Chapter 3

Process Models

Software Engineering: A Practitioner's Approach

6th Edition

Roger S. Pressman

The Waterfall Model (1)

- Sometimes called the *classic life cycle*
- Suggests a systematic, sequential (or linear) approach to s/w development
- The oldest paradigm for s/w engineering
- Works best when –
  - Requirements of a problem are reasonably well understood
  - Well-defined adaptations or enhancements to an existing system must be made
  - Requirements are well-defined and reasonably stable

The Waterfall Model (2)

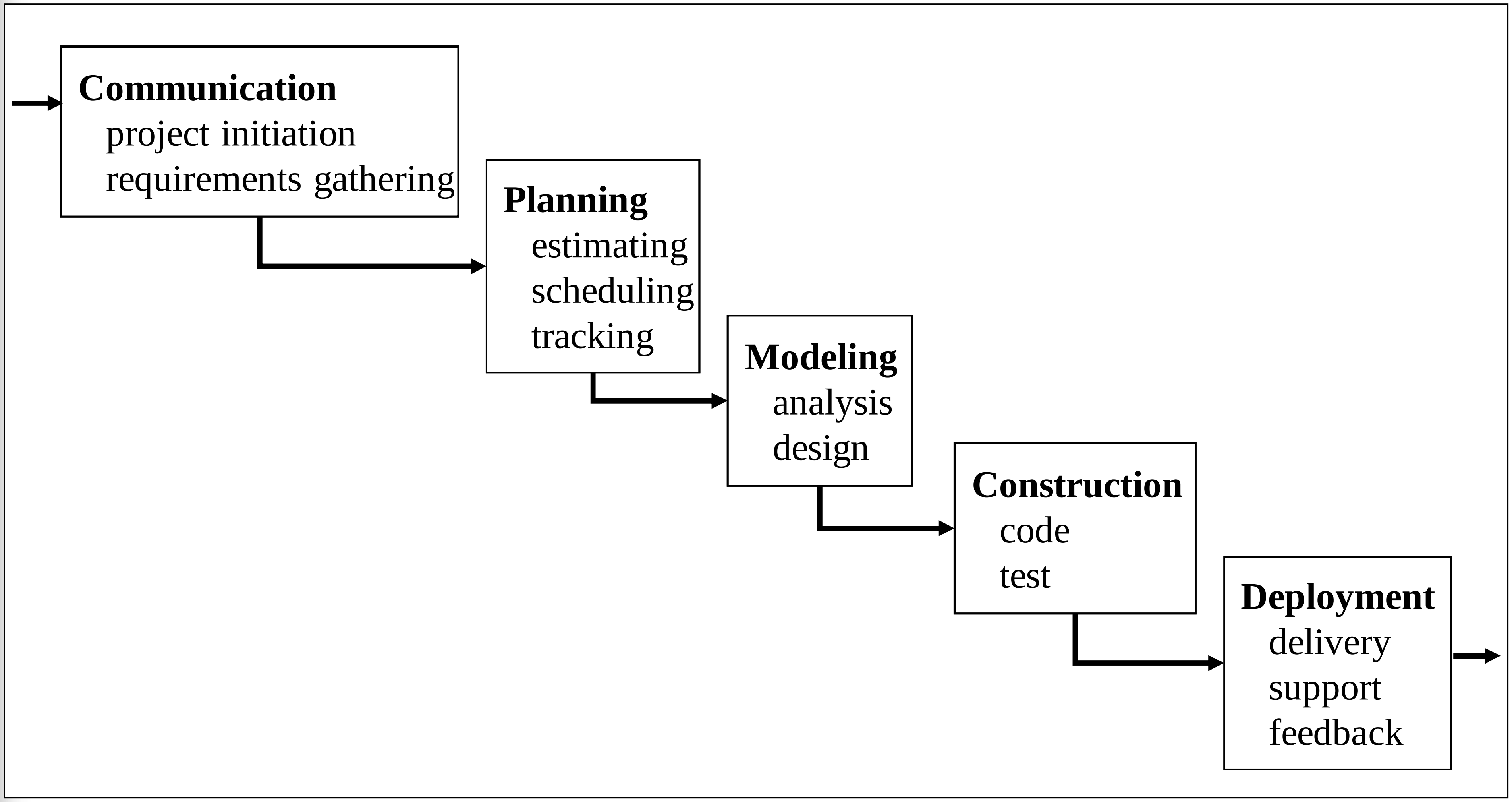
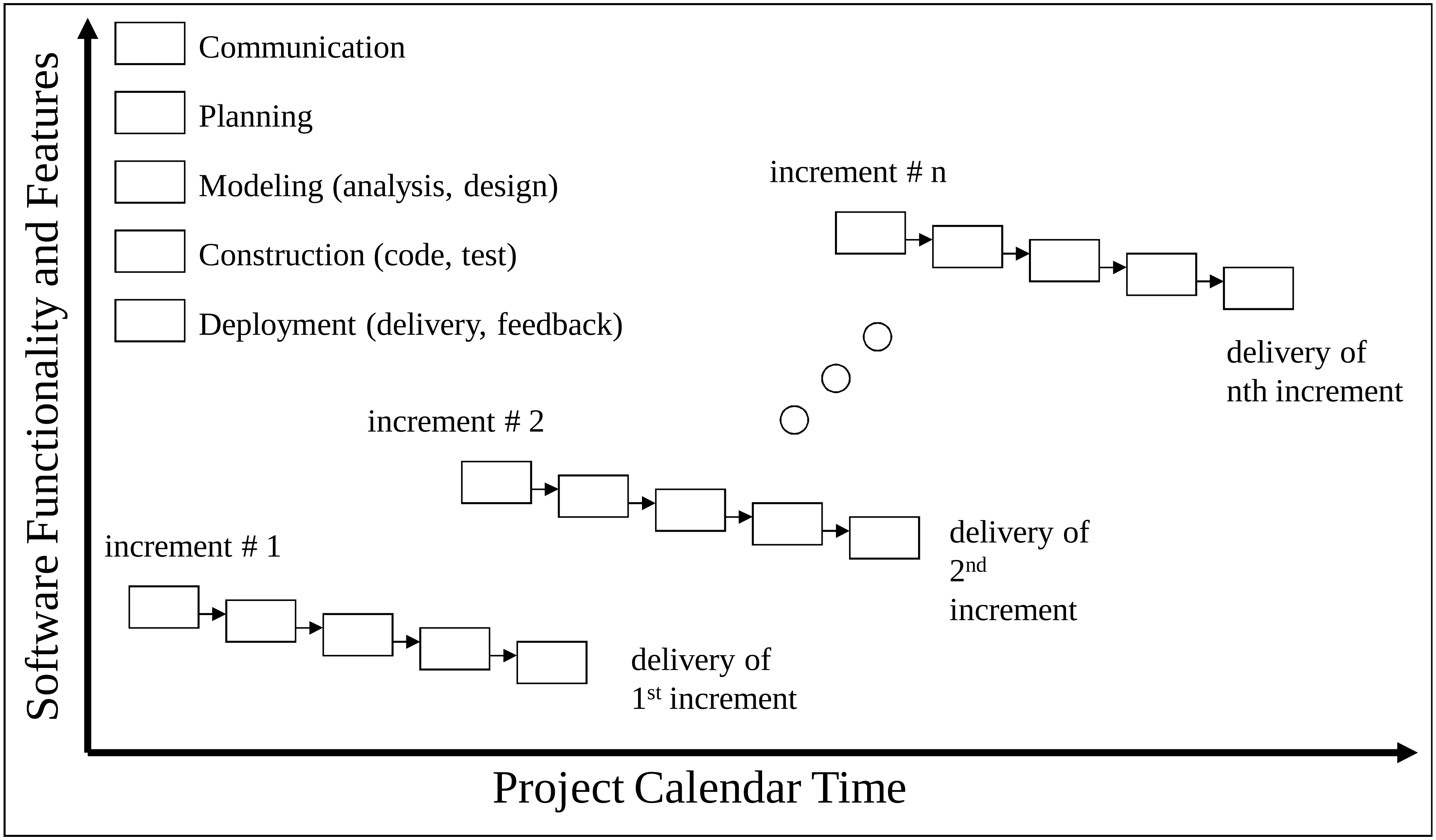


Fig 3.1: The waterfall model

The Waterfall Model - Problems

- Real projects rarely follow the sequential flow
  - Accommodates iteration indirectly
  - Changes can cause confusion
- It is often difficult for the customer to state all requirements explicitly
  - Has difficulty accommodating the natural uncertainty that exists at the beginning of many projects
- The customer must have patience
  - A working version of the program(s) will not be available until late in the project time-span
  - A major blunder, if undetected until the working program is reviewed, can be disastrous
- Leads to “blocking states”

Incremental Process Models (1)



Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # n
increment # 2
increment # 1
delivery of nth increment
delivery of 2nd increment
delivery of 1st increment
Software Functionality and Features
Project Calendar Time

Fig 3.2: The incremental model

Incremental Process Models (2)

- Combines elements of the waterfall model applied in an iterative fashion
- Each linear sequence produces deliverable "increments" of the software
- The first increment is often a *core product*
- The core product is used by the customer (or undergoes detailed evaluation)
- Based on evaluation results, a plan is developed for the next increment

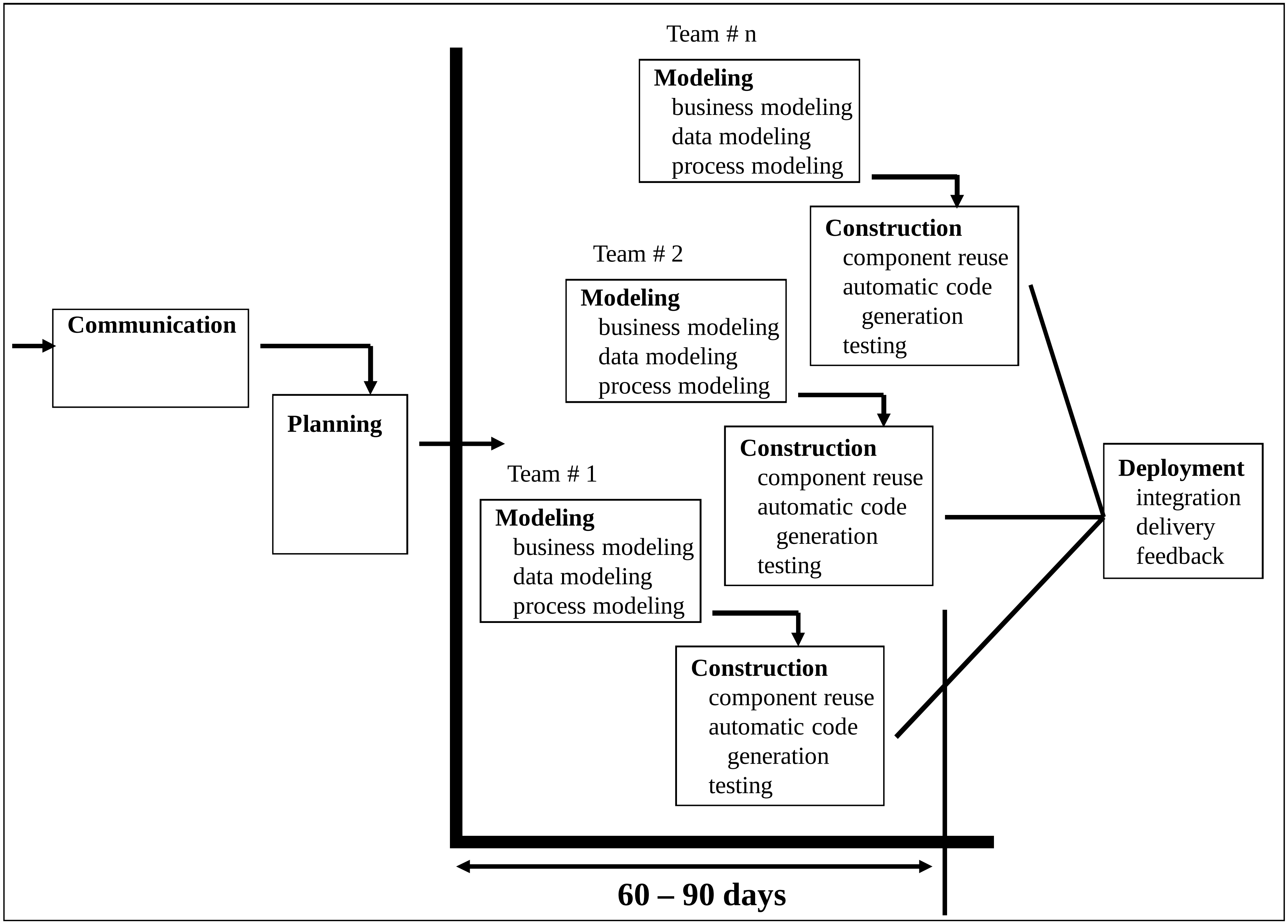
Incremental Process Models (3)

- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature
- But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment
- Particularly useful when
  - Staffing is unavailable
- Increments can be planned to manage technical risks

The RAD Model (1)

- Rapid Application Development
- Emphasizes a short development cycle
- A “high speed” adaptation of the waterfall model
- Uses a component-based construction approach
- May deliver software within a very short time period (e.g. , 60 to 90 days) if requirements are well understood and project scope is constrained

The RAD Model (2)



The RAD Model (3)

- The time constraints imposed on a RAD project demand “scalable scope”
- The application should be modularized and addressed by separate RAD teams
- Integration is required

The RAD Model - Drawbacks

- For large, but scalable projects, RAD requires sufficient human resources
- RAD projects will fail if developers and customers are not committed to the rapid-fire activities
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic
- If high performance is an issue, and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work
- RAD may not be appropriate when technical risks are high

Evolutionary Process Models

- Software, like all complex systems, evolves over a period of time
- Business and product requirements often change as development proceeds, making a straight-line path to an end product is unrealistic
- Evolutionary models are iterative

Prototyping

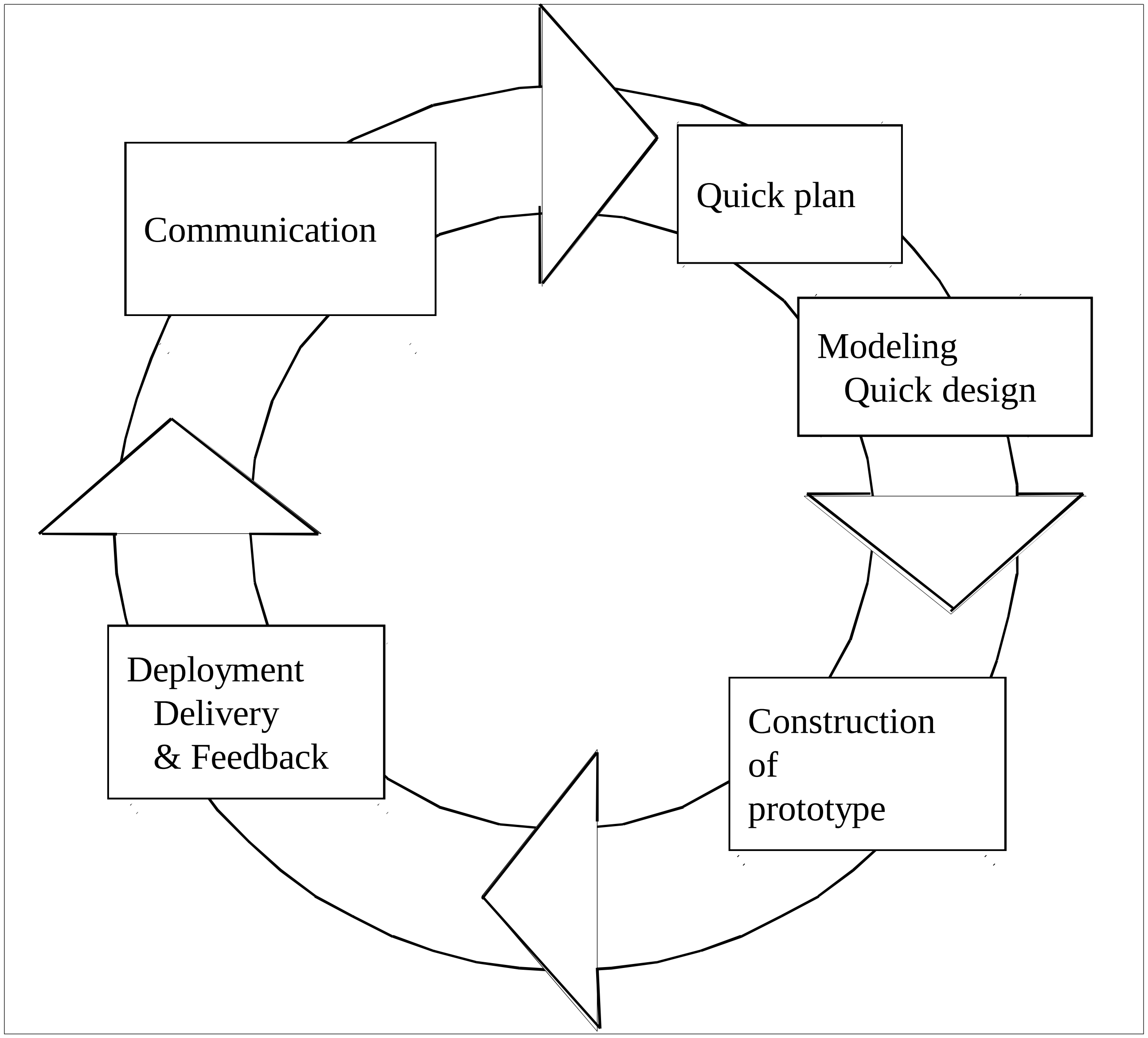


Fig 3.4: The prototyping model

Prototyping - Problems

- Customers may press for immediate delivery of working but inefficient products
- The developer often makes implementation compromises in order to get a prototype working quickly

The Spiral Model (1)

- Couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model
- It provides the potential for rapid development of increasingly more complete versions of the software
- It is a *risk-driven process model* generator
- It has two main distinguishing features
  - Cyclic approach
    - Incrementally growing a system’s degree of definition and implementation while decreasing its degree of risk
  - A set of *anchor point milestones*
    - A combination of work products and conditions that are attained along the path of the spiral

The Spiral Model (2)

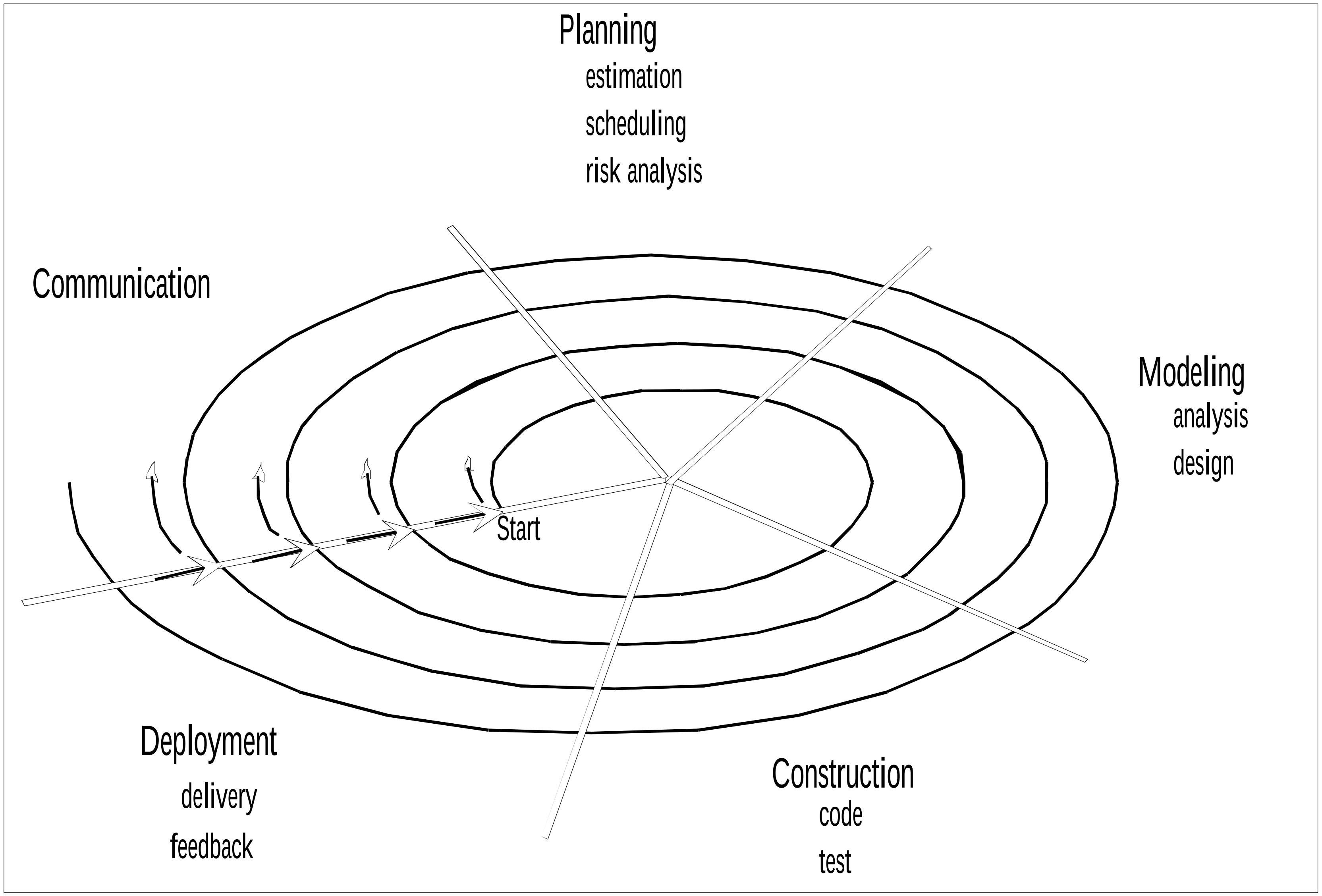


Fig 3.5: A typical spiral model

The Spiral Model (3)

- Unlike other process models that end when software is delivered, the spiral model can be adapted to apply throughout the life of the computer s/w
- The circuits around the spiral might represent
  - Concept development project
  - New Product development project
  - Product enhancement project
- The spiral model demands a direct consideration of technical risks at all stages of the project

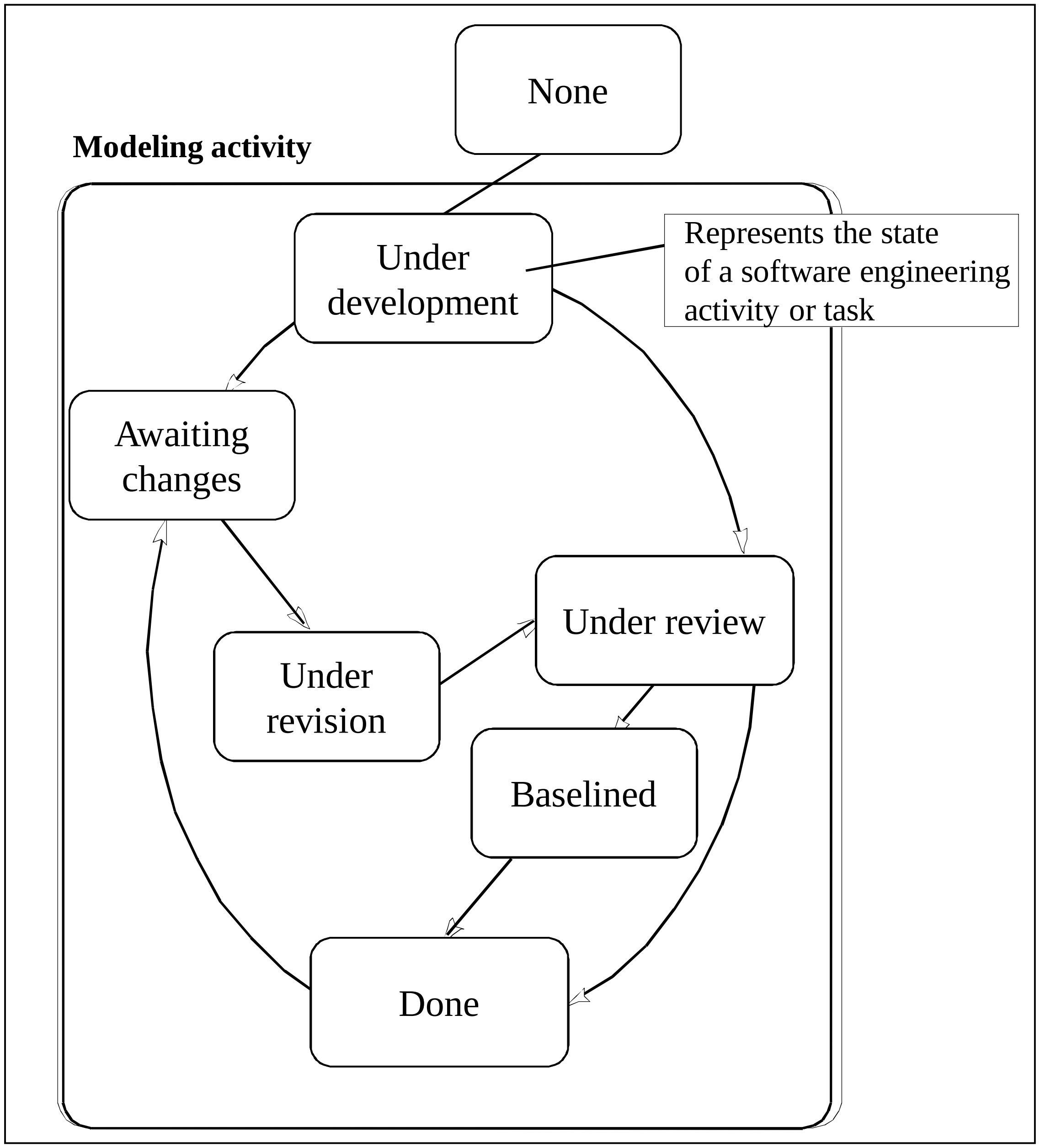
The Spiral Model - Drawbacks

- It may be difficult to convince customers (particularly in contract situations) that the evolutionary approach is controllable.
- It demands considerable risk assessment expertise and relies on this expertise for success. If a major risk is not uncovered and managed, problems will undoubtedly occur.

The Concurrent Development Model (1)

- Sometimes called *concurrent engineering*
- Can be represented schematically as a series of framework activities, s/w engineering actions and tasks, and their associated states
- Defines a series of events that will trigger transitions from state to state for each of the s/w engineering activities, actions, or tasks
- Applicable to all types of s/w development
- Defines a network of activities
- Events generated at one point in the process network trigger transitions among the states

The Concurrent Development Model (2)



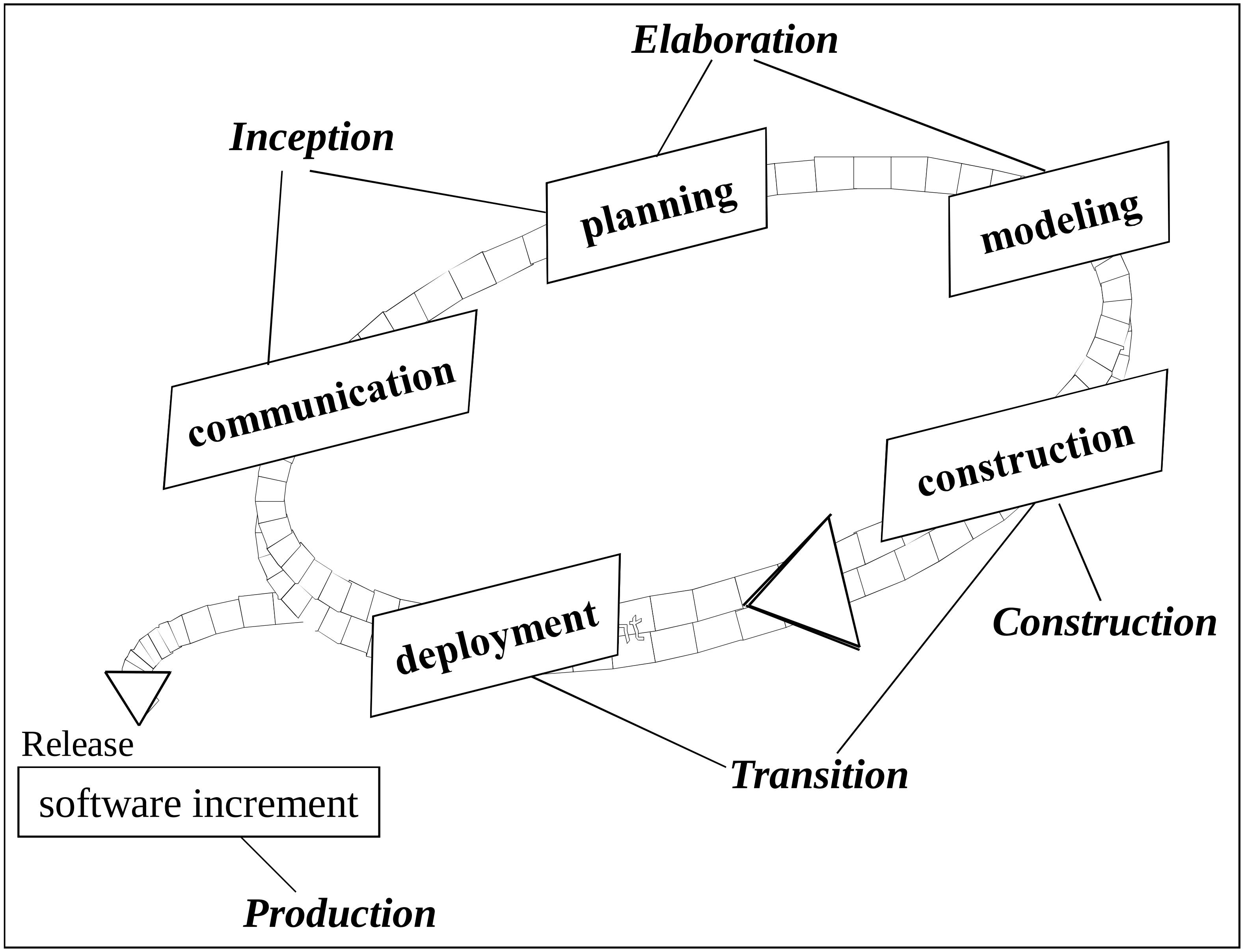
Weaknesses of Evolutionary Process Models

- Uncertainty in the number of total cycles required
  - Most project management and estimation techniques are based on linear layouts of activities
- Do not establish the maximum speed of the evolution
- Software processes should be focused on flexibility and extensibility rather than on high quality, which sounds scary
  - However, we should prioritize the speed of the development over zero defects. **Why?**

The Unified Process (UP)

- It is a use-case driven, architecture-centric, iterative and incremental software process
- UP is an attempt to draw on the best features and characteristics of conventional s/w process models
- Also implements many of the best principles of **agile software development**
- UP is a framework for object-oriented software engineering using UML (Unified Modeling Language)

Phases of the Unified Process



Phases of UP - Inception

- Encompasses both customer communication and planning activities
- Fundamental business requirements are described through a set of preliminary use-cases
  - A **use-case** describes a sequence of actions that are performed by an actor (e.g., a person, a machine, another system) as the actor interacts with the software
- A rough architecture for the system is also proposed

Phases of UP - Elaboration

- Encompasses customer communication and modeling activities
- Refines and expands the preliminary use-cases
- Expands the architectural representation to include five different views of the software
  - The use-case model
  - The analysis model
  - The design model
  - The implementation model
  - The deployment model
- In some cases, elaboration creates an “executable architectural baseline” that represents a “first cut” executable system

Phases of UP - Construction

- Makes each use-case operational for end-users
- As components are being implemented, **unit tests** are designed and executed for each
- Integration activities (component assembly and **integration testing**) are conducted
- Use-cases are used to derive a suite of **acceptance tests**

Phases of UP - Transition

- Software is given to end-users for **beta testing**
- The software team creates the necessary support information –
  - User manuals
  - Trouble-shooting guides
  - Installation procedures
- At the conclusion of the transition phase, the software increment becomes a usable software release

Phases of UP - Production

- Coincides with the deployment activity of the generic process
- The on-going use of the software is monitored
- Support for the operating environment (infrastructure) is provided
- Defect reports and requests for changes are submitted and evaluated