Chapter 4

- **Process Models**

Slide Set to accompany

Software Engineering: A Practitioner's Approach

by Roger S. Pressman

Table of Contents

4.1 Prescriptive Process Models

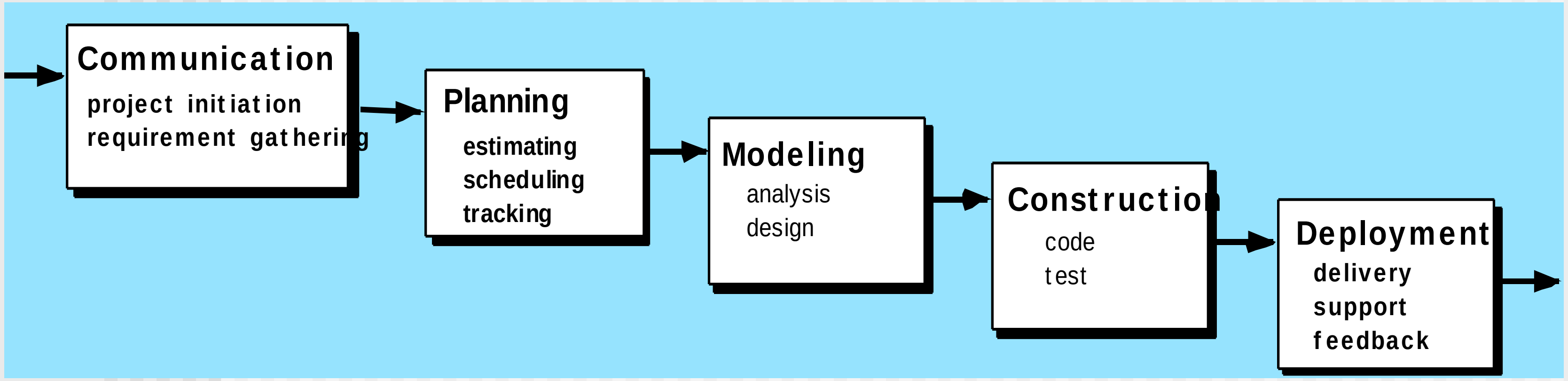
(= Traditional Process Models)

- Prescriptive process models advocate an orderly approach to software engineering

That leads to the questions:

- If prescriptive process models strive for structure and order, are they appropriate for a software world that thrives on change?
- Yet, if we reject them and replace them with something less structured, is it possible to achieve coordination and coherence in software work?

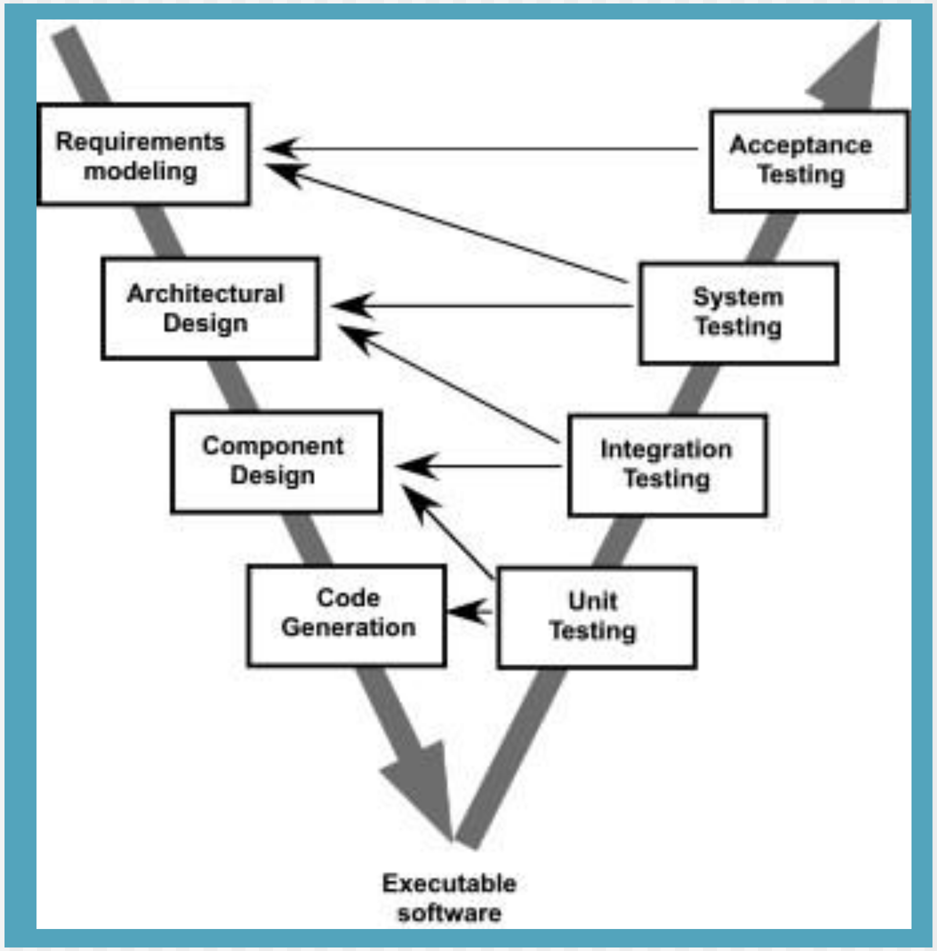
4.1.1 The Waterfall Model



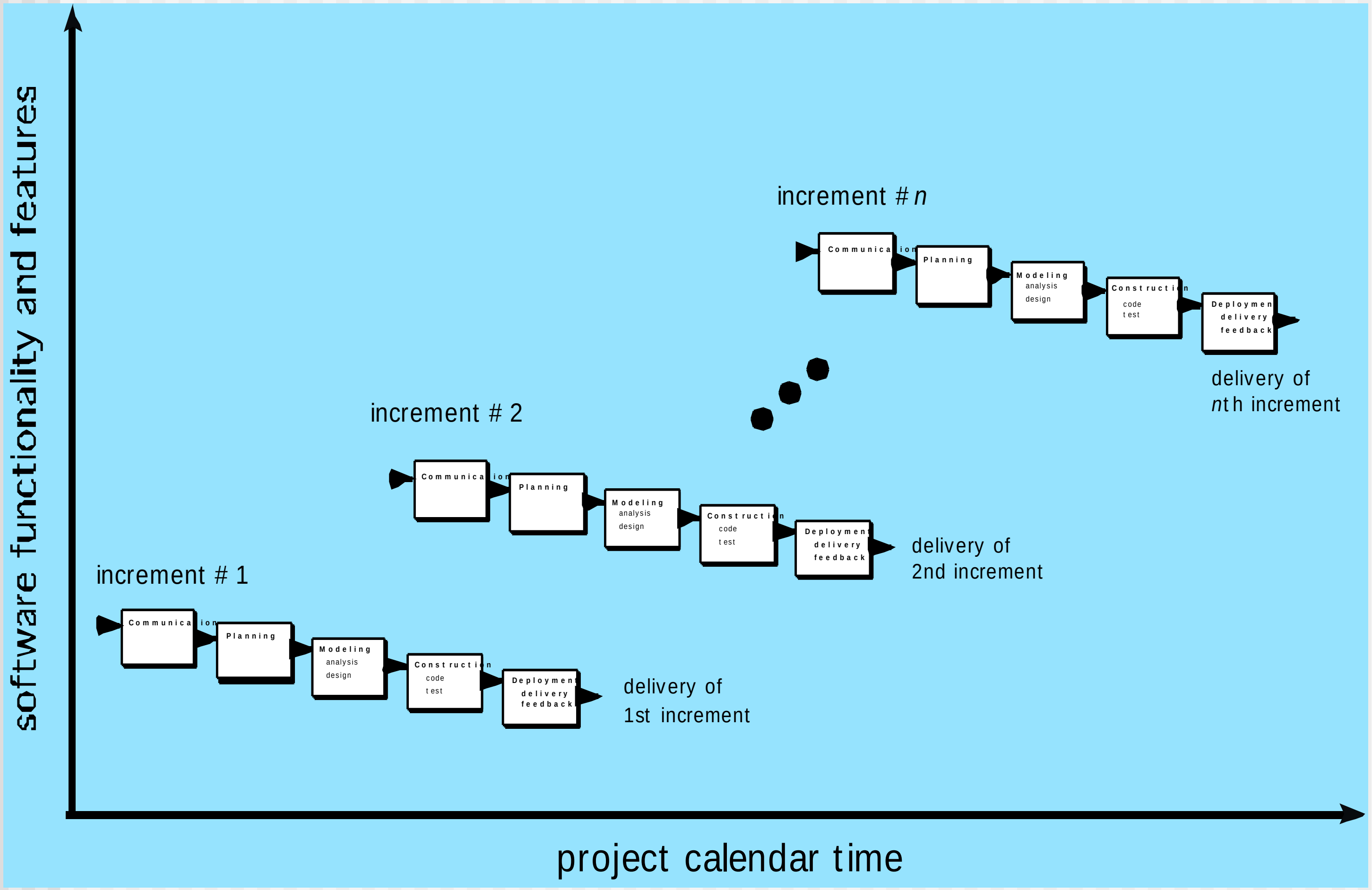
- A.k.a. the classic life cycle
- Logical model rather than a model for practice
- A variation of this model has arrows from each stage back to the previous stages

The V-Model

Quality assurance model has been added to the classic life cycle

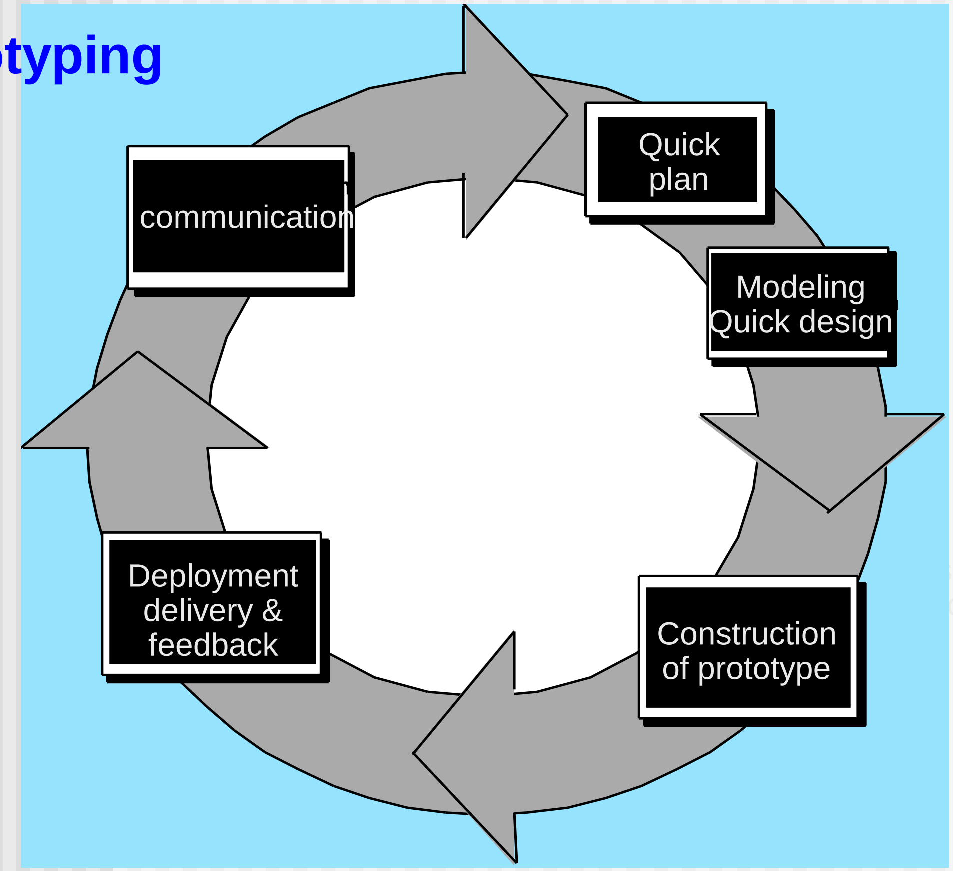


4.1.2 The Incremental Model



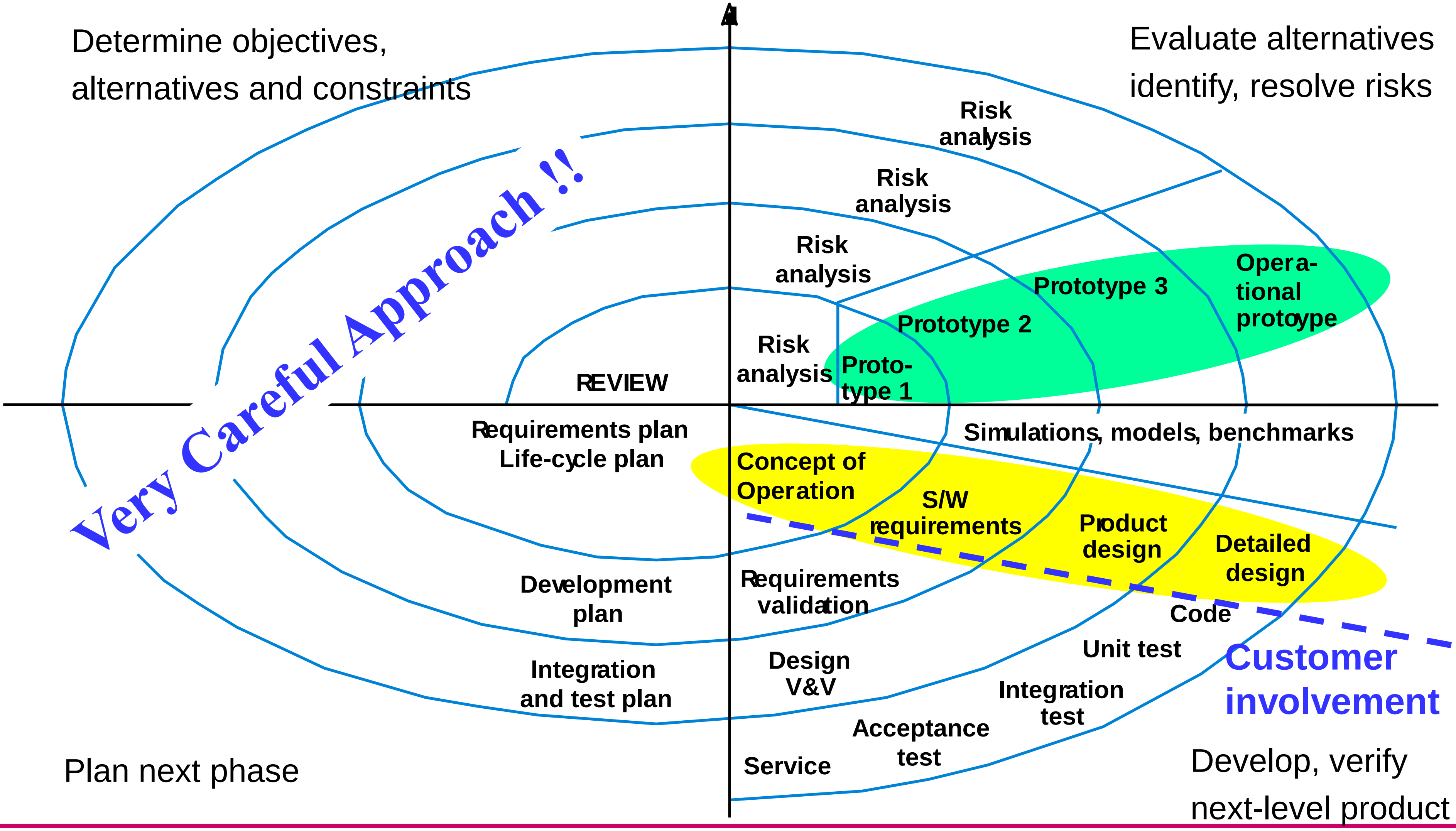
4.1.3 Evolutionary Process Models

The Prototyping Paradigm



[Brooks 95]
"In most projects, the first system built is barely usable. It may be too slow, too big, awkward in use or all three. There is no alternative but to start again … and build a redesigned version in which these problems are solved.
☛ Throw-away Prototype
(<–> Evolutionary prototype)

Boehm's spiral model of the software process



The Spiral Model

Two main distinguishing features[Boehm 01a]:

(1) a *cyclic approach* for incremental growing a system's degree of definition and implementation while decreasing its degree of risk.

(2) a set of *anchor point milestones* for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions."

4.2 Specialized Process Models(SKIP)

Some people do not view these as *process models*.

- Component based development—the process to apply when reuse is a development objective
- Formal methods—emphasizes the mathematical specification of requirements
- AOSD (= Aspect Oriented Software Development) —provides a process and methodological approach for defining, specifying, designing, and constructing *aspects*

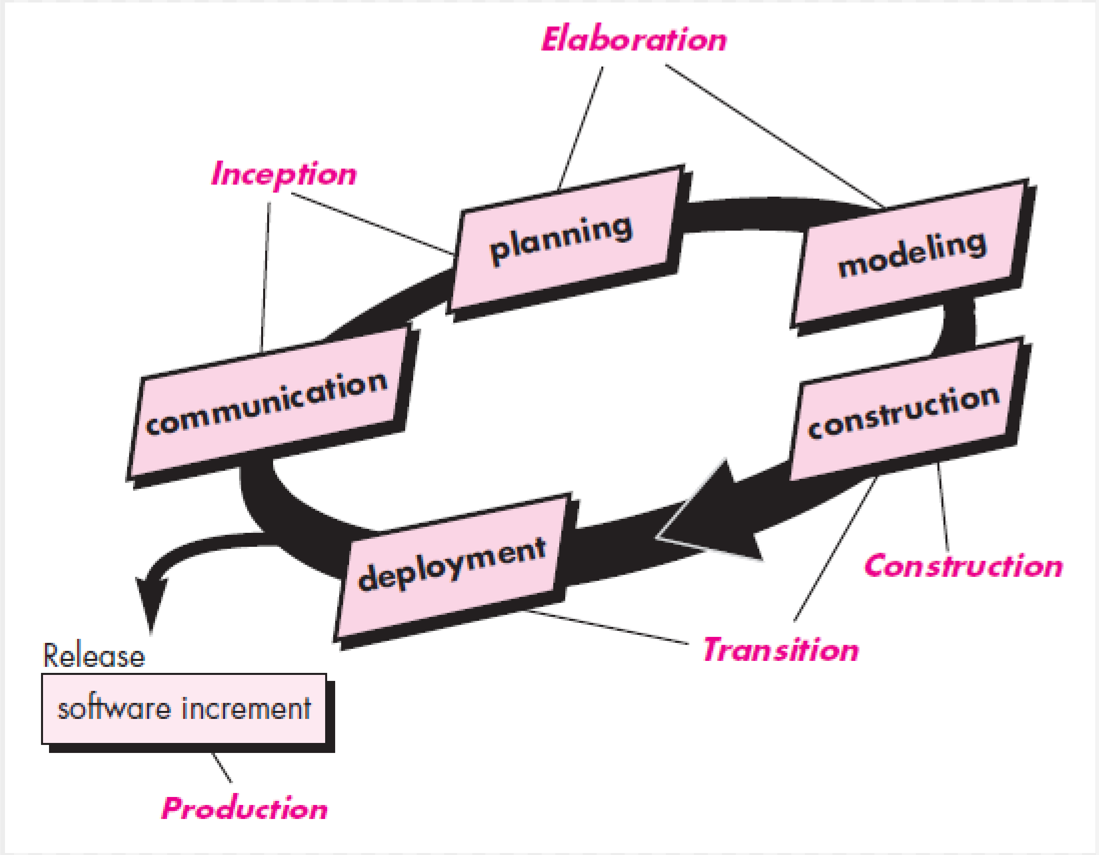
4.3 The Unified Process (UP)

Unified Process (= UP)

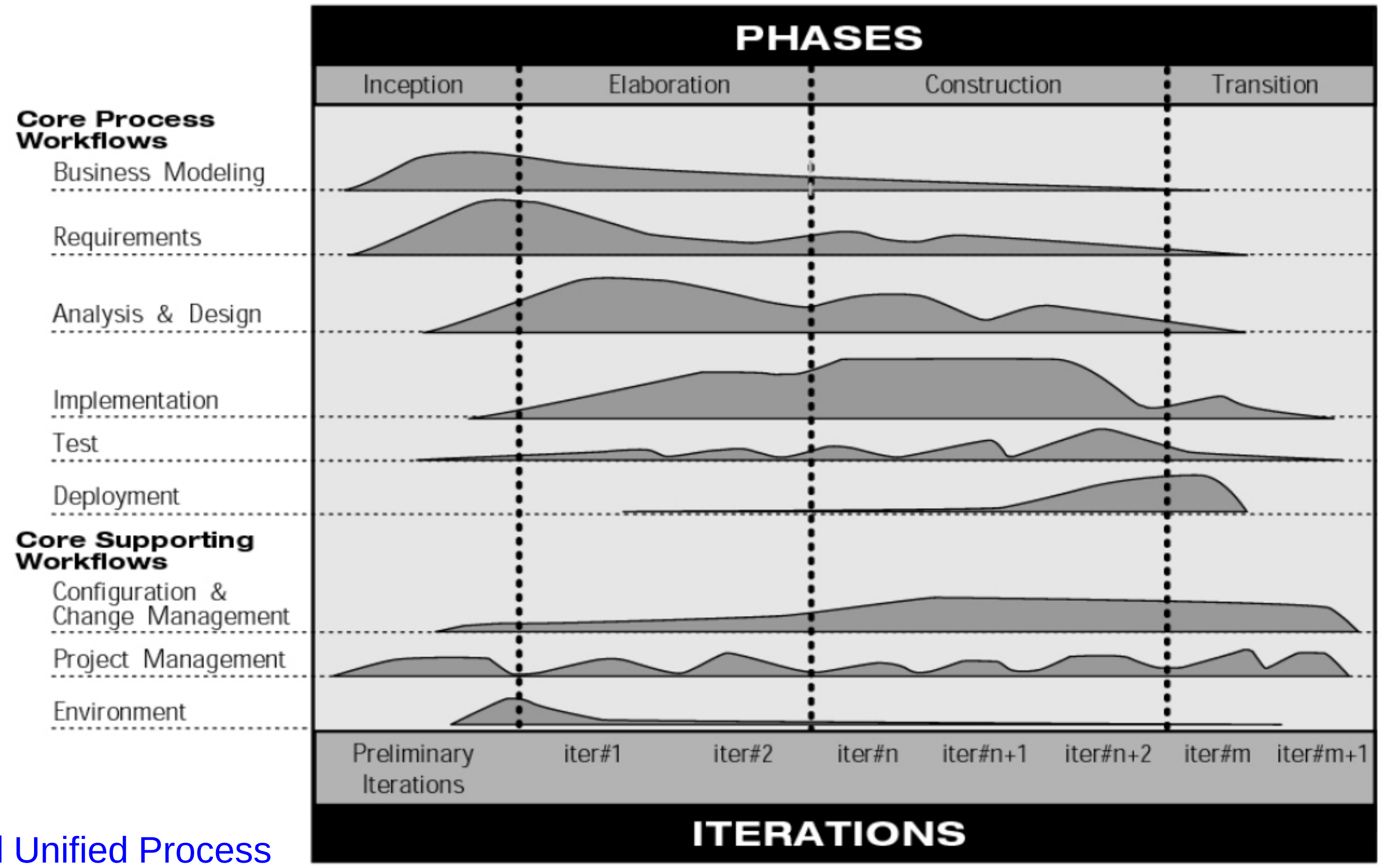
— a “use-case driven, architecture-centric, iterative and incremental” software process

* *Unified* Modeling Language (= UML)

- Developed by James Rumbaugh, Grady Booch and Ivar Jacobson



UP Phases



Source: Rational Unified Process Best Practices for Software Development Teams

The nine core process workflows

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

4.4 Personal and Team Process Models

4.4.1 Personal Software Process (PSP)

- **PSP**: Software process for person.
- In order to change an ineffective personal process, an individual must move through 4 phases.
- The PSP Evolution:
  - PSP0 - The Baseline Process
    - Includes some measurements and a reporting format
  - PSP1 - The Personal Planning Process
    - PSP0 + planning
  - PSP2 - Personal Quality Management
    - PSP1 + personal design and code reviews
  - PSP3 - A Cyclic Personal Process
    - For large scale development, subdivide larger programs into PSP2-sized pieces
    - Suitable for programs of up to several KLOC.

Personal Software Process (PSP)

- **Planning.**
  - Isolates requirements and develops both size and resource estimates.
  - A defect estimate (the number of defects projected for the work) is made.
  - All metrics are recorded on worksheets or templates.
  - Development tasks are identified and a project schedule is created.
- **High-level design.**
  - External specifications and design for each component are developed.
  - Prototypes are built when uncertainty exists.
  - All issues are recorded and tracked.
- **High-level design review.**
  - Formal verification methods 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 the measures and metrics, the effectiveness of the process is determined.
  - Measures and metrics should provide guidance for modifying the process to improve its effectiveness.

Personal discipline for software development

Do you do all these things?

4.4.2 Team Software Process (TSP)

- **TSP**: Software process for team
- To build self-directed teams that plan and track their work, establish goals, and own their processes and plans.
  - Can be pure software teams or integrated product teams (IPT) of 3 ~ 20 engineers.
- Show managers how to coach and motivate their teams and help them sustain peak performance.
- Accelerate software process improvement by making CMM Level 5 behavior normal and expected.
- Provide improvement guidance to high-maturity organizations.