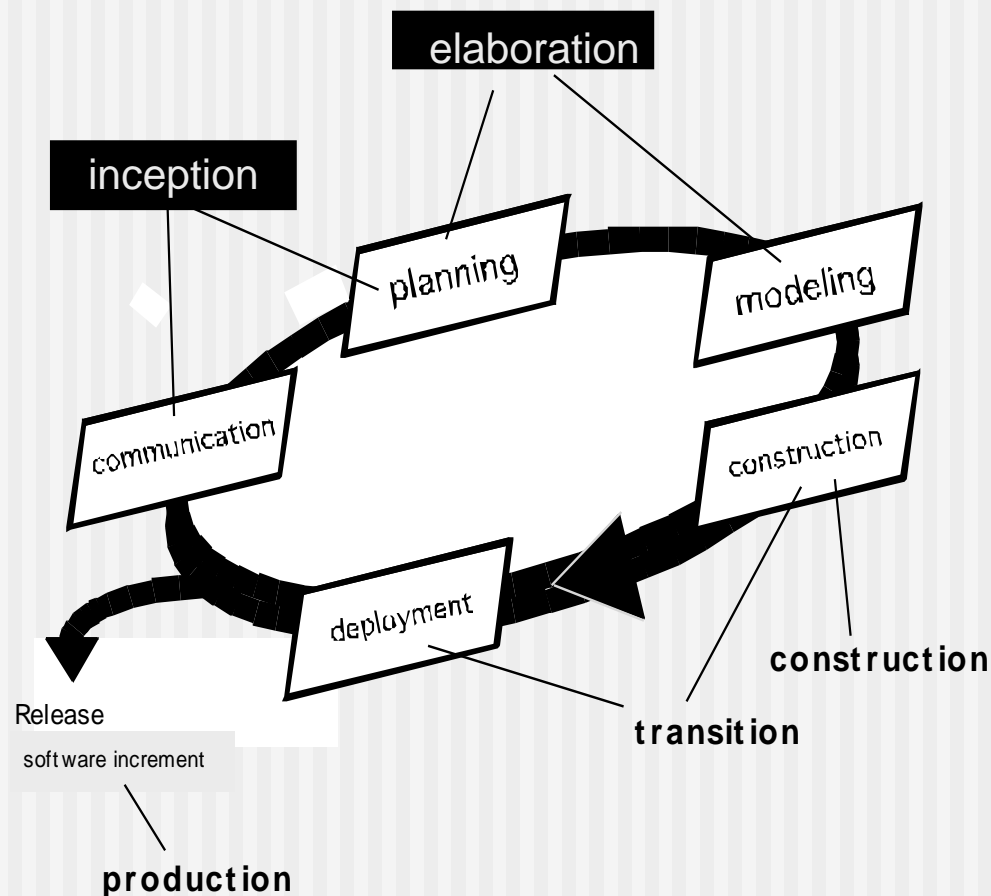


These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 8/e (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

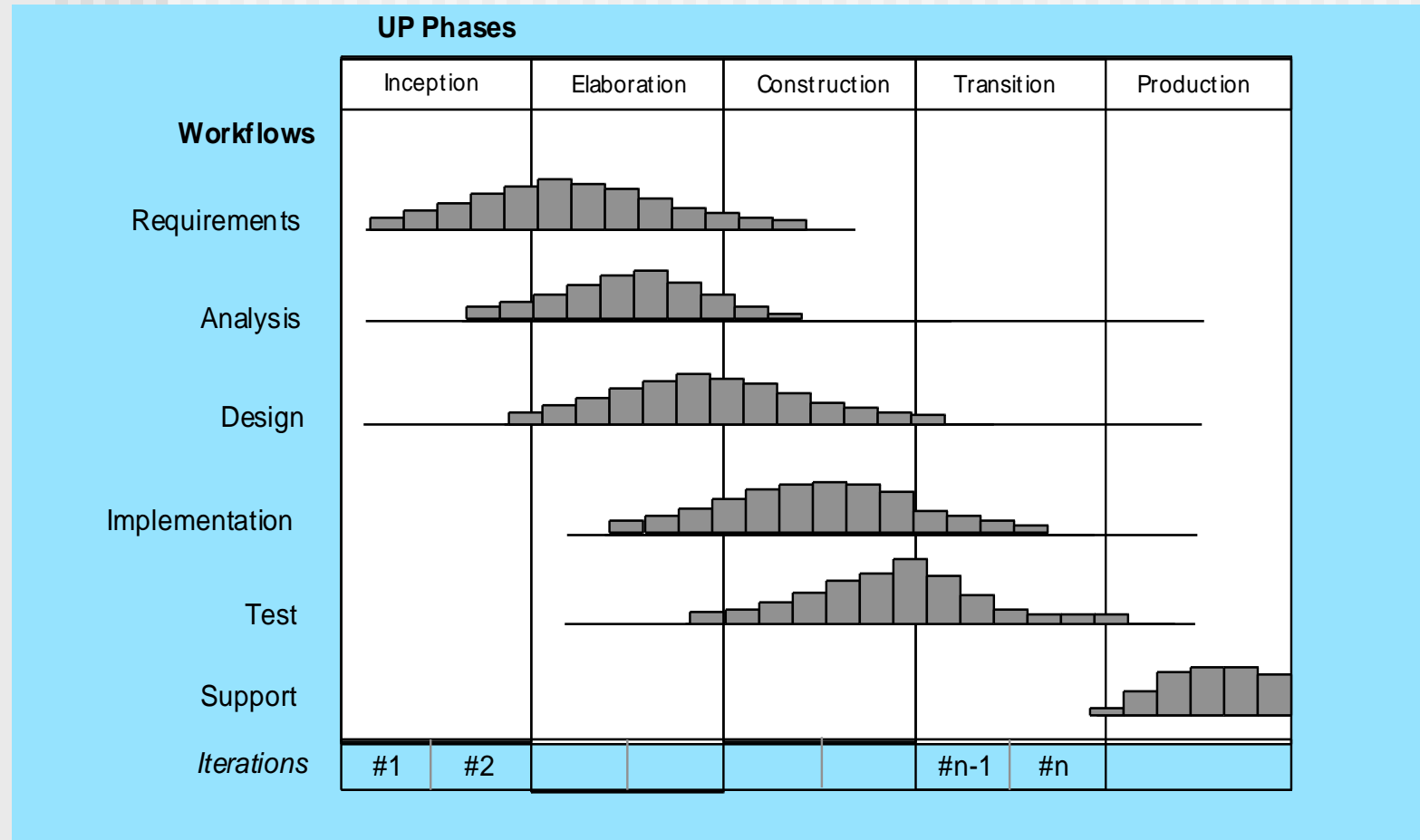
Still Other Process Models

- **Component based development**—the process to apply when reuse is a development objective
- **Formal methods**—emphasizes the mathematical specification of requirements
- **AOSD**—provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*
- **Unified Process**—a “use-case driven, architecture-centric, iterative and incremental” software process closely aligned with the Unified Modeling Language (UML)

The Unified Process (UP)



UP Phases



UP Work Products

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
phases and iterations.
Business model,
if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
including non-functional
Analysis model
Software architecture
Description.
Executable architectural
prototype.
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted workflows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

Personal Software Process (PSP)

- **Planning.** This activity isolates requirements and develops both size and resource estimates. In addition, a defect estimate (the number of defects projected for the work) is made. All metrics are recorded on worksheets or templates. Finally, development tasks are identified and a project schedule is created.
- **High-level design.** An external specification is created for each component and a component design is created. Prototypes are built when uncertainty exists. All issues are recorded and tracked.
- **High-level design review.** Formal verification methods (Chapter 21) are applied to uncover errors in the design. Metrics are maintained for all important tasks and work results.
- **Development.** The component level design is refined and reviewed. Code is generated, reviewed, compiled, and tested. Metrics are maintained for all important tasks and work results.
- **Postmortem.** Using measures and metrics collected the effectiveness of the process is determined. If this is a large amount of data it should be analyzed statistically), Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

Team Software Process (TSP)

- Build self-directed teams that plan and track their work, establish goals, and own their processes and plans. These can be pure software teams or integrated product teams (IPT) of three to about 20 engineers.
- Show managers how to coach and motivate their teams and how to help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
 - The Capability Maturity Model (CMM), a measure of the effectiveness of a software process, is discussed in Chapter 30.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.