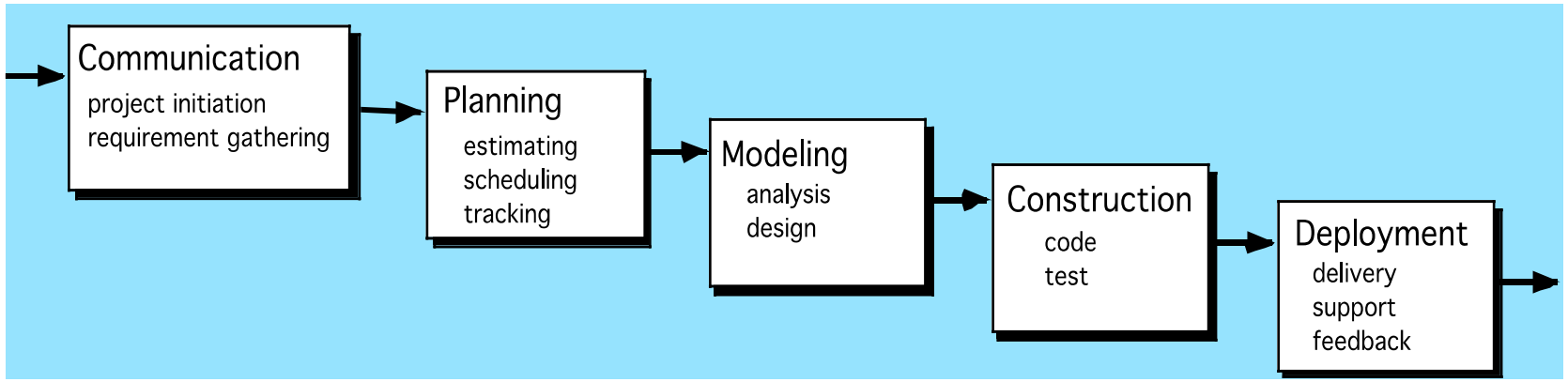
Software Life Cycle models

The Waterfall Model



It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.

(problems: 1. rarely linear, iteration needed. 2. hard to state all requirements explicitly. Blocking state. 3. code will not be released until very late.)

The classic life cycle suggests a systematic, sequential approach to software development.

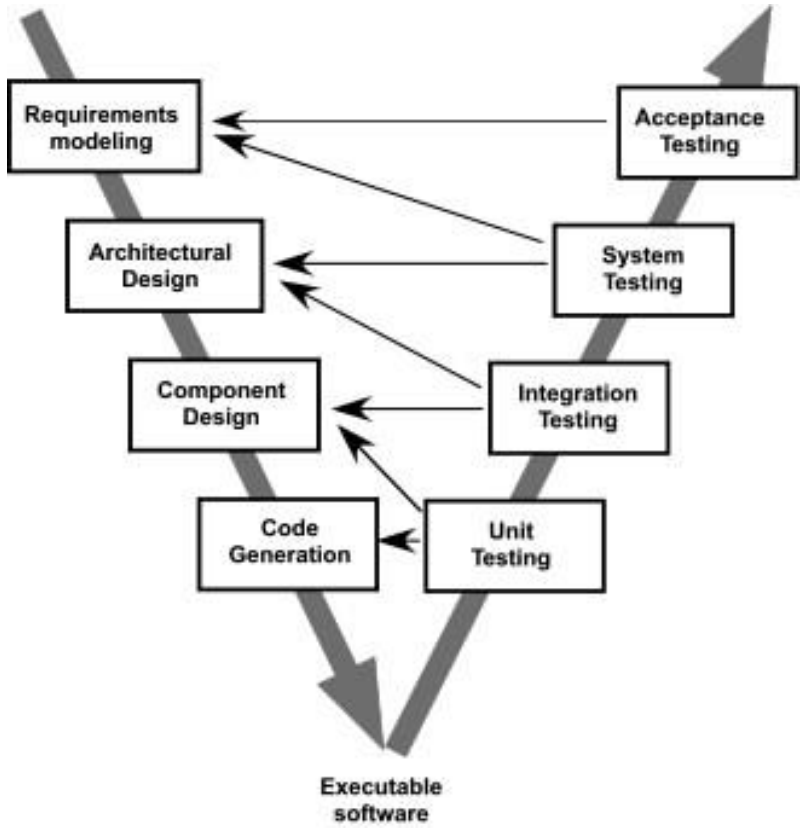
Advantages

- This model is very simple and is easy to understand.
- Phases in this model are processed one at a time.
- Each stage in the model is clearly defined.
- This model has very clear and well undestood milestones.
- Process, actions and results are very well documented.
- Reinforces good habits: define-before- design, design-before-code.
- This model works well for smaller projects and projects where requirements are well understood.

Drawbacks

- The model expects complete & accurate requirements early in the process, which is unrealistic
- It is difficult to define all requirements at the beginning of a project
- This model is not suitable for accommodating any change
- A working version of the system is not seen until late in the project's life
- It does not scale up well to large projects.
- Real projects are rarely sequential.

The V-Model(Verification and Validation model)



- **Verification:** It involves static analysis technique (review) done without executing code. It is the process of evaluation of the product development phase to find whether specified requirements meet.
- **Validation:** It involves dynamic analysis technique (functional, non-functional), testing done by executing code. Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.
- So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation phases are joined by coding phase in V-shape. Thus it is called V-Model.

The V-Model(Verification and Validation model)

- A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

- Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

Principles of V Model

- **Large to Small:** In V-Model, testing is done in a hierarchical perspective, For example, requirements identified by the project team, create High-Level Design, and Detailed Design phases of the project. As each of these phases is completed the requirements, they are defining become more and more refined and detailed.
- **Data/Process Integrity:** This principle states that the successful design of any project requires the incorporation and cohesion of both data and processes. Process elements must be identified at each and every requirements.
- **Scalability:** This principle states that the V-Model concept has the flexibility to accommodate any IT project irrespective of its size, complexity or duration.

- **Cross Referencing:** Direct correlation between requirements and corresponding testing activity is known as cross-referencing.
- **Tangible Documentation:** This principle states that every project needs to create a document. This documentation is required and applied by both the project development team and the support team. Documentation is used to maintaining the application once it is available in a production environment.

Advantages

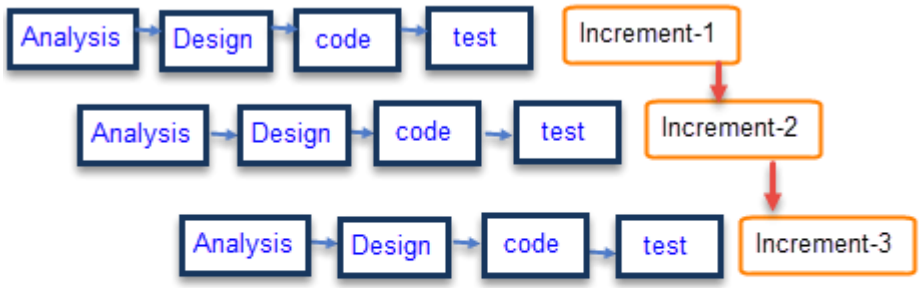
- This is a highly disciplined model and Phases are completed one at a time.
- V-Model is used for small projects where project requirements are clear.
- Simple and easy to understand and use.
- This model focuses on verification and validation activities early in the life cycle thereby enhancing the probability of building an error-free and good quality product.
- It enables project management to track progress accurately.

Disadvantages

- High risk and uncertainty.
- It is not a good for complex and object-oriented projects.
- It is not suitable for projects where requirements are not clear and contains high risk of changing.
- This model does not support iteration of phases.
- It does not easily handle concurrent events.

Incremental model

- Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle.
- Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.

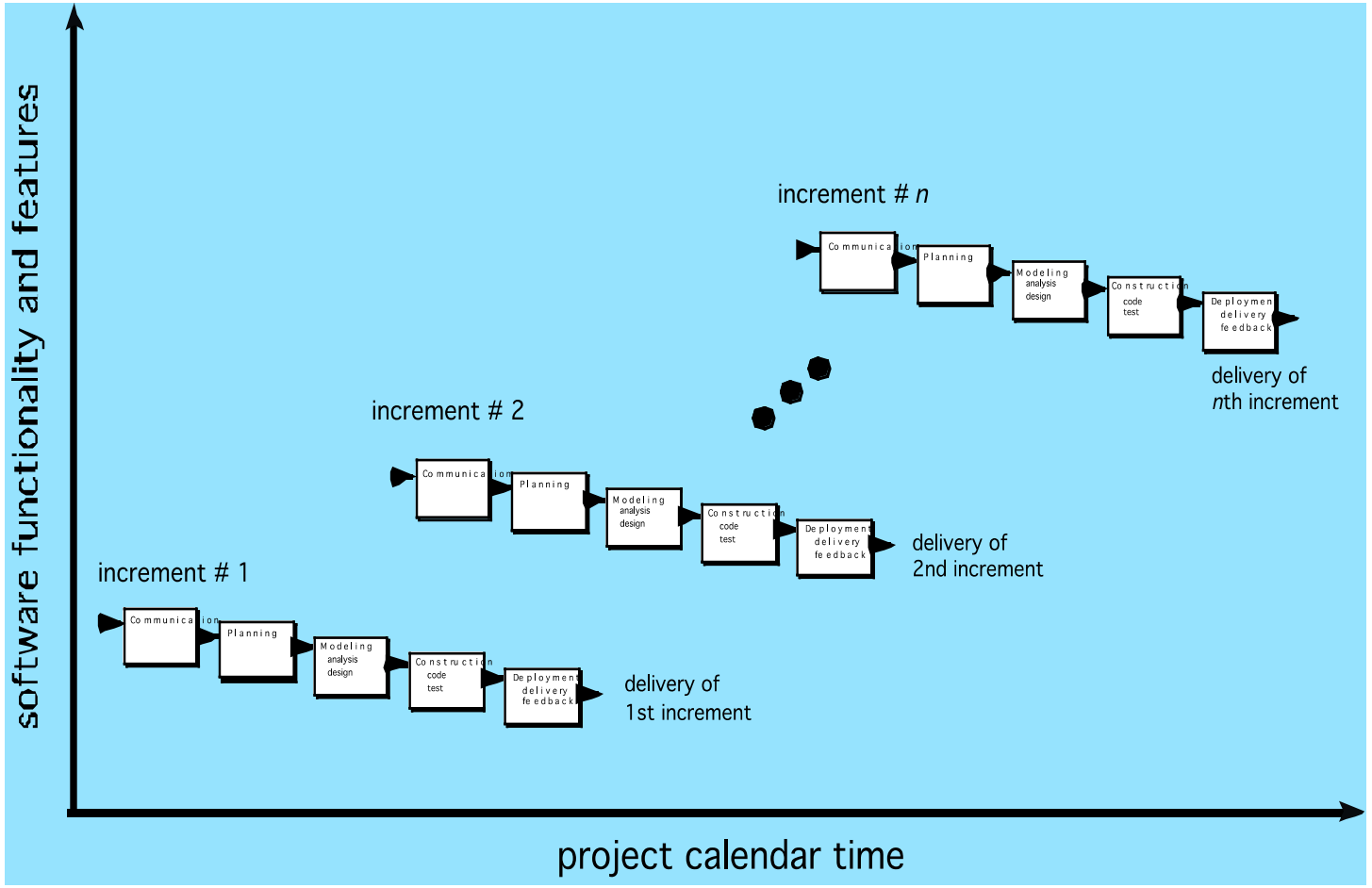


Incremental Model

- Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.
- The system is put into production when the first increment is delivered.
- The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments.
- Once the core product is analyzed by the client, there is plan development for the next increment.

- When initial requirements are reasonably well defined, but the overall scope of the development effort precludes a purely linear process.
  - A compelling need to expand a limited set of new functions to a later system release.
- It combines elements of linear and parallel process flows.
  - Each linear sequence produces deliverable increments of the software.
- The first increment is often a core product with many supplementary features.
  - Users use it and evaluate it with more modifications to better meet the needs.

The Incremental Model



Characteristics of Incremental model

- System development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are frozen

Advantages

- The software will be generated quickly
- It is flexible and less expensive to change requirements and scope
- Throughout the development stages changes can be done
- This model is less costly compared to others
- A customer can respond to each building
- Errors are easy to be identified

Disadvantages

- It requires a good planning designing
- Problems might arise due to system architecture because not all requirements collected up front for the entire software lifecycle
- Each iteration phase is rigid and does not overlap each other
- Rectifying a problem in one unit requires correction in all the units and consumes a lot of time

Incremental model can be used when…

- Requirements of the system are clearly understood
- demand for an early release of a product arises
- software engineering team are not very well skilled or trained
- high-risk features and goals are involved
- Development is web application and happens for product based companies

Evolutionary Models

- Software system evolves over time as requirements often change as development proceeds. Thus, a straight line to a complete end product is not possible. However, a limited version must be delivered to meet competitive pressure.
- Usually a set of core product or system requirements is well understood, but the details and extension have yet to be defined.
- You need a process model that has been explicitly designed to accommodate a product that evolved over time.
- It is iterative that enables you to develop increasingly more complete version of the software.
- Two types are introduced, namely Prototyping and Spiral models.

The Prototype model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives fo
- It does not identify the requirements like detaile
- It is software working model of limited functiona
- In this model, working programs are quickly pro

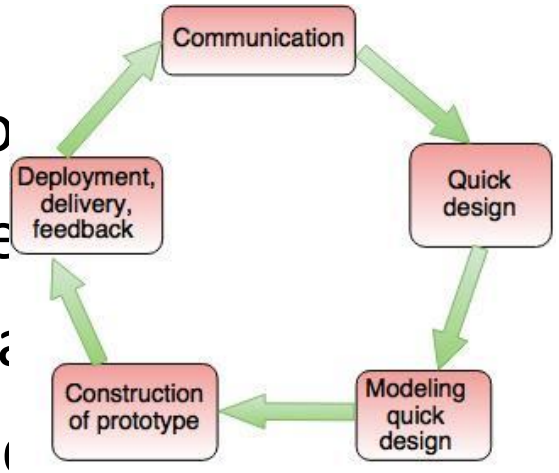


Fig. - The Prototyping Model

- **1. Communication**
  In this phase, developer and customer meet and discuss the overall objectives of the software.

  2. Quick design: Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

- **3. Modeling quick design:** This phase gives the clear idea about the development of software because the software is now built.
- It allows the developer to better understand the exact requirements.
- **4. Construction of prototype**
  The prototype is evaluated by the customer itself.

  5. Deployment, delivery, feedback: If the user is not satisfied with current prototype then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Evolutionary Models: Prototyping

- **When to use**: Customer defines a set of general objectives but does not identify detailed requirements for functions and features. Or Developer may be unsure of the efficiency of an algorithm, the form that human computer interaction should take.
- **What step**: Begins with communication by meeting with stakeholders to define the objective, identify whatever requirements are known, outline areas where further definition is mandatory. A quick plan for prototyping and modeling (quick design) occur. Quick design focuses on a representation of those aspects the software that will be visible to end users (interface and output). Design leads to the construction of a prototype which will be deployed and evaluated. Stakeholder's comments will be used to refine requirements.
- Both stakeholders and software engineers like the prototyping paradigm. Users get a feel for the actual system, and developers get to build something immediately. However, engineers may make compromises in order to get a prototype working quickly. The less-than-ideal choice may be adopted forever after you get used to it.

Advantages

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a thrown away prototype when the users are confused with it.
-

Evolutionary Models: The Spiral

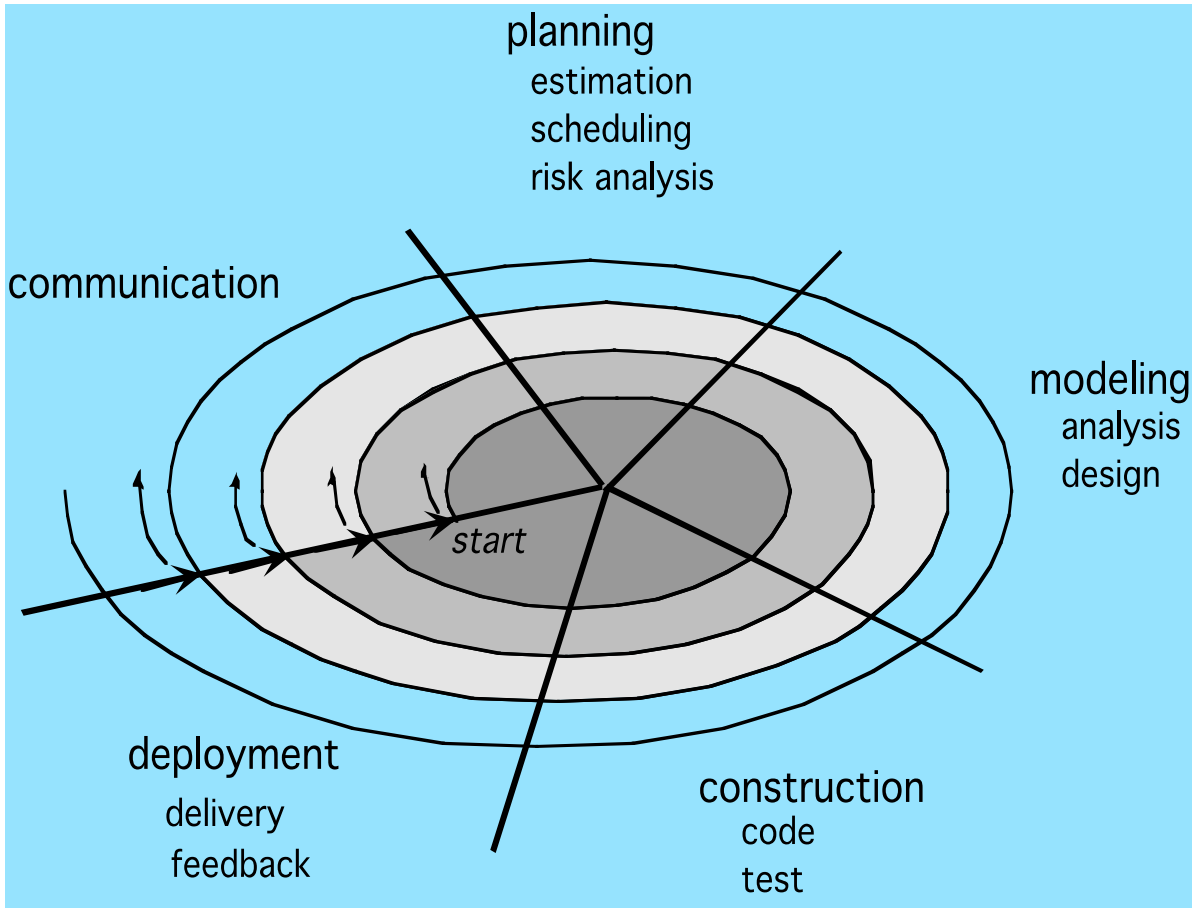
- Spiral Model is highly used in IT companies.
- This model involves strategies, which is a combination of incremental and prototype models.
- This Spiral Model is best to use for large projects which required more management and planning.

- A spiral model is a realistic approach to the development of large-scale software products because the software evolves as the process progresses.
- In addition, the developer and the client better understand and react to risks at each evolutionary level.
- The model uses prototyping as a risk reduction mechanism and allows for the development of prototypes at any stage of the evolutionary development.

- It maintains a systematic step wise approach, like the classic Life Cycle model but incorporates it into an iterative framework that more reflect the real world.
- If employed correctly, this model should reduce risks before they become problematic as consideration of technical risks are considered at all stages.

- A spiral model is divided into a set of framework activities
- Each of the framework activities represent one segment of the spiral path
- The first circuit around the spiral will result in the development of a product specification and progressively more sophisticated versions
- Unlike other development process, spiral model can be applied throughout the life of the software.

Evolutionary Models: The Spiral



Advantages

- Spiral Model mostly concentrates on risk analysis.
- Most useful for large and risky projects.
- Spiral Model can be used if requirement changes frequently.
- Focused model for all phases.
- Customer evaluation phase makes this model useful.

Disadvantages

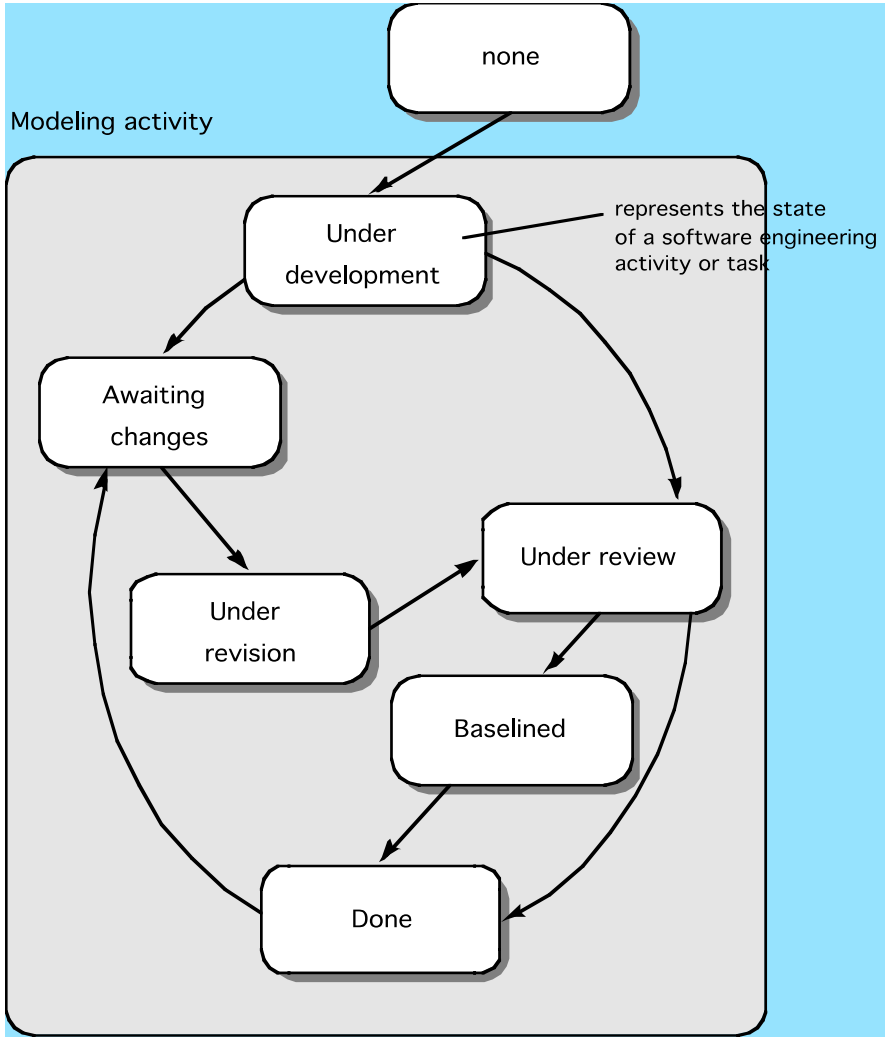
- For risk, analysis phase requires an expert person to make an analysis.
- Not useful for small projects.
- Project duration and cost could be infinite because of the spiral feature.
- Documentation could become lengthy.

Concurrent Model

- The *concurrent development model*, sometimes called *concurrent engineering*, can be represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states.

- The concurrent model is often more appropriate for system engineering projects where different engineering teams are involved.

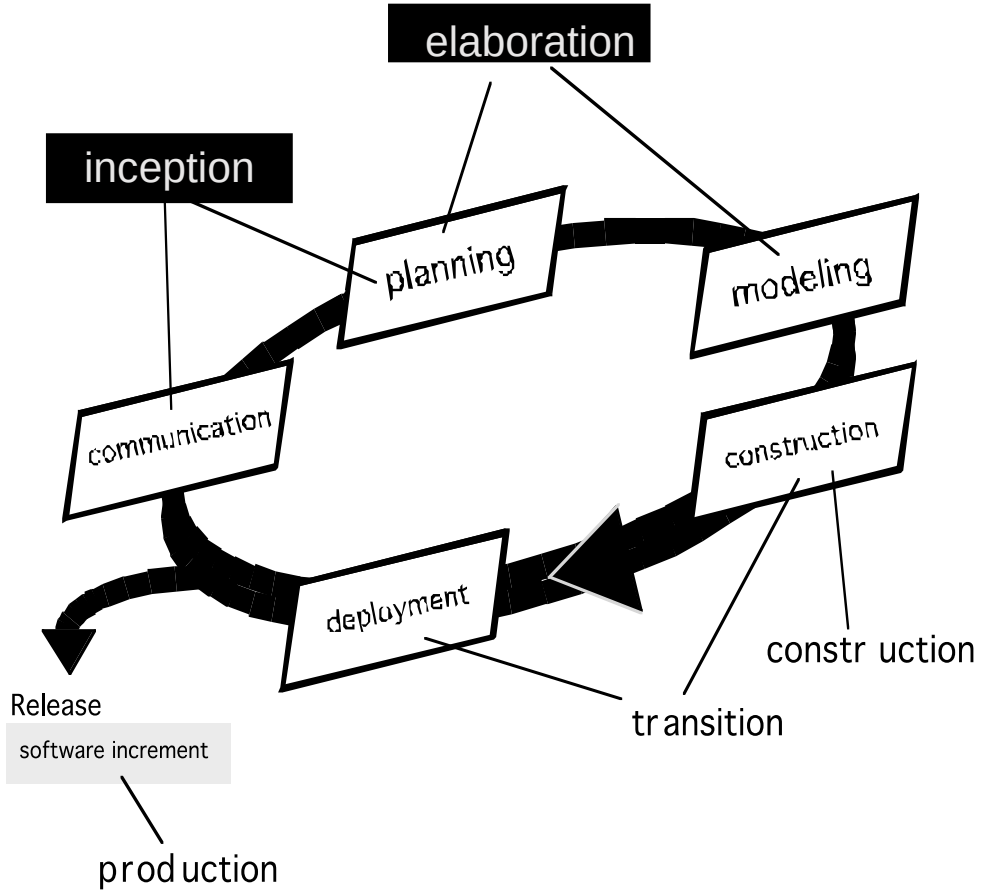
Concurrent Model



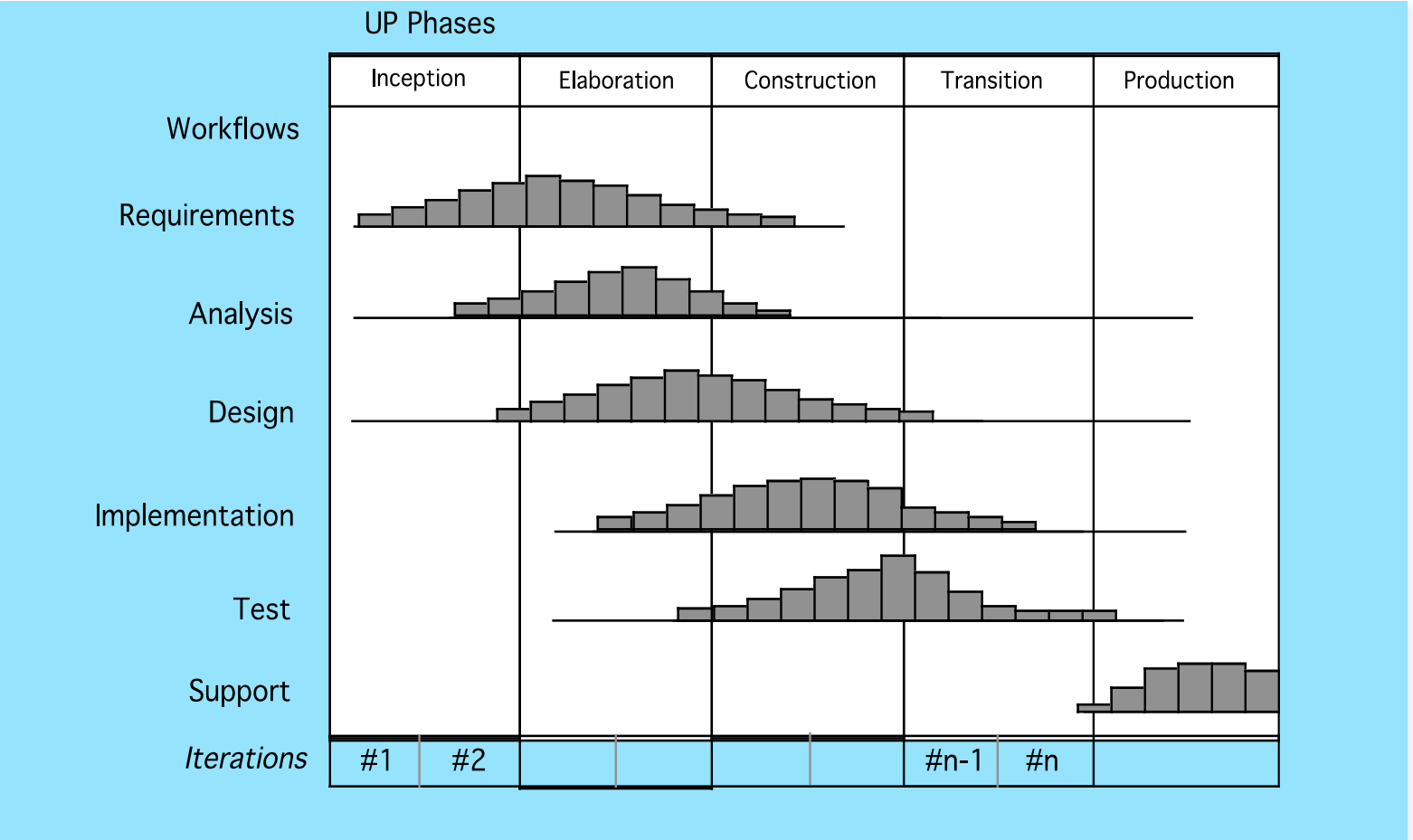
Still Other Process Models

- Component based development—the process to apply when reuse is a development objective (like spiral model)
- Formal methods—emphasizes the mathematical specification of requirements (easy to discover and eliminate ambiguity, incompleteness and inconsistency)
- Aspect Oriented software development (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) to model and develop object-oriented system iteratively and incrementally.

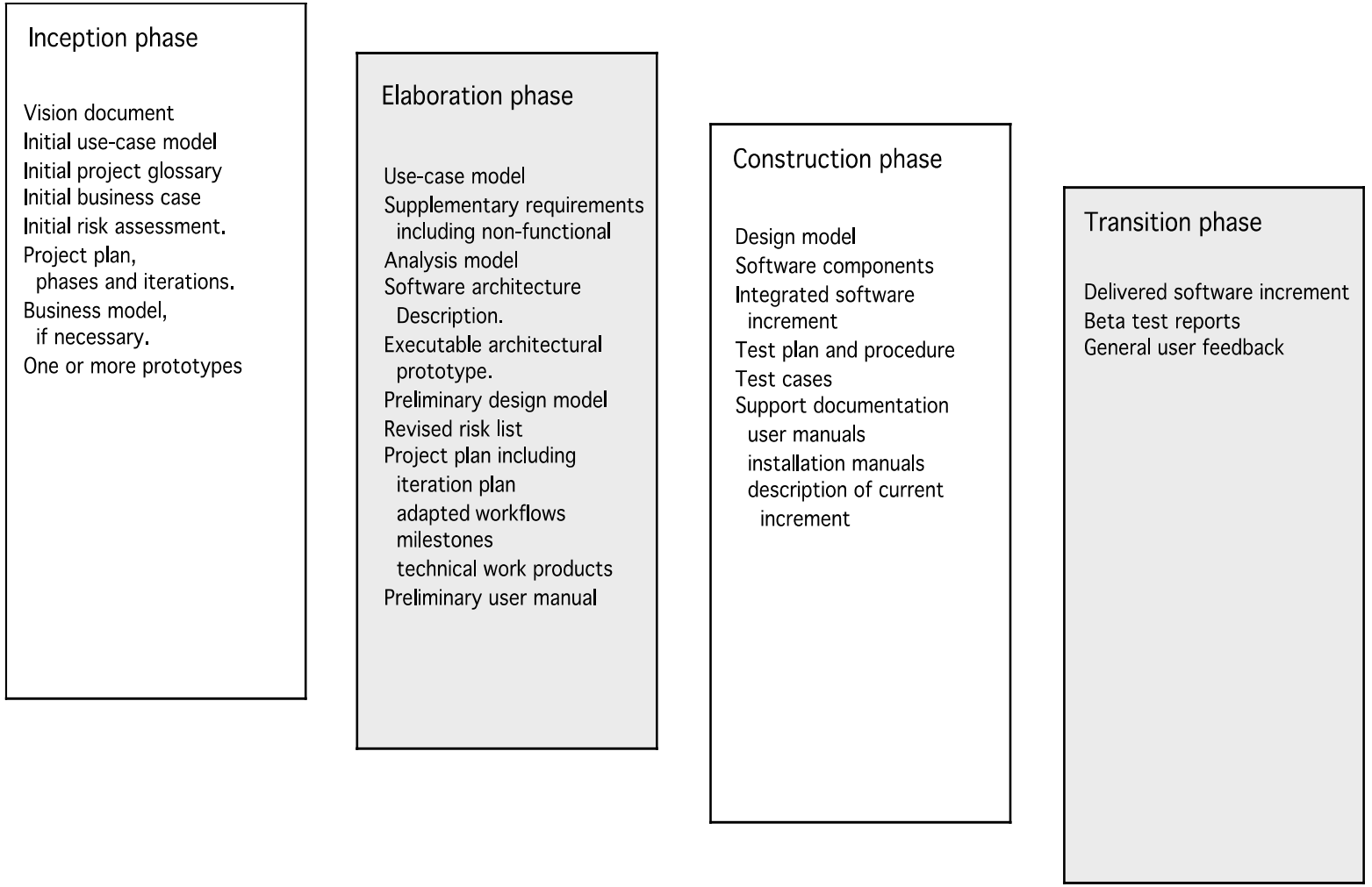
The Unified Process (UP)



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



Personal Software Process (PSP)

In some cases building your own personal or team's process is better to fit your needs:

- **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.** External specifications for each component to be constructed are developed 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 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 collected (this is a substantial amount of data that should be analyzed statistically), the effectiveness of the process is determined. 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.
- Provide improvement guidance to high-maturity organizations.
- Facilitate university teaching of industrial-grade team skills.