Chapter 3
Prescriptive Process Models

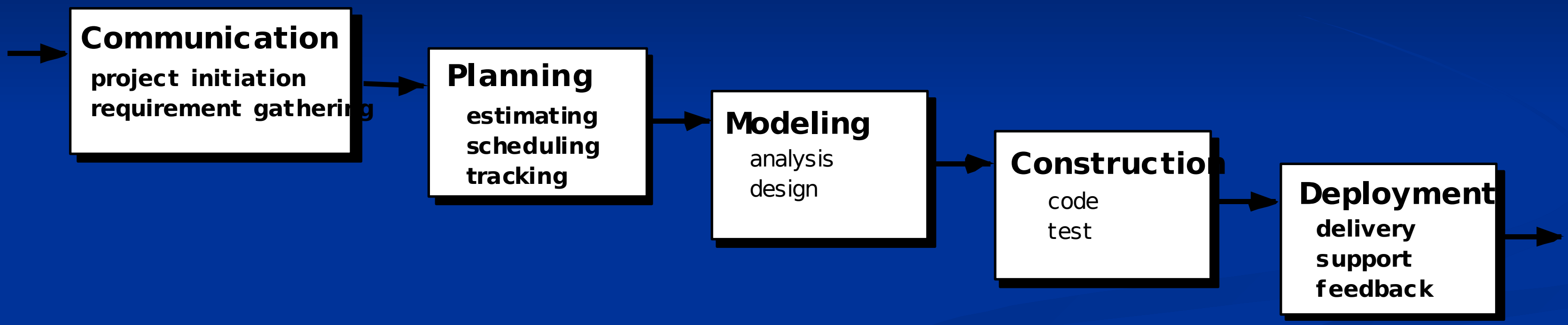
Prescriptive Models

- 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



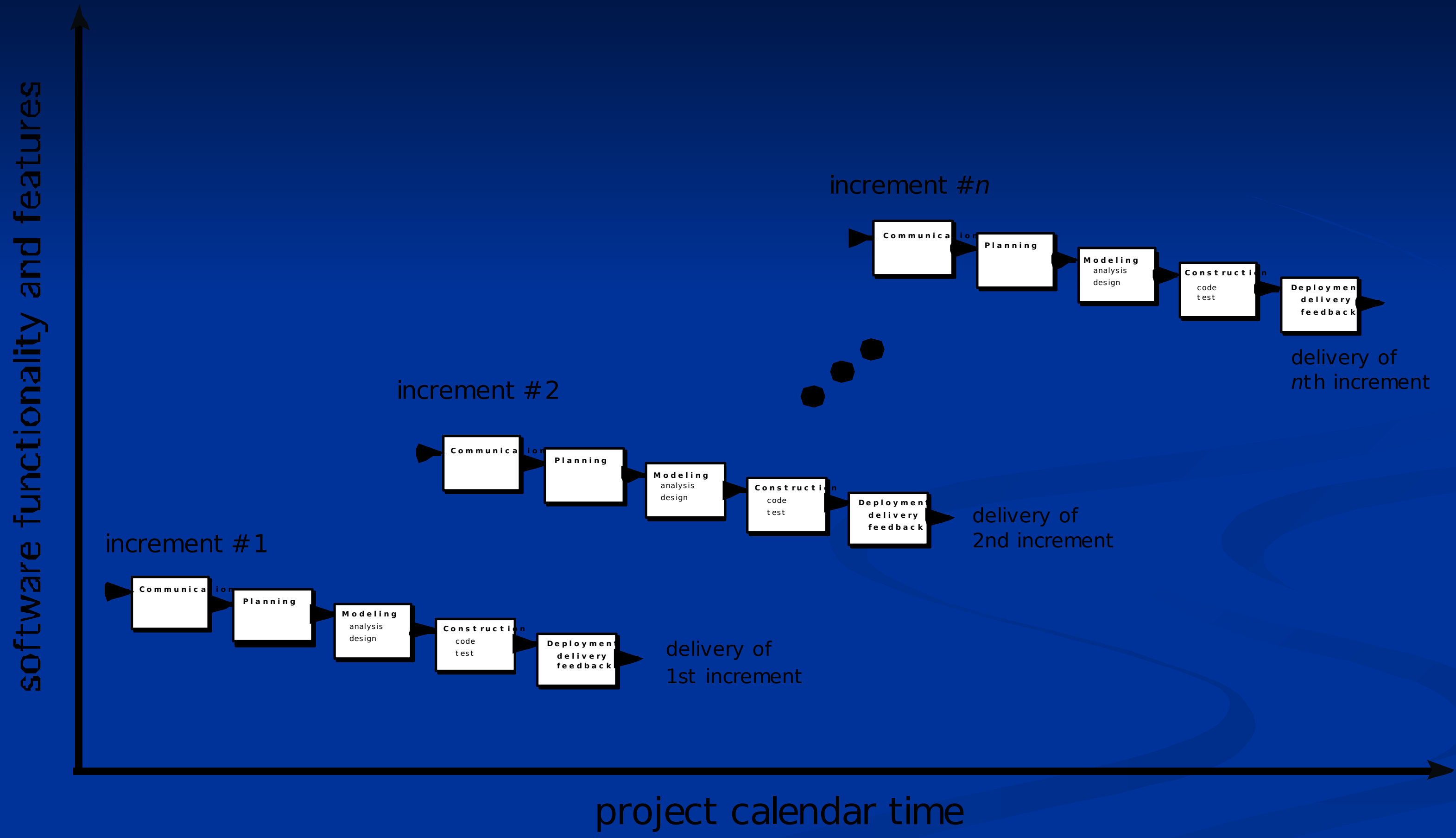
Water Fall Model

- Requirements for a problem are well understood
- Workflows from communication through deployment in a linear manner.
- Requirements are reasonably stable.
- Classic life cycle suggests a systematic sequential approach to s/w development.

Problems with waterfall model

- Real projects rarely follow the sequential flow that the model proposes.
- Changes can cause confusion . Leads to blocking states
- It is difficult for the customer to state all requirements explicitly.
- Customer must have patience . Working version of application available late in project time span.
- Major blunder if undetected can be disastrous.

The Incremental Model



Incremental Model

- When initial requirements are well defined .
- Need to provide a limited set of s/w functionality quickly & then refine & expand in later releases.
- Model combines elements of linear and parallel process flows.
- Applies linear sequence in staggered fashion.
- E.g word processing s/w.

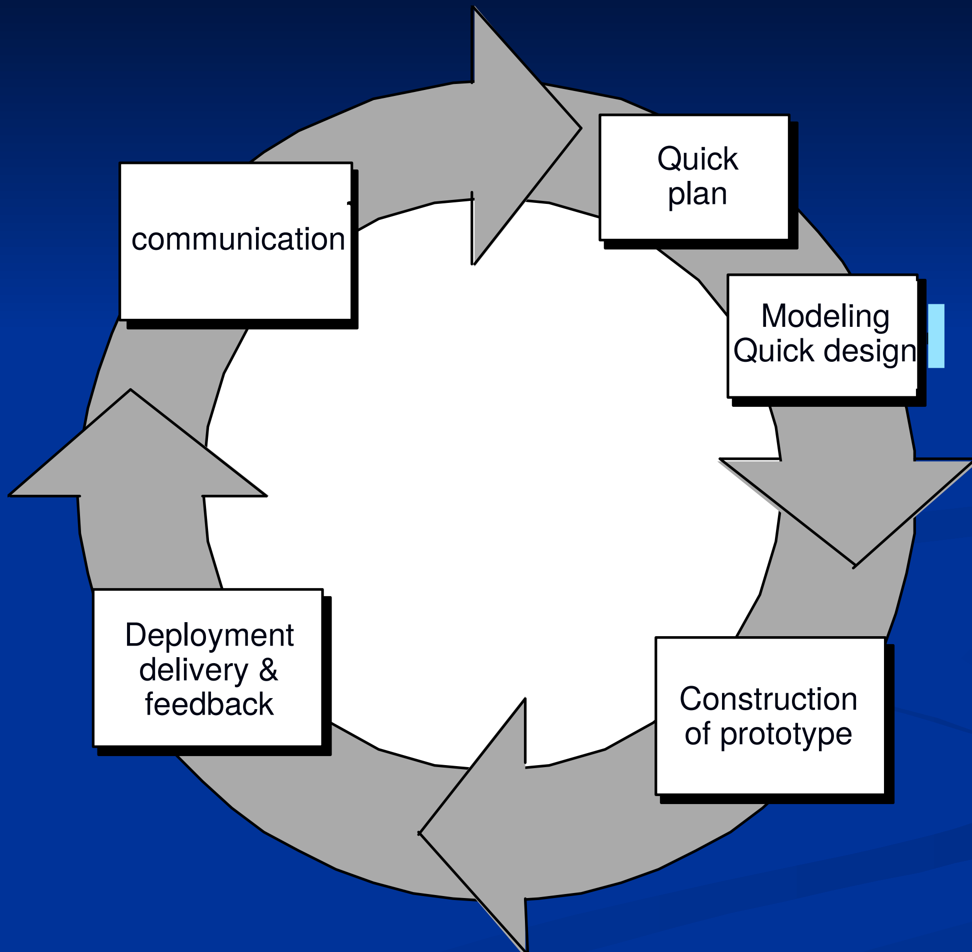
Incremental Model

- The first increment is often core product.
- As a result of use a plan is developed for the next increment.
- Focuses on the delivery of an operational product with each increment.
- Useful when staffing is unavailable for complete implementation.

Evolutionary Models:

- s/w evolves over a period of time . Business & product requirements change .
- Evolutionary models are iterative .
- They are characterized in a manner that enables you to develop increasingly more complex versions of the s/w
- Prototyping model
- Spiral model.

Evolutionary Models: Prototyping



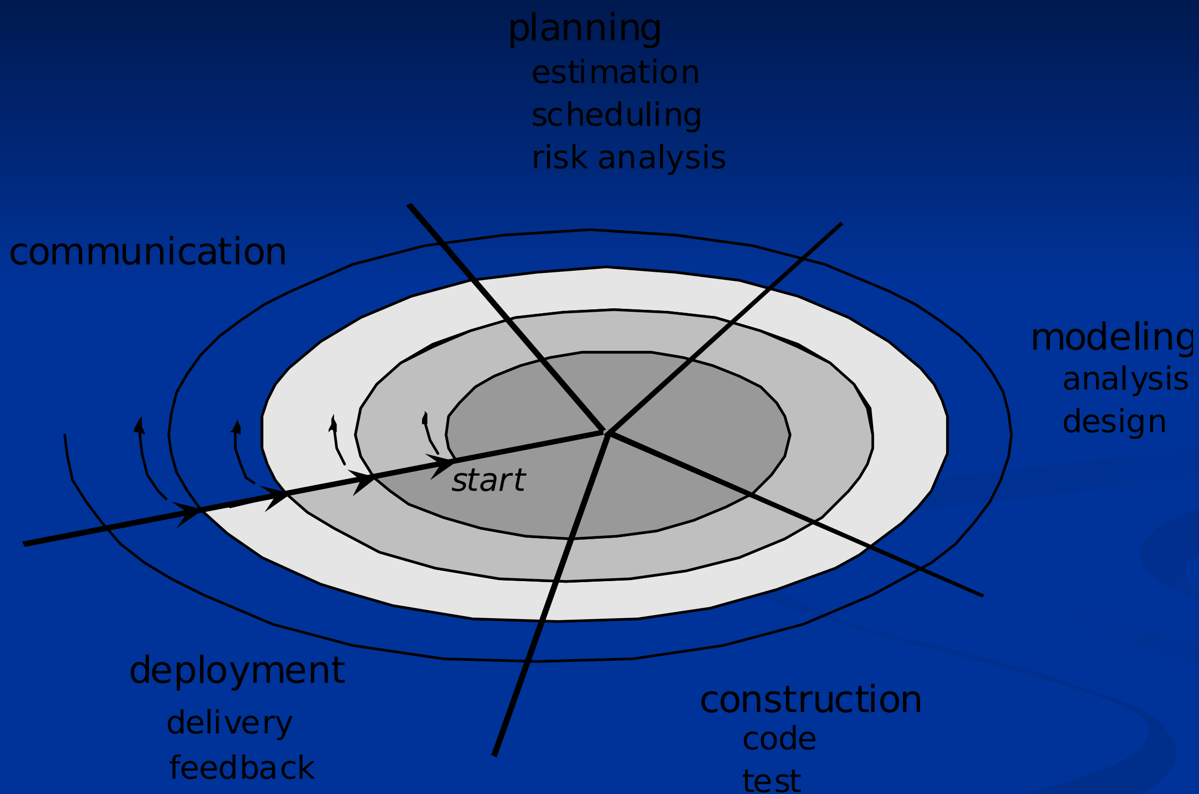
Evolutionary Models: Prototyping

- Customer defines a set of general objectives or developer unsure about efficiency of an algorithm.
- Prototyping assists to better understand what is to be built when requirements are fuzzy.
- What do you do with the prototype?
- Some prototypes are built as throwaways.
- Others evolve into actual systems.

Prototyping Problems

- Stakeholders see what appears to be a working version of the s/w. Hence not ready to wait for rebuilt product.
- Stakeholders want quick fixes to be applied to prototype.
- As a s/w engg you make implementation compromises in order to get the prototype to work quickly.
- Hence compromise on the quality of final product

Evolutionary Models: The Spiral



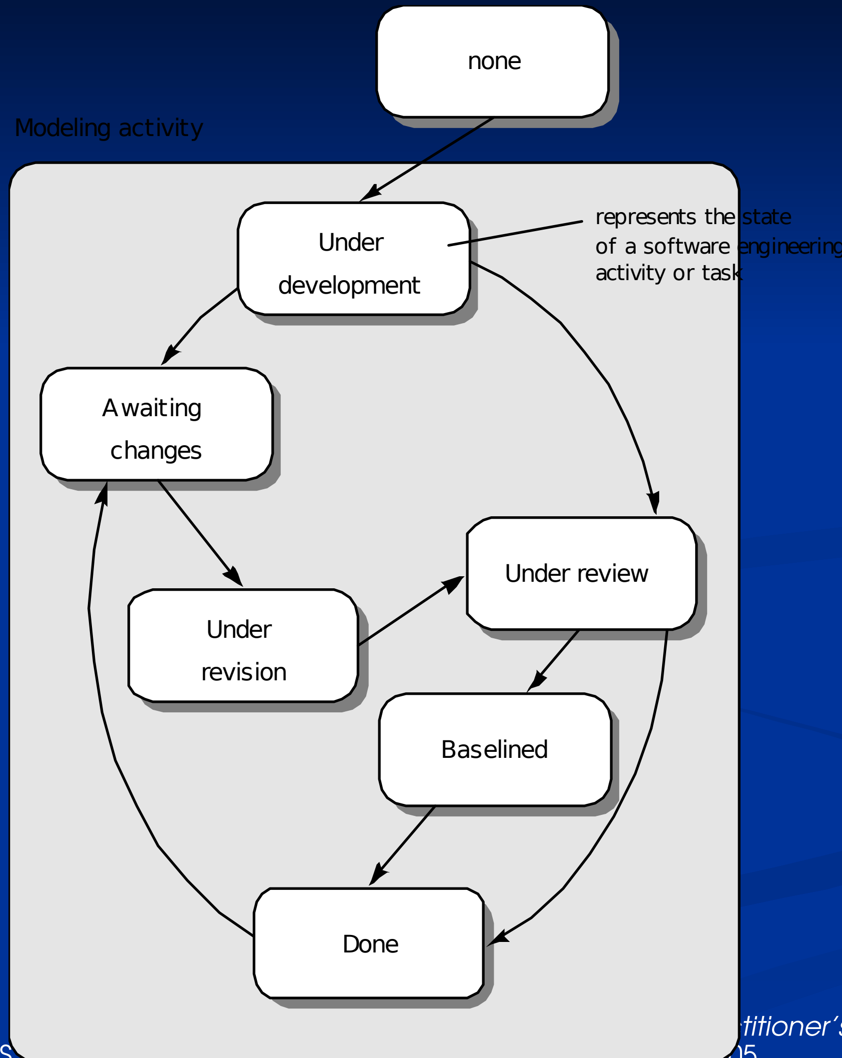
Evolutionary Models: The Spiral

- Spiral model couples the iterative nature of prototyping with the controlled & systematic aspects of waterfall model.
- S/w is developed in a series of evolutionary releases.
- Since s/w evolves as the process progresses , the developer & customer better understand & react to risks at each level.

The Spiral Model Problems

- It is difficult to convince customers that the evolutionary approach is controllable.
- Demands risk assessment expertise & relies on this expertise for success.

Concurrent Model



Concurrent Model

- Allows s/w team to represent iterative & concurrent elements of any of the process models.

- All s/w engineering activities exist concurrently but reside in different states.

- E.g early in a project the communication activity has completed its first iteration & exists in the ***awaiting changes*** *state*.

- The modeling activity which existed in the inactive state while initial communication was completed , now makes a transition into the ***under development*** state

- Concurrent modeling defines a series of events that will trigger transitions from state to state for each of the S.E activities.

- Weakness of evolutionary Process

- 1. Poses a problem to project planning because of uncertain number of cycles required to construct the product.

- 2. Does not establish the maximum speed of the evolution.

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)

Component-based development

- Component-based software engineering (CBSE) is an approach to software development that relies on software reuse.
- It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific.
- Components are more abstract than object classes and can be considered to be stand-alone service providers.

Component based Development

- Available component-based products are researched & evaluated for the application domain.
- Component integration issues are considered.
- A software architecture is designed to accommodate the components.
- Components are integrated into the architecture.
- Comprehensive testing is conducted to ensure proper functionality.

CBSE essentials

- Independent components specified by their interfaces.
- Component standards to facilitate component integration.
- Middleware that provides support for component inter-operability.
- A development process that is geared to reuse.

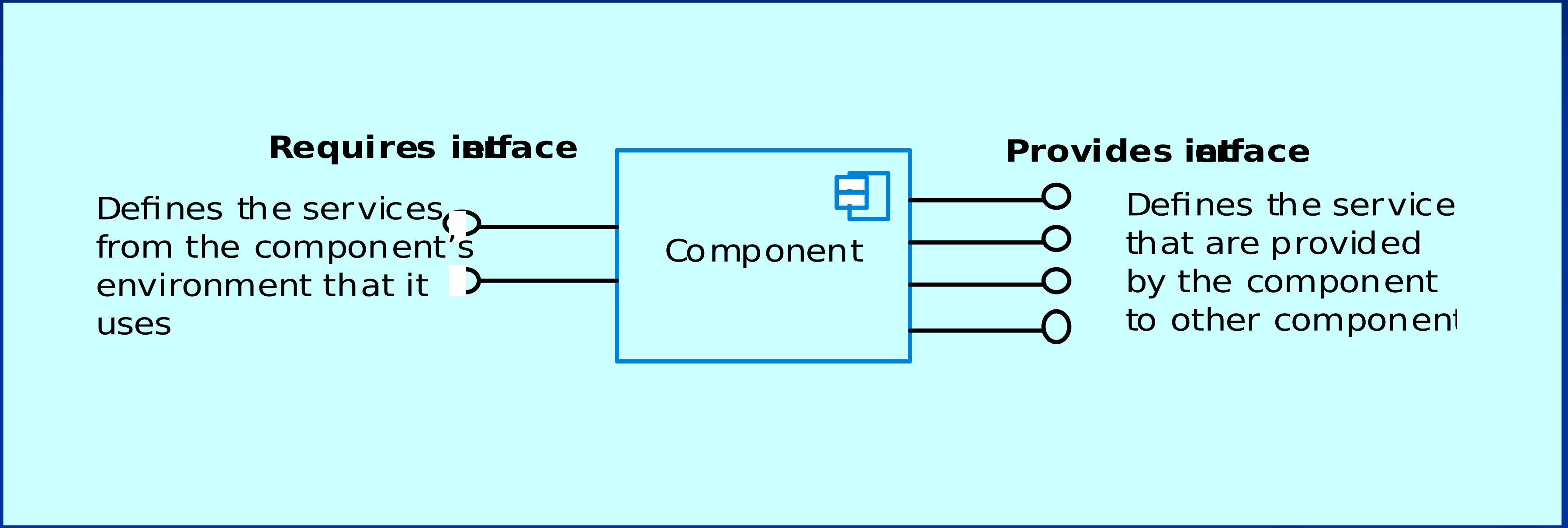
CBSE and design principles

- Apart from the benefits of reuse, CBSE is based on sound software engineering design principles:
  - Components are independent so do not interfere with each other;
  - Component implementations are hidden;
  - Communication is through well-defined interfaces;
  - Component platforms are shared and reduce development costs.

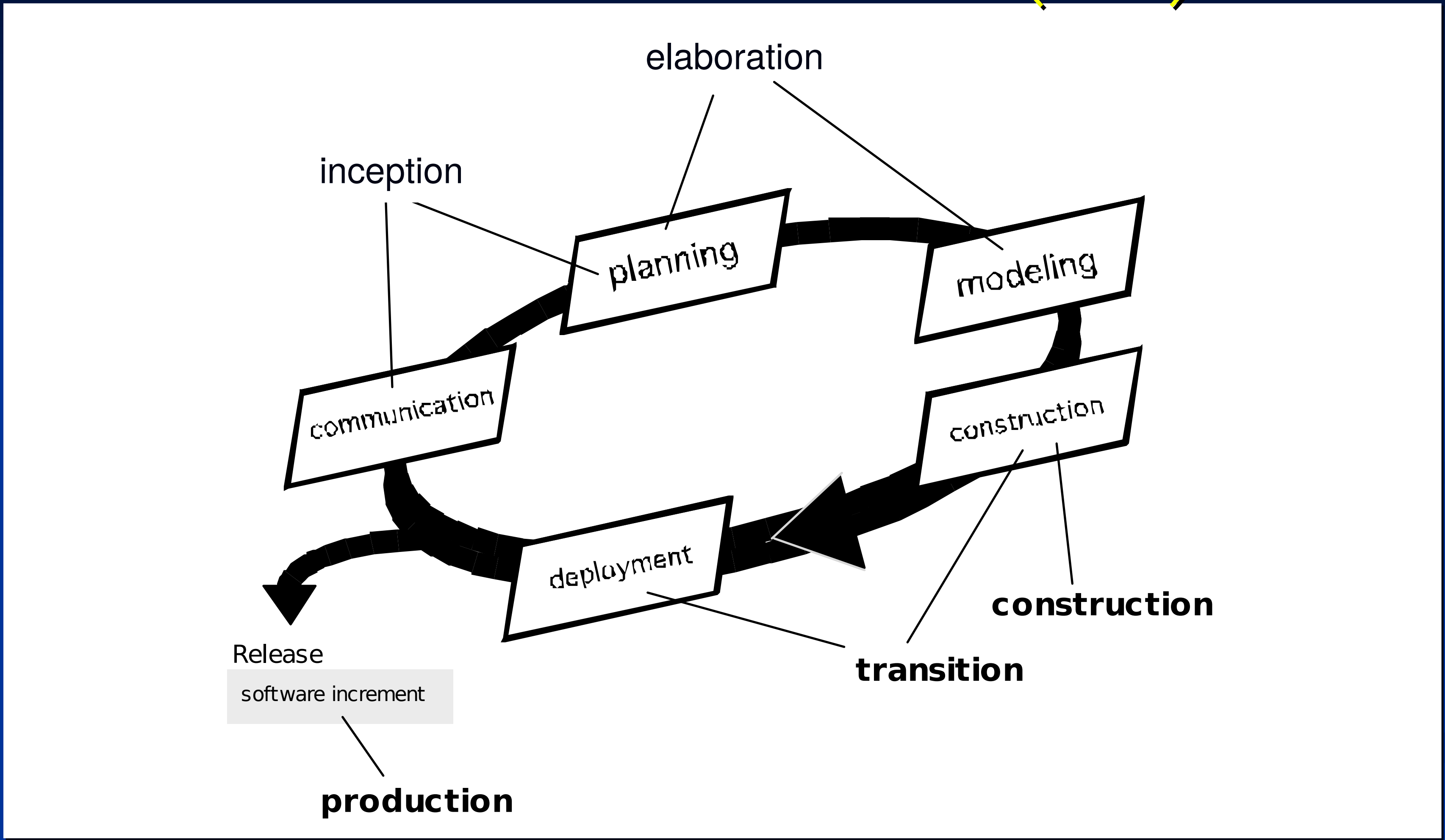
CBSE problems

- Component trustworthiness - how can a component with no available source code be trusted?
- Component certification - who will certify the quality of components?
- Emergent property prediction - how can the emergent properties of component compositions be predicted?
- Requirements trade-offs - how do we do trade-off analysis between the features of one component and another?

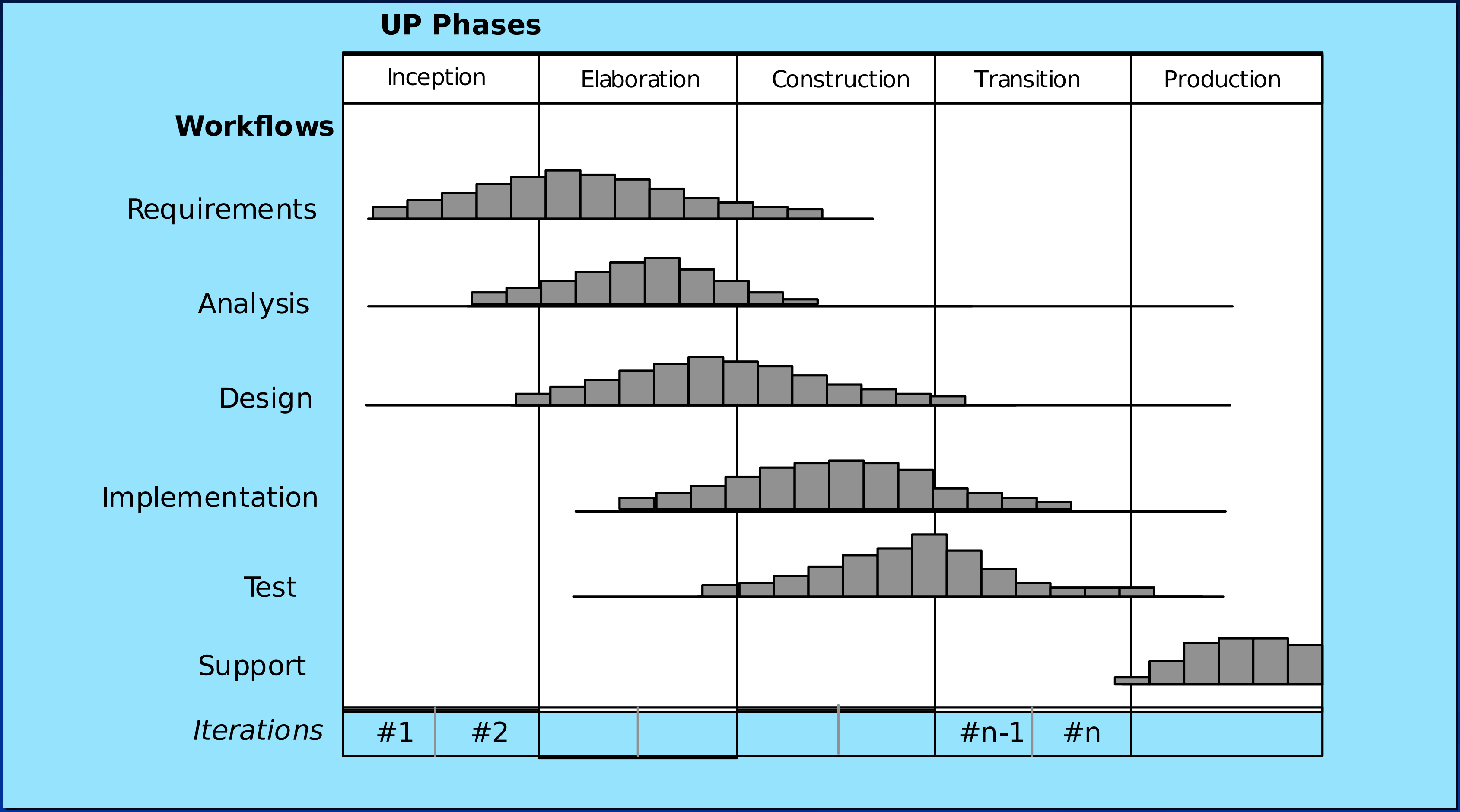
Component interfaces



The Unified Process (UP)



UP Phases



UP Phases
Inception
Elaboration
Construction
Transition
Production
Workflows
Requirements
Analysis
Design
Implementation
Test
Support
Iterations
#1
#2
#n-1
#n

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

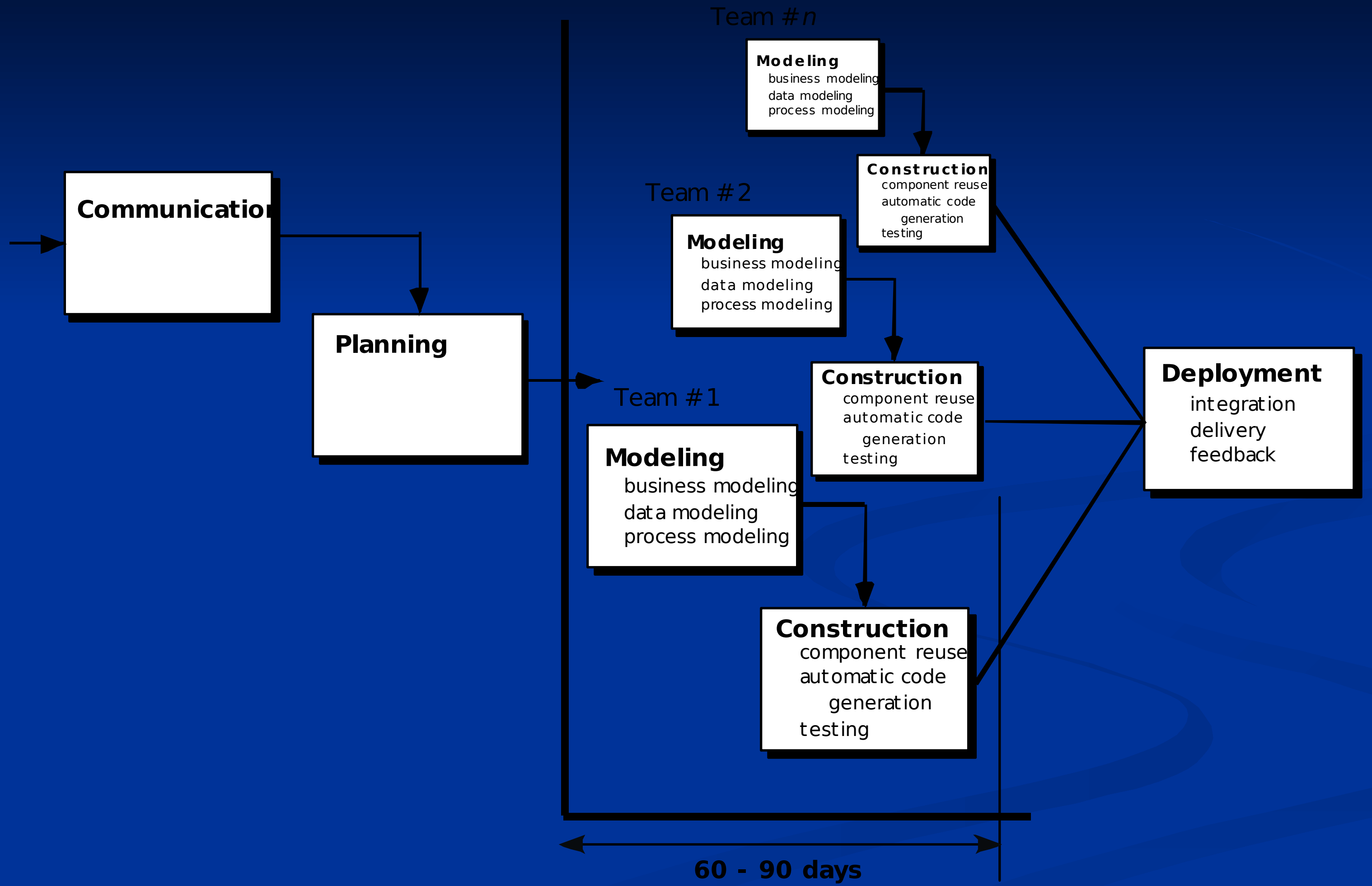
Construction phase

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

The RAD Model



- Formal Specification Language: is composed of 3 primary components.
- A syntax that defines the specific notation with which the specification is represented.
- A semantics to help define a "universe of objects" that will be used to describe the system.
- A set of relations that define the rules that indicate which objects properly satisfy the specification.
- E.g OCL

Formal Methods Model

- Encompasses a set of activities that leads to formal mathematical specification of computer s/w.
- It enables you to specify , develop, & verify a computer – based system by applying a rigorous , mathematical notation.
- Ambiguity , incompleteness & inconsistency can be discovered & corrected more easily.