INTRODUCTION TO SOFTWARE ENGINEERING

MSc (CS)
SCMS, SPPU, Pune

WHAT IS SOFTWARE

Software's Dual Role

Software is a product

- Transforms information - produces, manages, acquires, modifies, displays, or transmits information
- Delivers computing potential of hardware and networks

Software is a vehicle for delivering a product

- Controls other programs (operating system)
- Effects communications (networking software)
- Helps build other software (software tools & environments)

WHAT IS SOFTWARE?

Computer programs and associated documentation such as requirements, design models and user manuals.

Software products may be developed for a particular customer or may be developed for a general market.

Software products may be

- Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
- Bespoke (custom) - developed for a single customer according to their specification.

New software can be created by developing new programs, configuring generic software systems or reusing existing software

CATEGORIES OF SOFTWARE

System Software
- Service other software – OS, Compilers, Databases etc

Application software
- Solve specific business need – Billing, ERP etc

Engineering / Scientific software
- Number crunching – CAD, Simulation, Weather etc

Embedded Software
- Resides within a product – Elevators, Home electronics etc.

Product Line Software
- Commonly used software – Word processor, Spreadsheets etc

Web applications
- Run on internet or intranet. Slowly merging into other categories of software

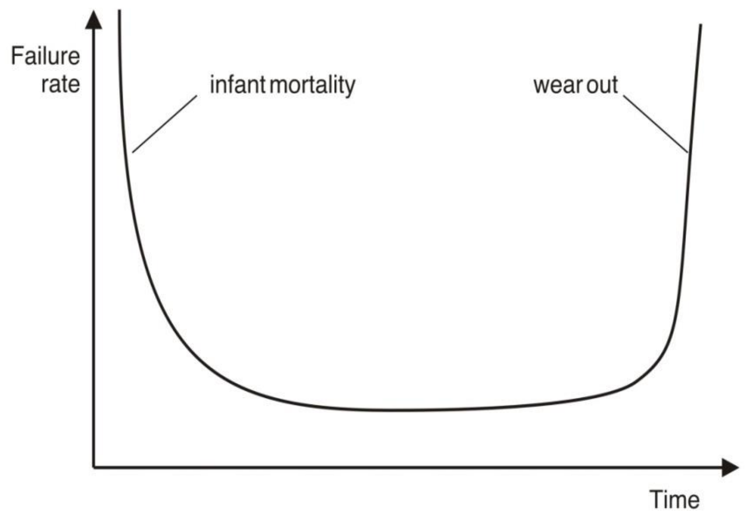
Artificial intelligence software
- Nonnumeric algorithms to solve complex problems – Robotics, Pattern recognition etc

HARDWARE VS. SOFTWARE

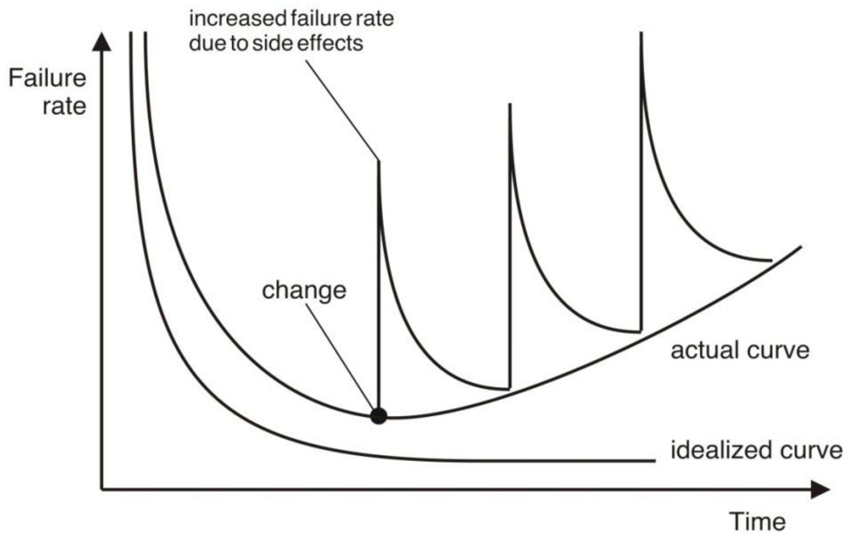
Hardware	Software
➢ Manufactured	➢ Developed/engineered
➢ Wears out	➢ Deteriorates
➢ Built using components	➢ Custom built
➢ Relatively simple	➢ Complex

WEAR VS. DETERIORATION

Hardware wears out over time



Software deteriorates over time



MANUFACTURING VS. DEVELOPMENT

Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.

In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.

Unlike hardware, software costs are concentrated in design rather than production.

ATTRIBUTES OF A GOOD SOFTWARE

The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable

Maintainability

- Software must evolve to meet changing needs

Dependability

- Software must be trustworthy

Efficiency

- Software should not make wasteful use of system resources

Acceptability

- Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems

CHANGE IS CONSTANT

Software change is inevitable

- New requirements emerge when the software is used
- The business environment changes
- Errors must be repaired
- New computers and equipment is added to the system
- The performance or reliability of the system may have to be improved

A key problem for organisations is implementing and managing change to their existing software systems

THE PITFALLS OF SW DEVELOPMENT….

In the early days software was ‘personalized’ and small applications.
Sophistication of applications grew, programs grew in size & complexity
Software projects were taking a lot longer than originally envisaged
Software was costing a lot more to develop than initially estimated
Software was being delivered to the customer only to fail
Maintenance became difficult
Absence of a structured and methodical approach caused:

- Cascading effect of errors made in one phase carried over to subsequent phases causing rework and therefore delay
- Difficulty in estimating effort, and measuring progress caused schedule slippage and budget overruns
- Quality of delivered software in terms of defects was not estimated, nobody knew how many defects to expect in the deliverables
- Version control , CM, Change Control – Non existent or poor

Demands of Today’s Business - Hardware sophistication, Technological advances in the software, Frequently changing business needs

NEED FOR ENGINEERING APPROACH

It is clear that a discipline for software development must exist --one that integrates comprehensive methods for all stages in software development, better tools for automating these methods, more powerful techniques for software quality assurance, and an overall philosophy for coordination, control and management. Such a discipline is ***Engineering.***

The Engineering Approach : Any branch of engineering uses

- Methods & Techniques
- Tools
- Standards

  for designing and building a product

Software development also consists of building something - good, reliable software.

SOFTWARE ENGINEERING - DEFINITIONS

Many have developed personal definitions of *software engineering,* a definition proposed by Fritz Bauer serves as a basis for discussion:

> [SE is] the establishment and use of sound engineering principles (principles or practices reasonably accepted) in order to obtain economically software that is reliable and works efficiently on real machines.

Gaps :

It says little about the technical aspects of software quality; it does not directly address the need for customer satisfaction or timely product delivery; it omits mention of the importance of measurement and metrics; it does not state the importance of a mature process. Bauer's definition provides us with a baseline.

Questions that continue to challenge software engineers.

What "<u>sound engineering principles</u>" can be applied to computer software development? How do we "economically" build software so that it is "reliable"? What is required to create computer programs that work "<u>efficiently</u>" on not one but many different "real machines"?

SOFTWARE ENGINEERING (SE)

The basic principle of SE is to use structured, formal, disciplined methods for building and using systems, just as in any other branch of engineering. Software Engineering provides certain paradigms that encompass methods, tools and procedures.

- **Methods** *provide the rules and steps for carrying out SE tasks, such* as project planning and estimation, system and s/w requirements analysis, design of data structure, program architecture and algorithm procedure, coding, testing and maintenance.
- ***Tools*** *provide automated or semi-automated support for methods. They can be* integrated into a system which automates a range of SE methods, called CASE (Computer-Aided SE). CASE tools are available to support & automate most of the methods mentioned.
- ***Procedures*** *bind the methods and tools into a framework, defining the sequence in* which methods will be applied, the deliverables (documents, forms, reports, etc.) that are required, the controls that ensure quality and coordinate change, and the milestones that help a manager assess progress.

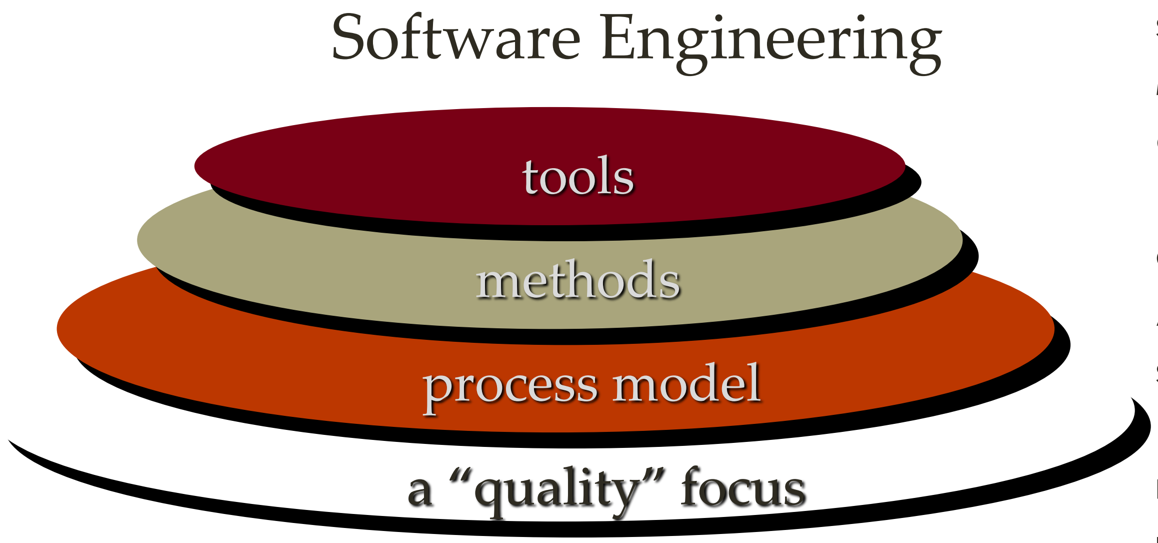
These paradigms may be viewed as models of s/w development.

THE PROCESS

The process is a dialogue in which the knowledge that must become the software is brought together and embodied in the software. The process provides interaction between users and designers, between users and evolving tools, and between designers and evolving tools [technology].

Indeed, building computer software is an iterative learning process, and the outcome, something that Howard Baetjer would call “software capital,” is an embodiment of knowledge collected, distilled, and organized as the process is conducted.

A LAYERED TECHNOLOGY



Tools: This layer contains automated or semi-automated tools that offer support for the framework and the method each software engineering project will follow.
Method: This layer contains the methods, the technical knowledge and "how-tos" in order to develop software.
Process: This layer consists of the framework that must be established for the effective delivery of software.
A Quality Focus: This layer is the fundamental layer for software engineering. It is of great importance to test the end product to see if it meets its specifications. Efficiency, usability, maintenance and reusability are some of the requirements that need to be met by new software.

A GENERIC VIEW OF SOFTWARE ENGINEERING

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities. Regardless of the entity to be engineered, the following questions must be asked and answered:

What is the problem to be solved?

What characteristics of the entity are used to solve the problem?

How will the entity (and the solution) be realized?

How will the entity be constructed?

What approach will be used to uncover errors that were made in the design and construction of the entity?

How will the entity be supported over the long term, when corrections, adaptations, and enhancements are requested by users of the entity?

A GENERIC VIEW OF SOFTWARE ENGINEERING

Three generic phases, regardless of application area, project size, or complexity. Each phase addresses one or more of the questions

- **Definition phase** focuses on what - identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified.

- **Development phase** focuses on how - define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed.

- **Support phase** focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software.

SOFTWARE PROCESS

- **Process**: A particular method, generally involving a number of steps

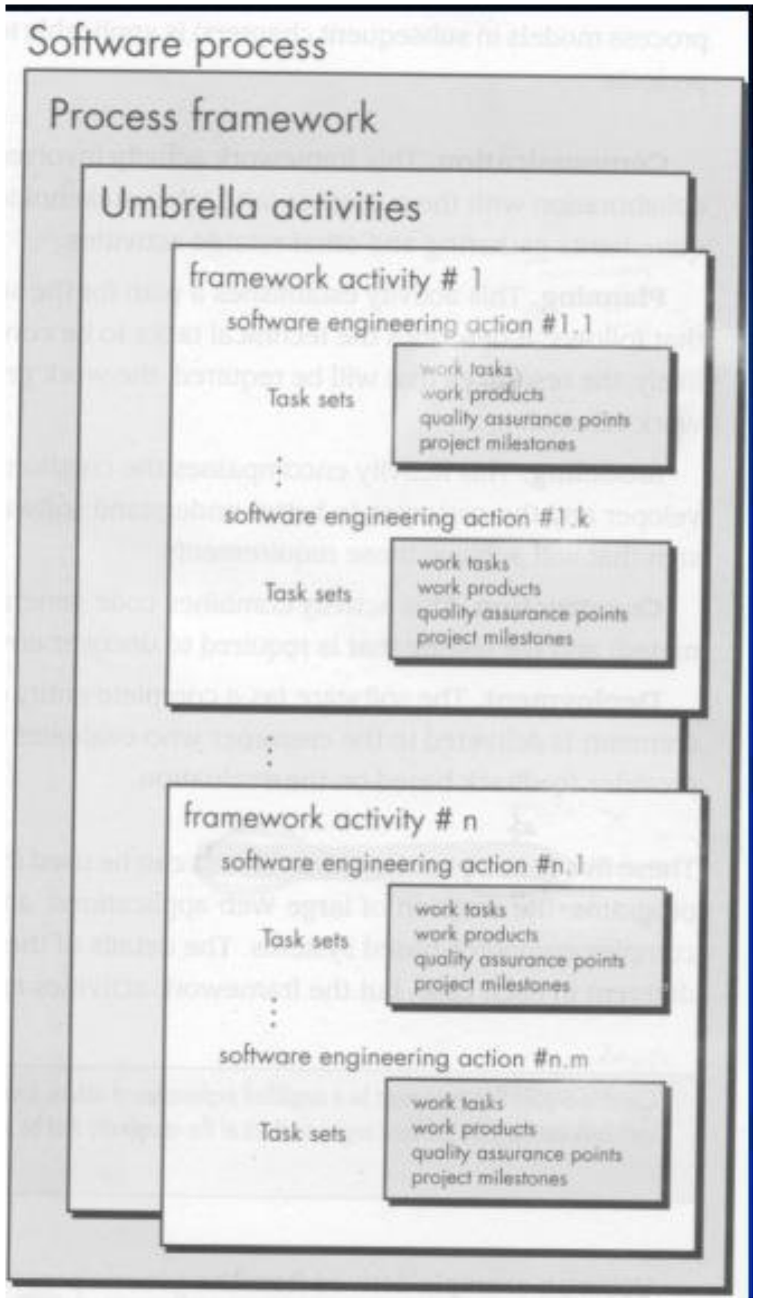
- **Software Process**: A set of steps, along with ordering constraints on execution, to produce software with desired outcome

Many types of activities performed by different people in a software project

Process is distinct from product – products are outcomes of executing a process on a project

A PROCESS FRAMEWORK

- Establishes the foundation for a complete software process
- Identifies a number of framework activities applicable to all software projects
- Also include a set of umbrella activities that are applicable across the entire software process.



A PROCESS FRAMEWORK

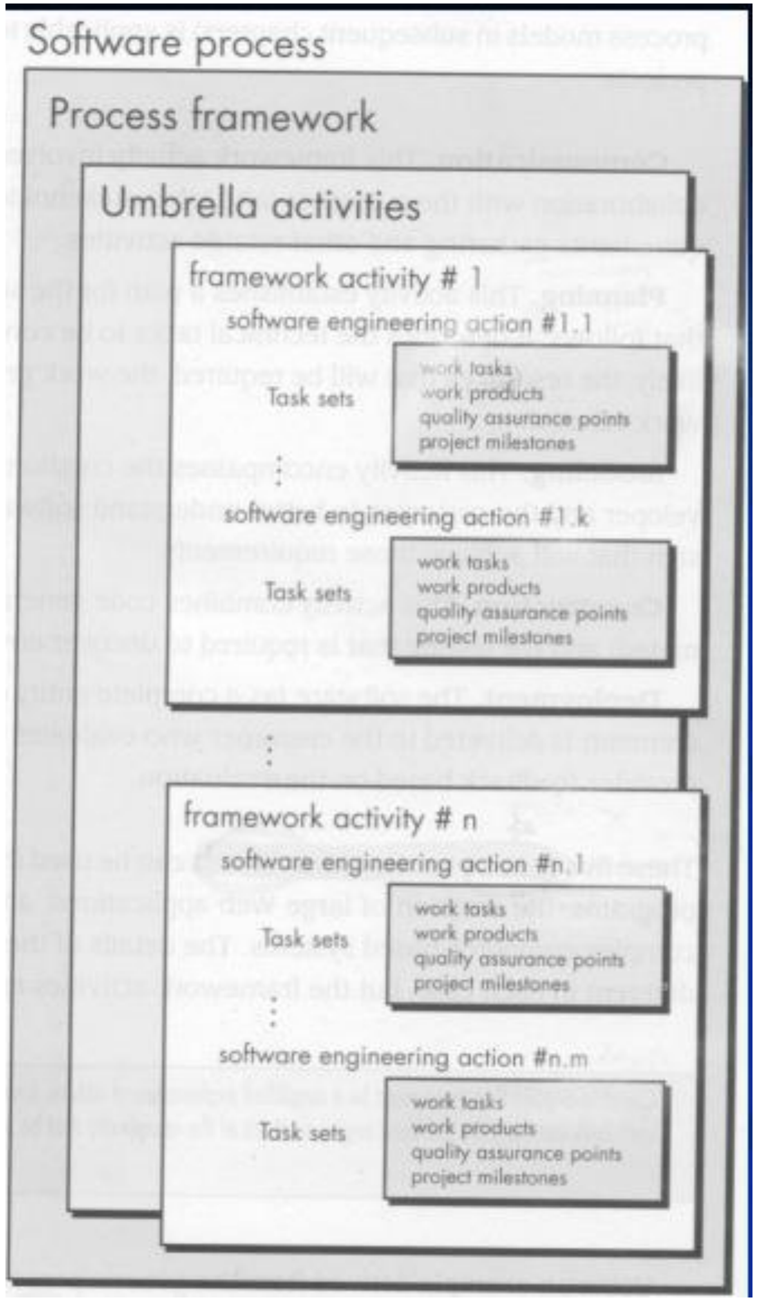
A Process Framework

- Establishes the foundation for a complete software process
- Identifies a small number of framework activities that applies to all s/w projects, regardless of size/complexity

Umbrella activities applicable across entire s/w process.

Each framework activity has set of s/w engineering actions

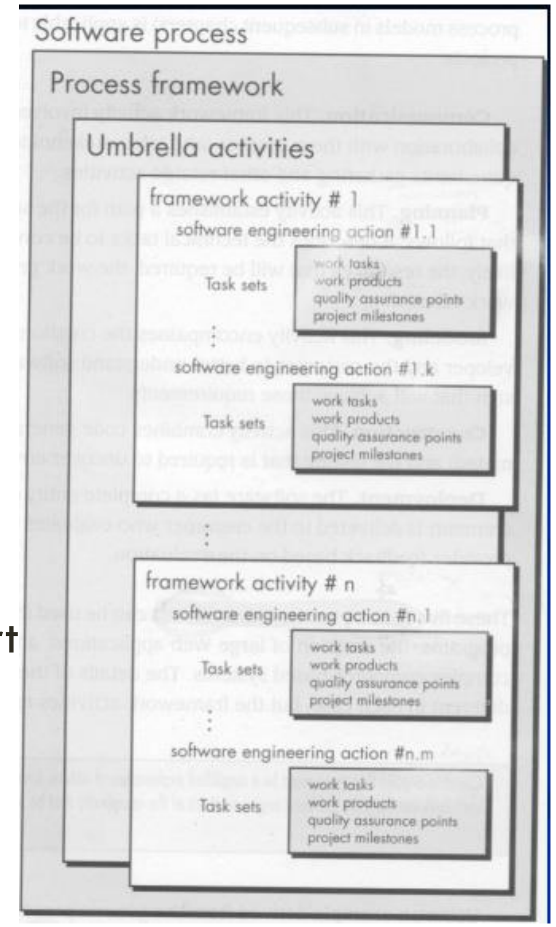
Each s/w engineering action (e.g., design) has collection of related tasks



FRAMEWORK ACTIVITIES

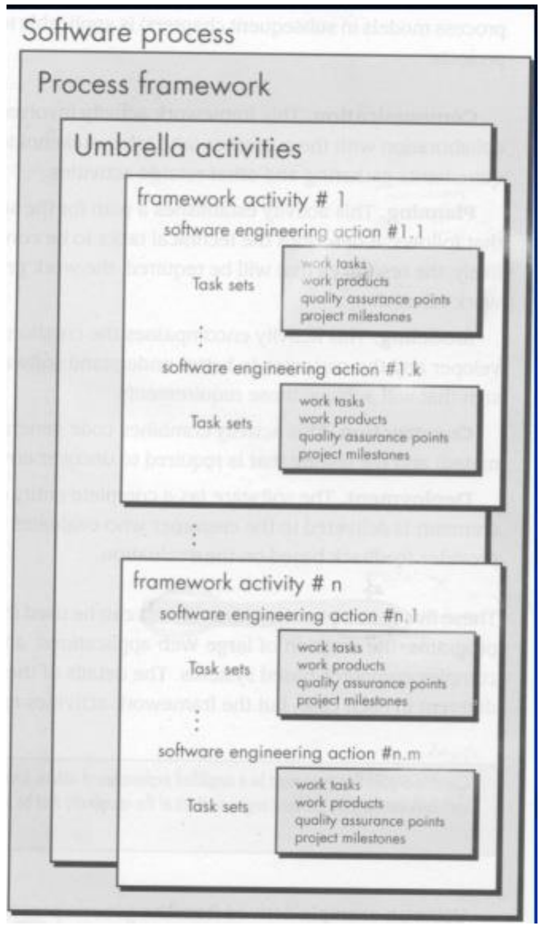
1. **Communication:** The software development starts with the communication between customer and developer.
2. **Planning:** It consists of complete estimation, scheduling for project development and tracking.
3. **Modeling:** Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.

   The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.
4. **Construction:** Construction consists of code generation and the testing part. Coding part implements the design details using an appropriate programming language. Testing is to check whether the flow of coding is correct or not. Testing also check that the program provides desired output.
5. **Deployment:** Deployment step consists of delivering the product to the customer and take feedback from them. If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.



UMBRELLA ACTIVITIES

- Software Project Tracking and Control (Assess progress and take action to maintain commitment)
- Risk Management (Assess risks that will affect outcome and quality)
- Software Quality Assurance (Define and conduct activities to ensure software quality)
- Formal Technical Reviews (Assess Work products to uncover and rectify defects)
- Measurement (Define and collect process, project and product metrics)
- Software Configuration Management (Manage effect of change)
- Reusability Management (Define criteria and establish mechanisms for reuse)
- Work Product Preparation and Production (Activities required to create the work products)



THE PROCESS MODEL: ADAPTABILITY

Framework activities will always be applied on every project ...

BUT

Tasks (and degree of rigor) for each activity will vary based on:

- the type of project
- characteristics of the project
- common sense judgment; concurrence of the project team

SOFTWARE PROJECT

- Project – to build a s/w system within cost and schedule and with high quality which satisfies the customer
- Project goals – high Q and high P
- Suitable process needed to reach goals
- For a project, the process to be followed is specified during planning

AD-HOC DEVELOPMENT

No formal process - Sounds great! No learning required.

Drawbacks?

- Some important actions (design, testing) may go ignored
- Not clear when to start or stop doing each task
- Does not scale well to multiple people
- Not easy to review or evaluate one's work
- Code didn't match user's needs (no requirements!)
- Code was not planned for modification, not flexible

Key observation: The later a problem is found, the more expensive it is to fix.

DEVELOPMENT PROCESS

A set of phases and each phase being a sequence of steps

Sequence of steps for a phase - methodologies for that phase.

Why have phases
- To employ divide and conquer
- Each phase handles a different part of the problem
- Helps in continuous validation

Common activities: Requirements analysis, architecture, design, coding, testing, delivery

Different models perform them in different manner

LIFE CYCLE PHASES

Software lifecycle: series of steps / phases

- Requirements Analysis & Specification
- High-level (Architectural) Design
- Detailed (Object-oriented) Design
- Implementation, Integration, Debugging
- Testing, Profiling, Quality Assurance
- Operation and Maintenance
- Other possibilities: Risk Assessment, Prototyping

In each phase

- Mark out a clear set of steps to perform
- Produce a tangible document or item
- Allow for review of work
- Specify actions to perform in the next phase

REQUIREMENT ANALYSIS, DESIGN

Requirement Analysis

To understand and state the problem precisely

Forms the basis of agreement between user and developer

Specifies “what” , not “how”

Not an easy task, Requirement specifications of a medium systems can run to hundreds of pages

Output is the software Requirement Specification (SRS)

Design

A major step in moving from problem domain to solution domain

- Architecture design – components & connectors needed in the system
- HLD – modules and data structures to implement the architecture
- Detailed design – logic of modules

Most methodologies focus on architecture or high level design -Outputs are architecture/design/logic design document

CODING, TESTING

Coding

Goal: Implement the design with simple and easy to understand code.

Converts design into code in specific language, Output is Code

Affects both testing and maintenance

Testing

Goal: Identify MOST of defects

Defects get introduced in each phase, need to be detected and removed to achieve high quality

Testing plays this important role

A very expensive task; has to be properly planned and executed

Outputs are Test plans/results & final tested (hopefully reliable) code

Distribution of effort

❖ Requirement Analysis	-	10-20%
❖ Design	-	10-20%
❖ Coding	-	20-30%
❖ Testing	-	30-50%

PRESCRIPTIVE PROCESS MODELS

Prescriptive process models advocate an orderly approach to software engineering

Prescribed set of Process elements, Framework activities, Tasks, Work Products, QA, Change control

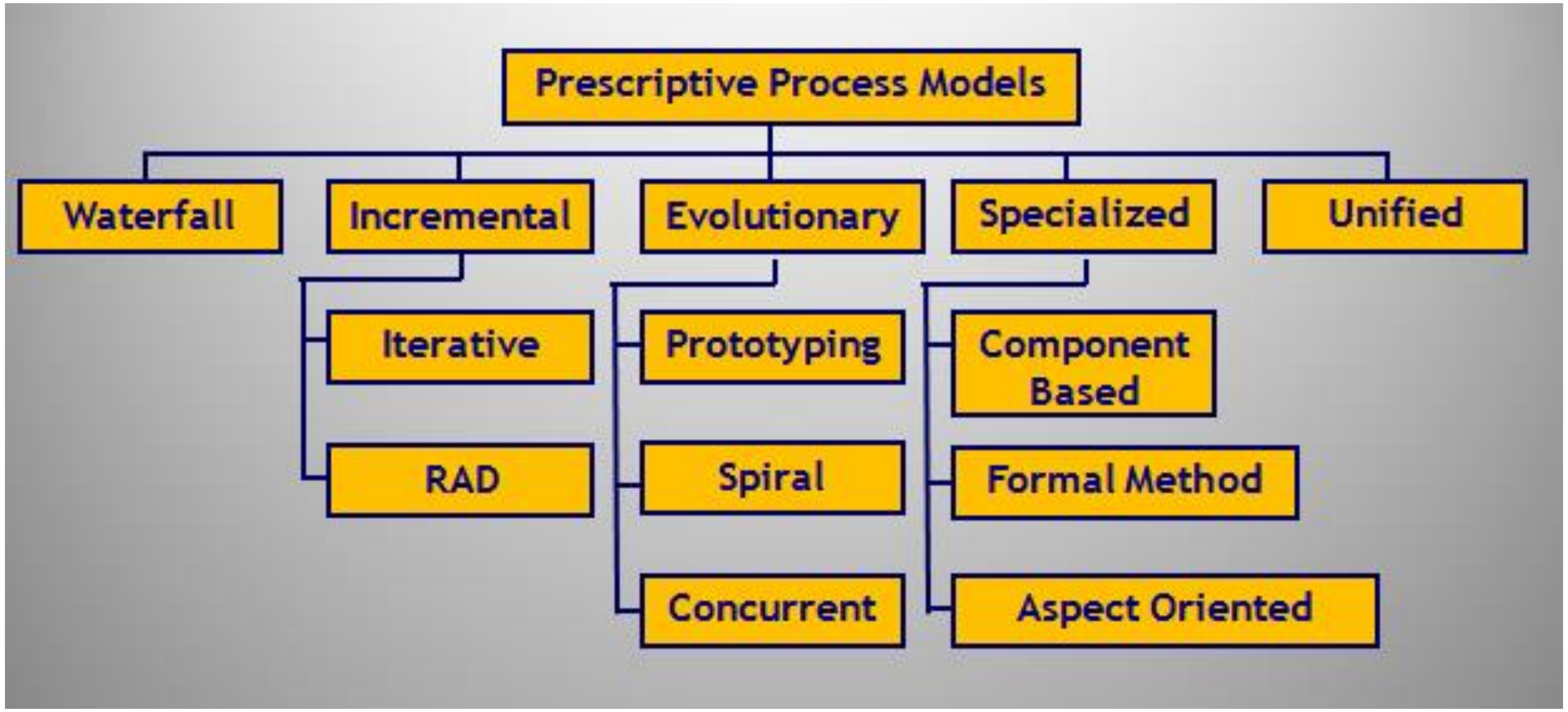
Prescribes work flow

Accommodate Specific nature of work, People who will work and Environment in which the work will happen

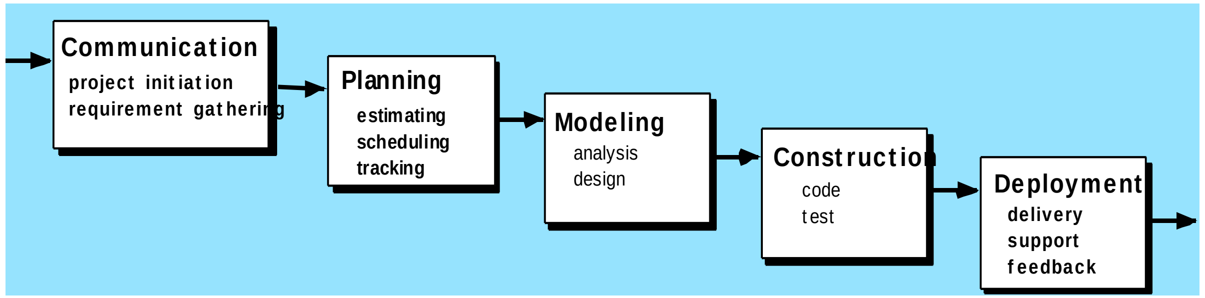
Provide a road-map so that progress can be monitored

Generic Process Framework – Communication, Planning, Modeling, Construction, Deployment

PRESCRIPTIVE PROCESS MODELS



THE WATERFALL MODEL



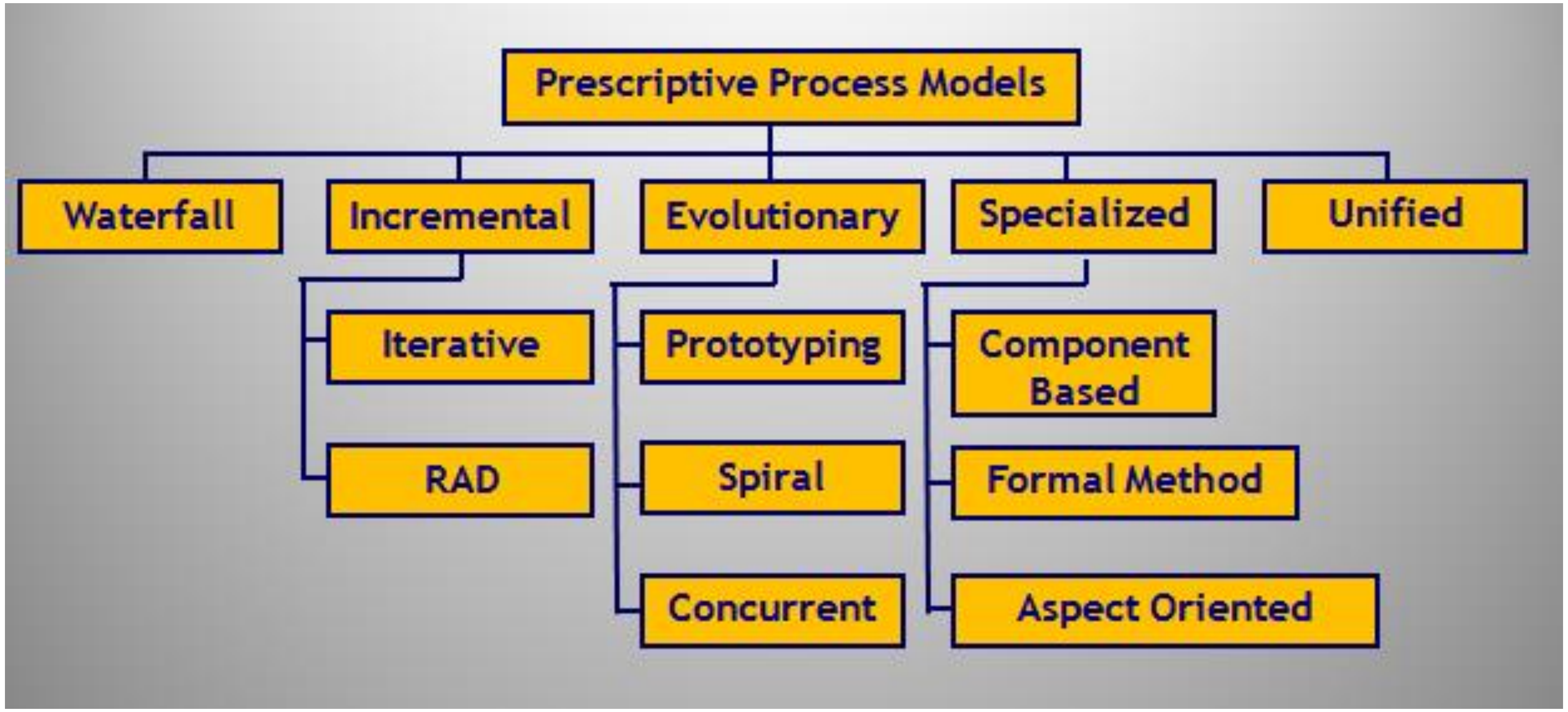
Classic Lifecycle Model
Systematic Sequential Approach to Software Development
Oldest Paradigm for Software Development
Advantages

- Formal, standard; specific phases with clear goals
- Easy to administer in a contractual setup – each phase is a milestone
- Useful where requirements are clear at start of project
- Good feedback loops between adjacent phases

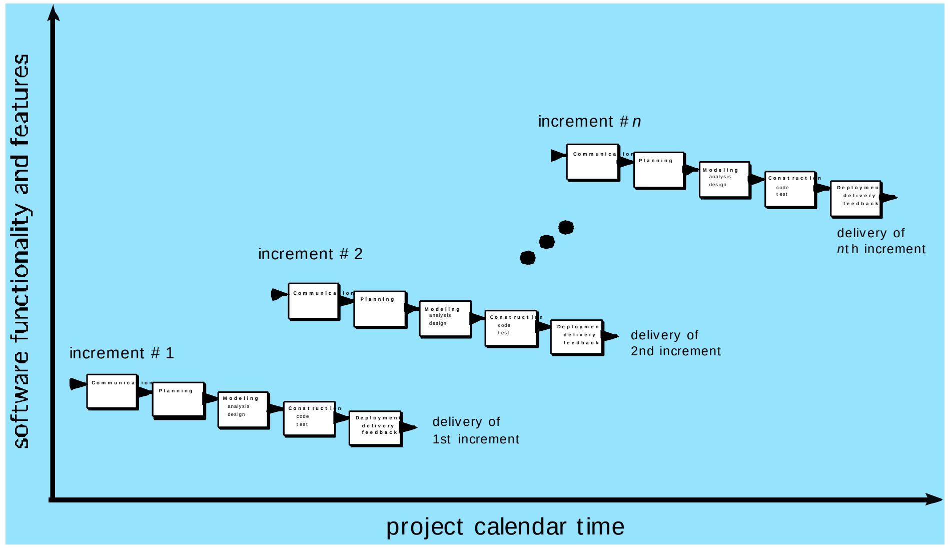
DRAWBACKS OF WATERFALL MODEL

- Requirements are not clear at beginning of project in most situations
- Requires a lot of planning up front (not always easy)
- Nothing to show until almost done ("we're 90% done I swear!“) - working version of software not available till late, so no early feedback can leads to ‘Blocking State’
- Few business systems have stable requirements
- Change Management –difficulty in accommodating change after the process is underway.
- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Costly to "swim upstream" back to a previous phase
- Model is appropriate when requirements are well-understood and changes will be fairly limited during the design process.
- Mostly used for large systems engineering projects where a system is developed at several sites.

PRESCRIPTIVE PROCESS MODELS

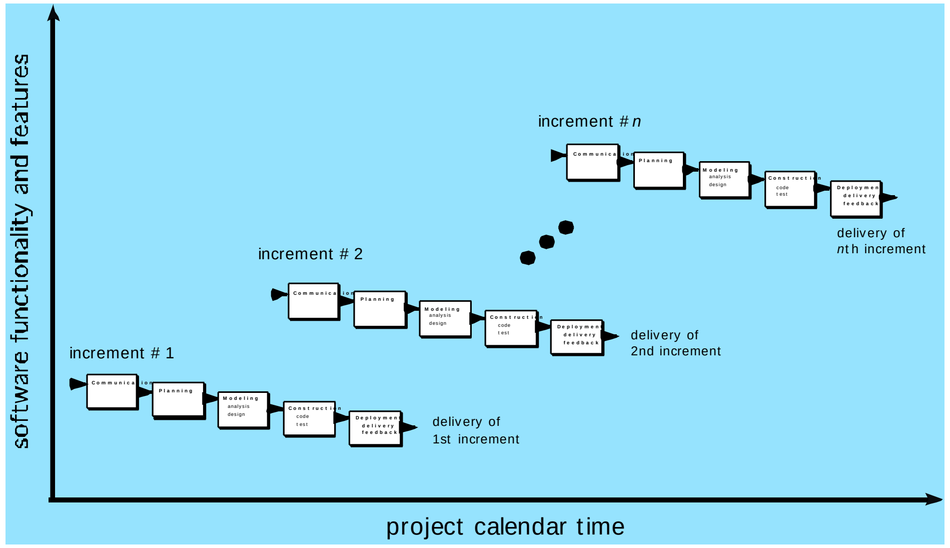


THE INCREMENTAL MODEL



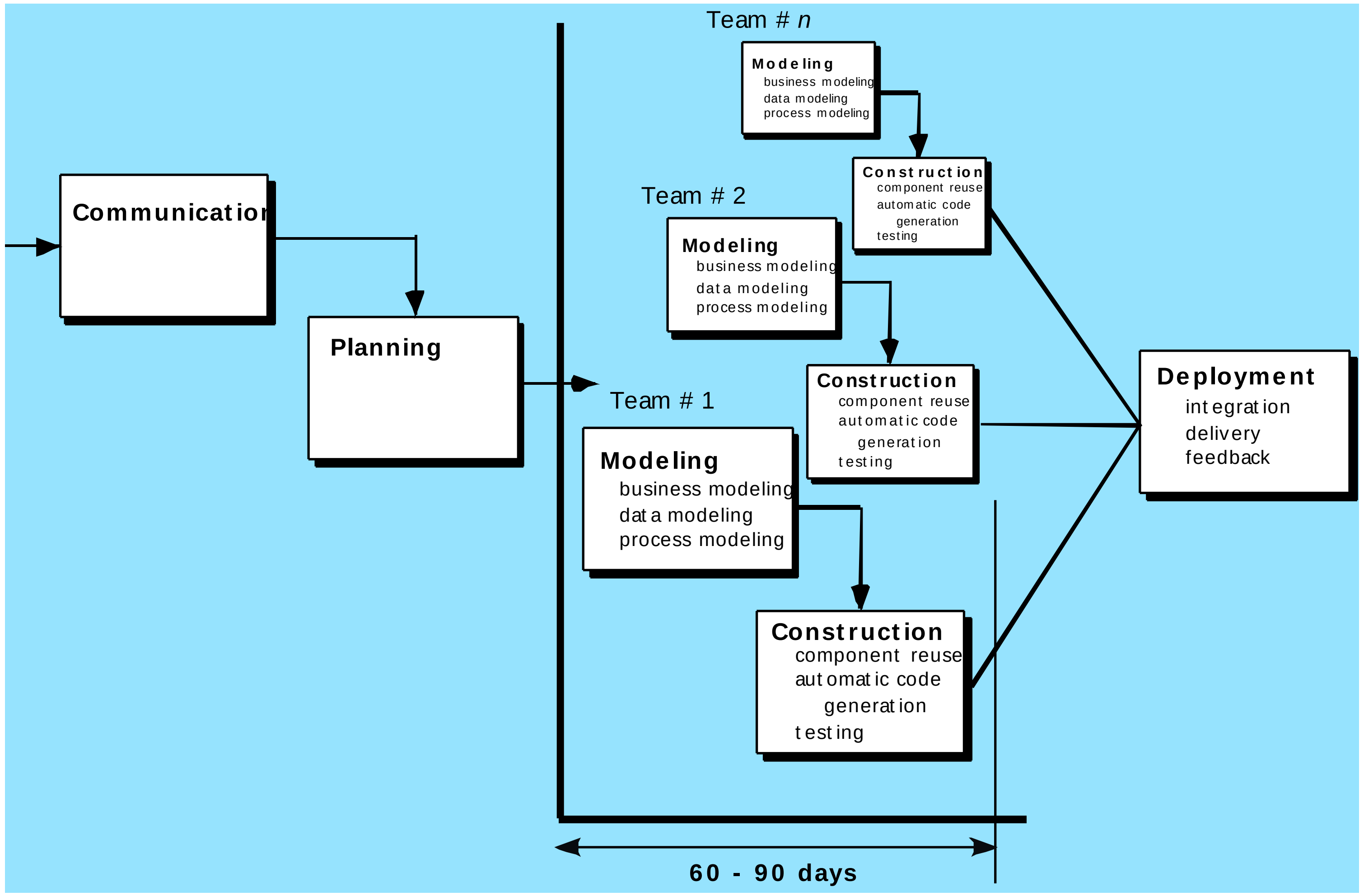
- There are many situations in which initial software requirements are reasonably well-defined, but the overall scope of the development efforts precludes a limited set of software functionality to user quickly and then refine and expand on the functionality in later software releases.
- In such cases, a process model that is designed to produce the software in increments is chosen.

THE INCREMENTAL MODEL



- Incremental Model combines elements of Waterfall model applied in an iterative fashion
- Software produced in increments
- First increment is the core product – Basic requirements
- Based on customer evaluation & feedback, additional delivery of features and functionality undertaken
- The process is repeated until the complete product is produced
- Unlike prototyping this focuses on delivery of an operational product at each increment
- This model is particularly appropriate when staffing is unavailable
- Increments can be planned to manage risks – schedule, Technical and Market

THE RAD (RAPID APPLICATION DEVELOPMENT) MODEL



THE RAD (RAPID APPLICATION DEVELOPMENT) MODEL

- Incremental software process model that emphasizes a short development cycle
- The RAD model is high-speed adaptation of the waterfall model
- If requirements are well understood and project scope is constrained, the RAD process enables a development team create a fully functional system within a very short period of time.
- Rapid Application Development (RAD) is a linear sequential model
- Incremental model emphasizes short development cycle – 60 to 90 days
- Component Based Construction approach – Reuse, Automatic code generation
- Multiple Software Teams work in parallel
- Used primarily in information systems applications

PHASES OF THE RAD MODEL

Business Modeling

- Information flow among business functions modeled
- What information drives the business process?
- What information is generated?
- Who generates it?
- Where does the information go?
- Who processes it?

Data Modeling

- Information flow refined into a set of data objects needed to support the business
- Attributes of the objects are identified
- Relationships between the objects are defined

PHASES OF THE RAD MODEL

Process Modeling

- Data objects are transformed to achieve the information flow necessary to implement a business function
- Processing descriptions are created for adding, modifying, deleting and retrieving a data object

Application Generation

- RAD assumes use of fourth generation techniques (4GT’s)
- RAD process works to make use of existing program components
- RAD process encourages creation of reusable components
- automated tools used to facilitate construction of software

Testing and Turnover

- Component reuse reduces overall testing time
- New components must be tested

THE RAD MODEL

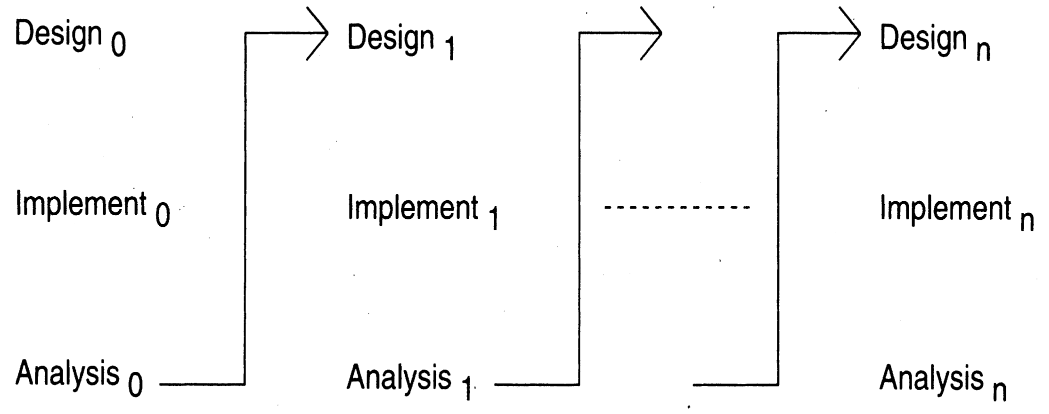
RAD approach maps into the generic framework activities:

- Communication works to understand the business problems and requirements
- Planning is essential because multiple software teams work in parallel
- Modelling encompasses three major phases – Business modelling, data modelling and process modelling – and establishes design representations the serve as basis of RAD's construction activity.
- Construction emphasizes use of pre-existing software components and application of automatic code generation
- Finally deployment establishes basis for subsequent iterations.

DRAWBACKS OF RAD

- For large scaleable projects, RAD requires sufficient human resources to create the right number of RAD teams
- Also each major function can be addressed by a separate team
- RAD requires developers and customers committed to rapid-fire activities necessary to complete tasks in small time frame
- If the system cannot be modularized, building components for RAD will be problematic
- If high performance is the issue RAD may not be appropriate since tuning of interfaces with system components required
- It is not appropriate when technical risks are high
- Change Management may not be easy

ITERATIVE DEVELOPMENT



Counters the "all or nothing" drawback of the waterfall model

Combines benefit of prototyping and waterfall

Focuses on delivering operational units with each increment

First increment is often a core product

Each increment is complete in itself

Can be viewed as a sequence of waterfalls

Feedback from one iteration is used in the future iterations

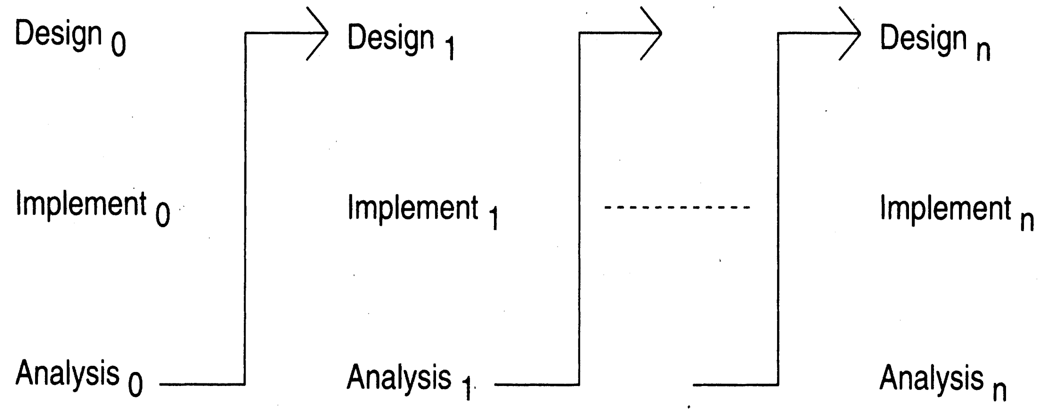
Applicability: where response time is important, risk of long projects cannot be taken, all requirements not known

Newer approaches like XP, Agile,… all rely on iterative development

Benefits: Get-as-you-pay, feedback for improvement

Drawbacks: Architecture/design may not be optimal, rework may increase, total cost may be more

ITERATIVE MODEL



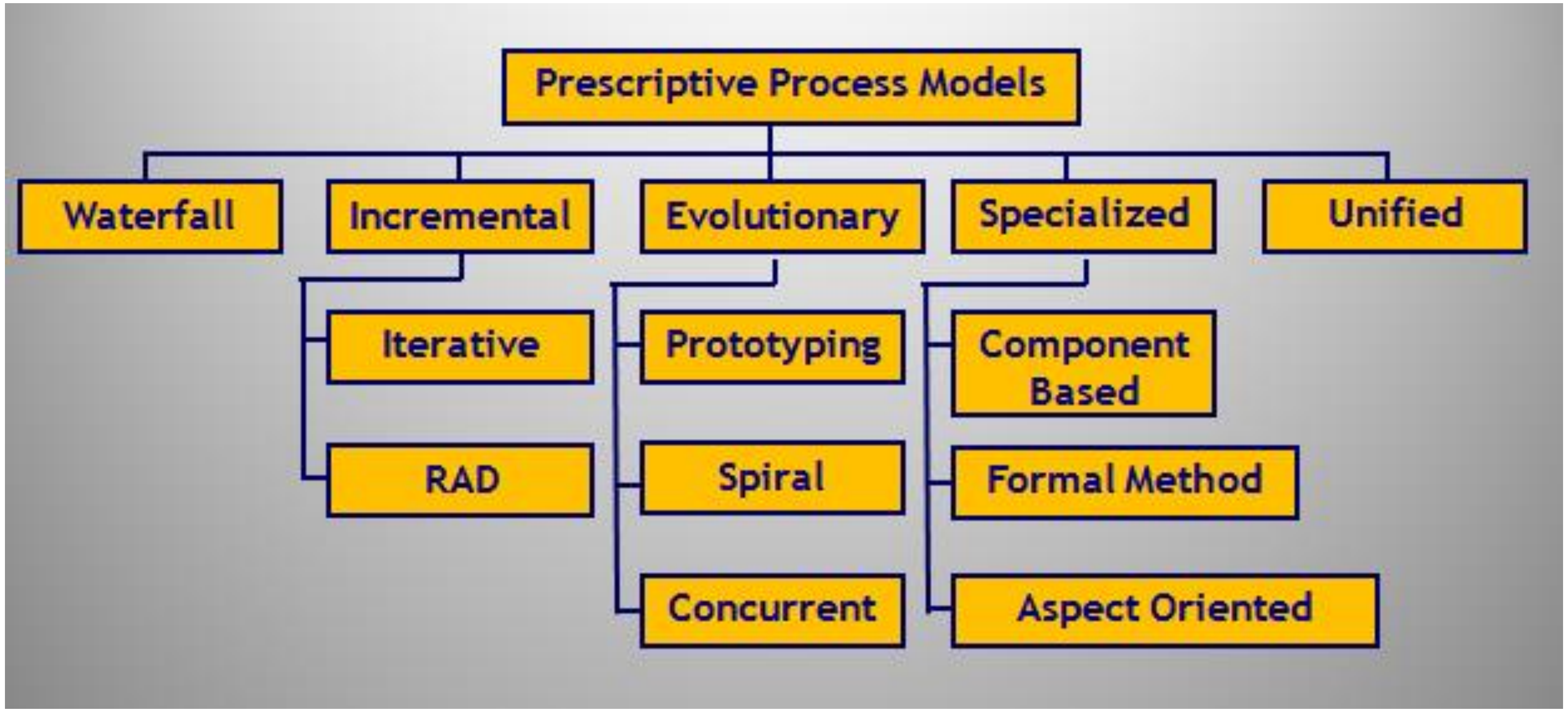
The Iterative model, is useful

- When staffing is unavailable for a complete implementation.
- When the project is of a very large nature
- When prioritisation is required in terms of functionality delivery
- Acceptability of new system is in doubt at initial stages

The drawbacks of Iterative model

- Integration may become an issue at a later date
- Multiple cycles and repeat of testing required
- Some of rework will be required if increments are not planned properly

PRESCRIPTIVE PROCESS MODELS



EVOLUTIONARY PROCESS MODELS

Evolutionary Process models produce an increasingly more complete version of the software with each iteration

- Business and Product requirements often change as development processed, making a straight-line path to an end product unrealistic
- Tight market deadlines make completion of a comprehensive software product impossible, but a limited version must be introduced to meet competitive or business pressure.
- A set of core product or system requirements is well understood, but details of the product or system extensions are yet to be defined.

Accommodates business and product requirement changes during the development cycle

Iterative process

Complexity of software increases as it evolves

EVOLUTIONARY MODEL : PROTOTYPING

Prototyping begins with communication. The s/w team and customer meet to define the overall objectives of the application and identify whatever requirements that are known

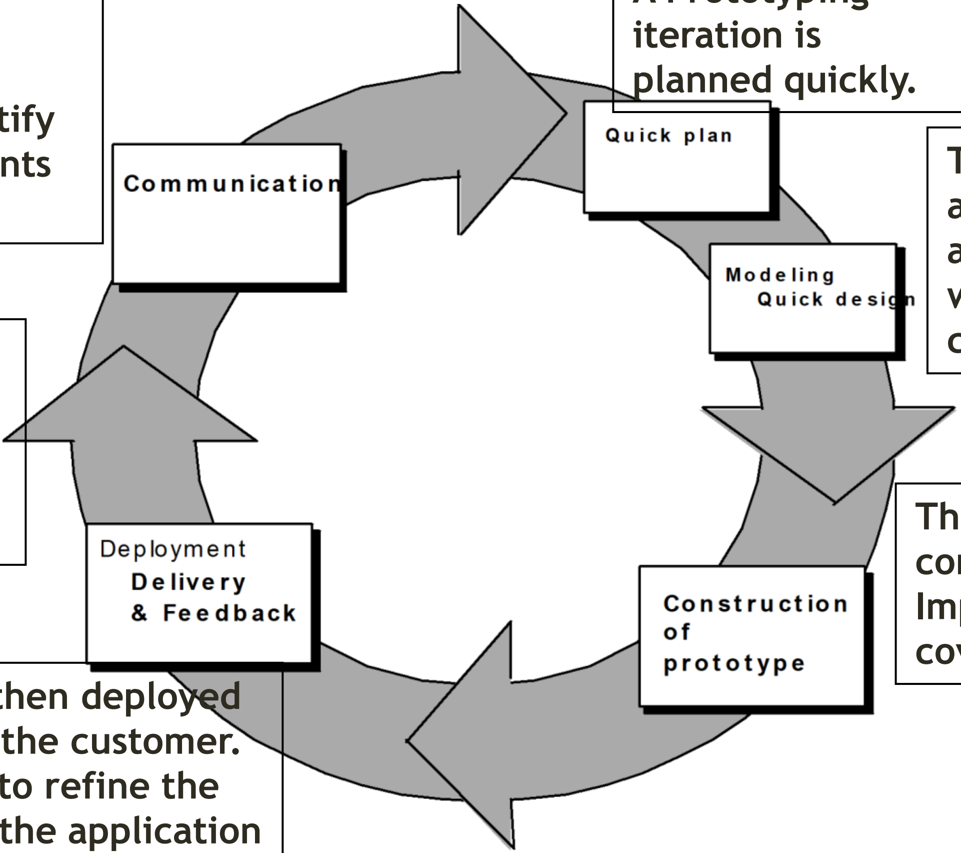
A Prototyping iteration is planned quickly.

The quick design focuses on a representation of those aspects of the software that will be visible to the customer

The quick design leads to construction of a prototype. Important business rules are covered

The prototype is then deployed and evaluated by the customer. Feedback is used to refine the requirements for the application

Iteration occurs as the prototype is tuned to satisfy the needs of the customer



PROTOTYPING

Prototyping may offer the best approach when

- Your customer has legitimate need but is clueless about the details
- Or when a developer is unsure about the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interactions should take
- Or where a proof of concept is required before decision making

Although Prototyping can be used as a standalone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models.

THE PROTOTYPING MODEL

Helps developer and customer to better understand functionality when requirements are fuzzy

Overall objectives defined, identify known requirements and outline areas where more detailing is required

Quick design focusing on end-user visible details (e.g. UI)

After prototype is built, the developer attempts to make use of existing program fragments or applies tools that enable working programs to be generated rapidly

Prototype constructed, deployed and evaluated

Further iterations to add functionality

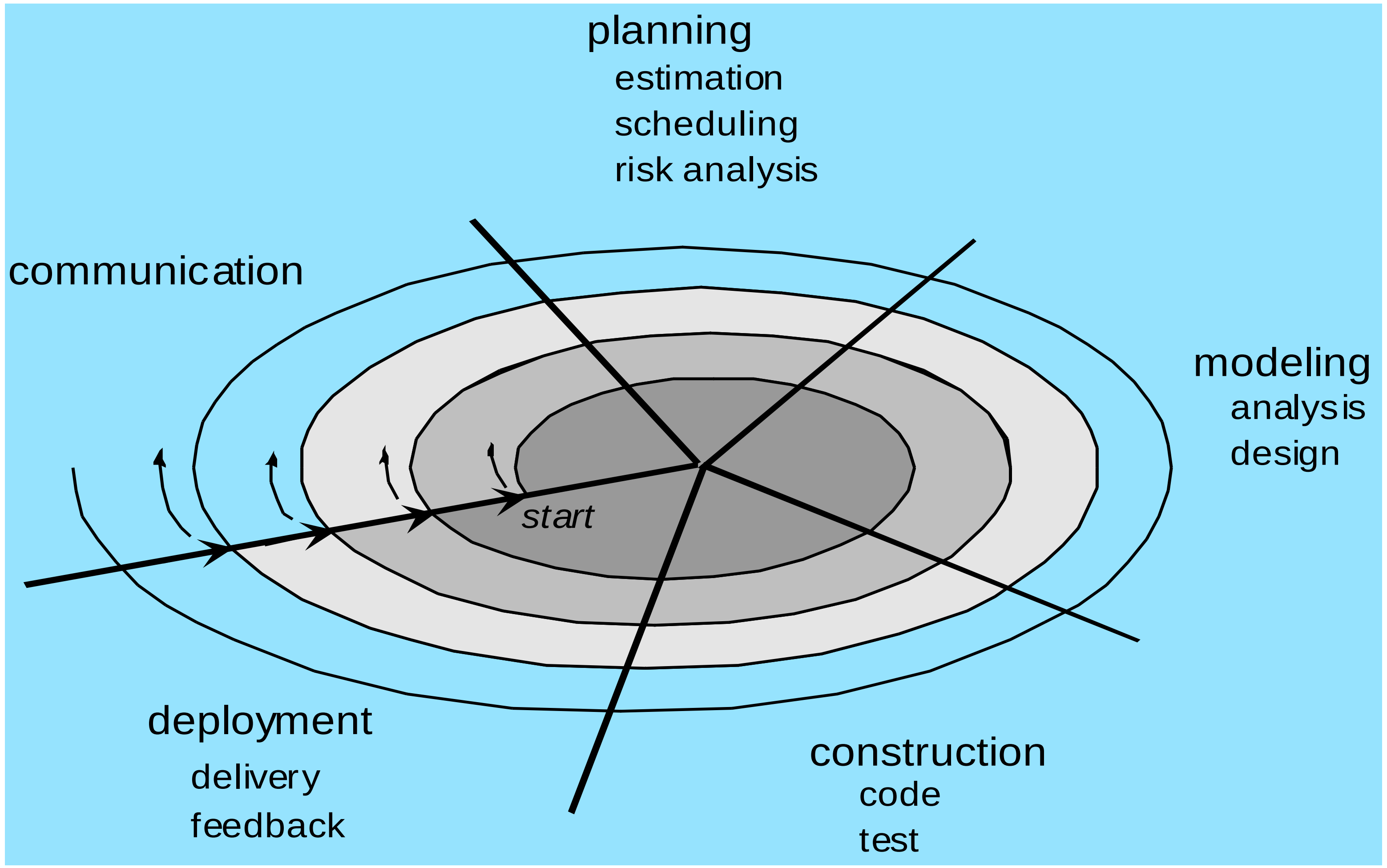
Working v/s Throwaway Prototype

The Prototype can become the product even if not engineered for quality

PROTOTYPING - DRAWBACKS

- The Prototype becomes the product even if not engineered for quality
- Developers make compromises (inappropriate language, technology, environment) during prototype creation which gets carried forward into the final product – one forgets the reason for the selection
- 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, smarting but smarter. The management question, is not whether to build a pilot system and throw it away but whether to plan in advance to build a throwaway.
- Additional costs in case of throwaway prototype
- Customer assumes he is seeing a working version of the software, unaware that overall software quality or long-term maintainability is not factored in the product. Customer cries foul and demands that "a few fixes" be applied to make the prototype a working product. Too often, software development management relents and ends with poor quality products

EVOLUTIONARY MODELS: THE SPIRAL



SPIRAL MODEL

Originally proposed by Barry Boehm in "A Spiral Model for Software Development & Enhancement", Computer, Vol.21, no.5, May 1988

A spiral model is divided into a set of framework activities or task regions. As this evolutionary process begins, the software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center

- First circuit around the spiral might result in the development of a product specification
- Subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- Each pass thru the planning region results in adjustments to the project plan
- Cost and schedule are adjusted based on feedback derived from the customer after delivery.
- Project Manager adjust the planned number of iterations required to complete the software

THE SPIRAL MODEL

The Spiral Model is an evolutionary software process model that couples the iterative nature of the prototyping with the controlled and systematic aspects of the waterfall model.

Provides potential for rapid development of incremental versions

Toll gates at milestones to assess work products, risks, re-plan

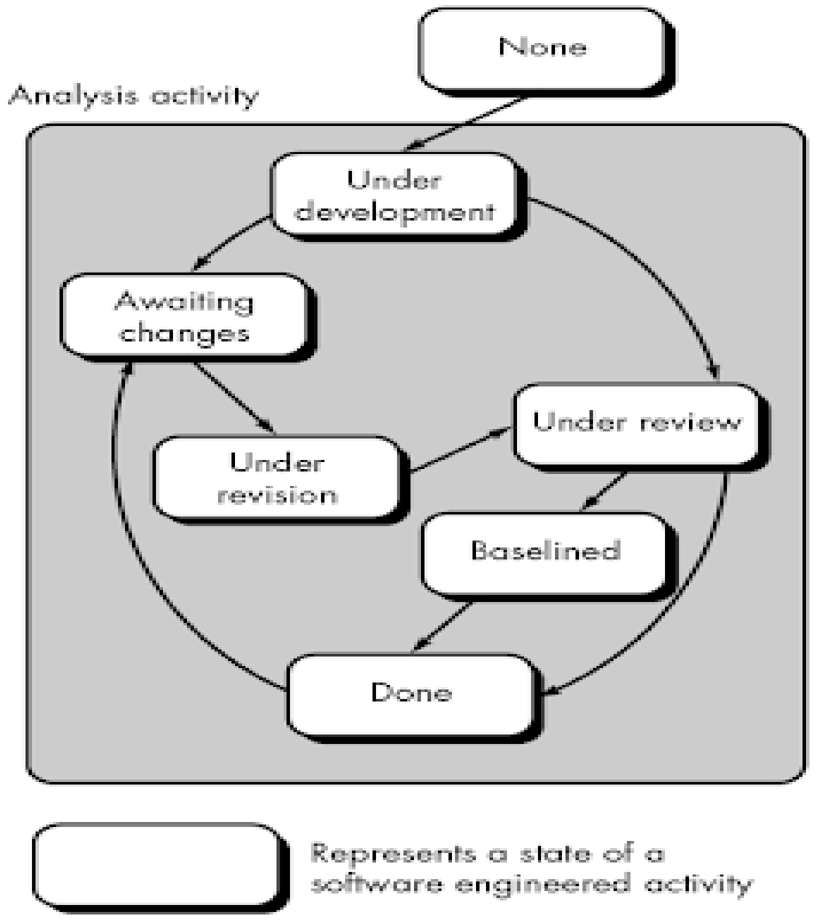
Can be used throughout the lifecycle of the product

More realistic approach for development of large-scale systems

Spiral model has two distinguishing features

- One is a cyclic approach for incrementally growing a system’s degree of definition and implementation while decreasing the its degree of risk.
- The other is a set of anchor point milestones for ensuring stakeholder commitment to feasible and mutually satisfactory system solutions

CONCURRENT DEVELOPMENT MODEL



The concurrent process model can be represented schematically as a series of major technical activities, tasks, and their associated states.

The activity—analysis—may be in any one of the states noted at any given time.

Similarly, other activities (e.g., design or customer communication) can be represented in an analogous manner.

All activities exist concurrently but reside in different states.

For example, early in a project the customer communication activity (not shown in the figure) has completed its first iteration and exists in the awaiting changes state. The analysis activity (which existed in the none state while initial customer communication was completed) now makes a transition into the under development state. If, however, the customer indicates that changes in requirements must be made, the analysis activity moves from the under development state into the awaiting changes state.

CONCURRENT DEVELOPMENT MODEL

The concurrent process model defines a series of events that will trigger transitions from state to state for each of the software engineering activities.

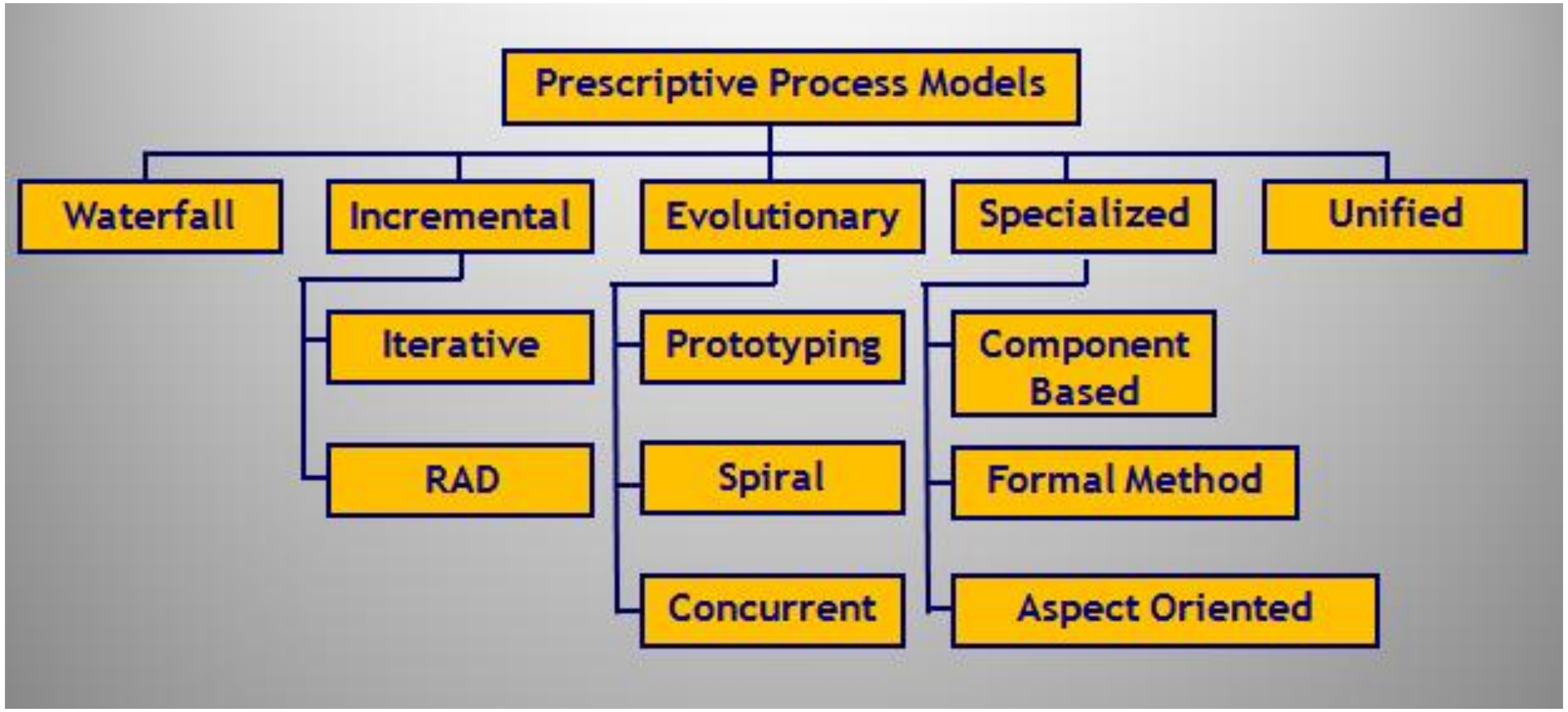
Example - during early stages of design, an inconsistency in the analysis model is uncovered. This generates the event analysis model correction which will trigger the analysis activity from the done state into the awaiting changes state.

Concurrent Model is applicable to all types of software development and provides an accurate picture of the current state of a project.

Rather than confining software engineering activities, actions and tasks to a sequence of events, it defines a network of activities. Each activity, action or task on the network exists simultaneously with other activities, actions or tasks.

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

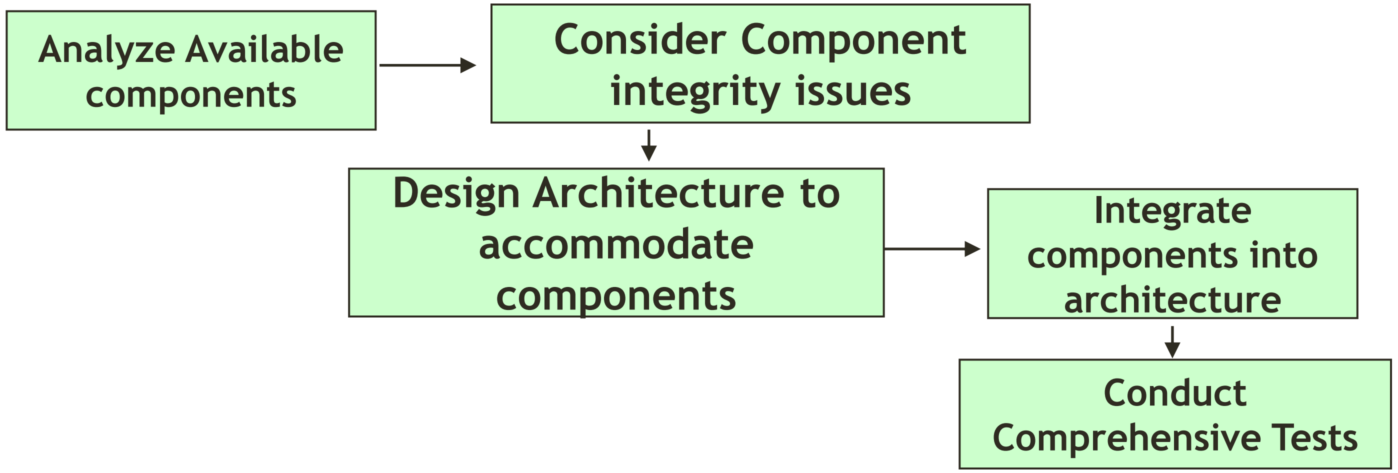
PRESCRIPTIVE PROCESS MODELS



COMPONENT BASED DEVELOPMENT

Component based development is an approach to software development that focuses on the design and development of reusable components.

It not only identifies candidate components but also qualifies each component's interface, adapts components to remove architectural mismatches, assembles components into a selected architectural style, and updates components as requirements for the system change.



Analyze Available components
Consider Component integrity issues
Design Architecture to accommodate components
Integrate components into architecture
Conduct Comprehensive Tests

The model leads to software reuse and resuability provides with number of measurable benefits

It is estimated that component based development leads to a 70% reduction in development cycle time and 84% reduction in software costs

THE FORMAL METHODS MODEL

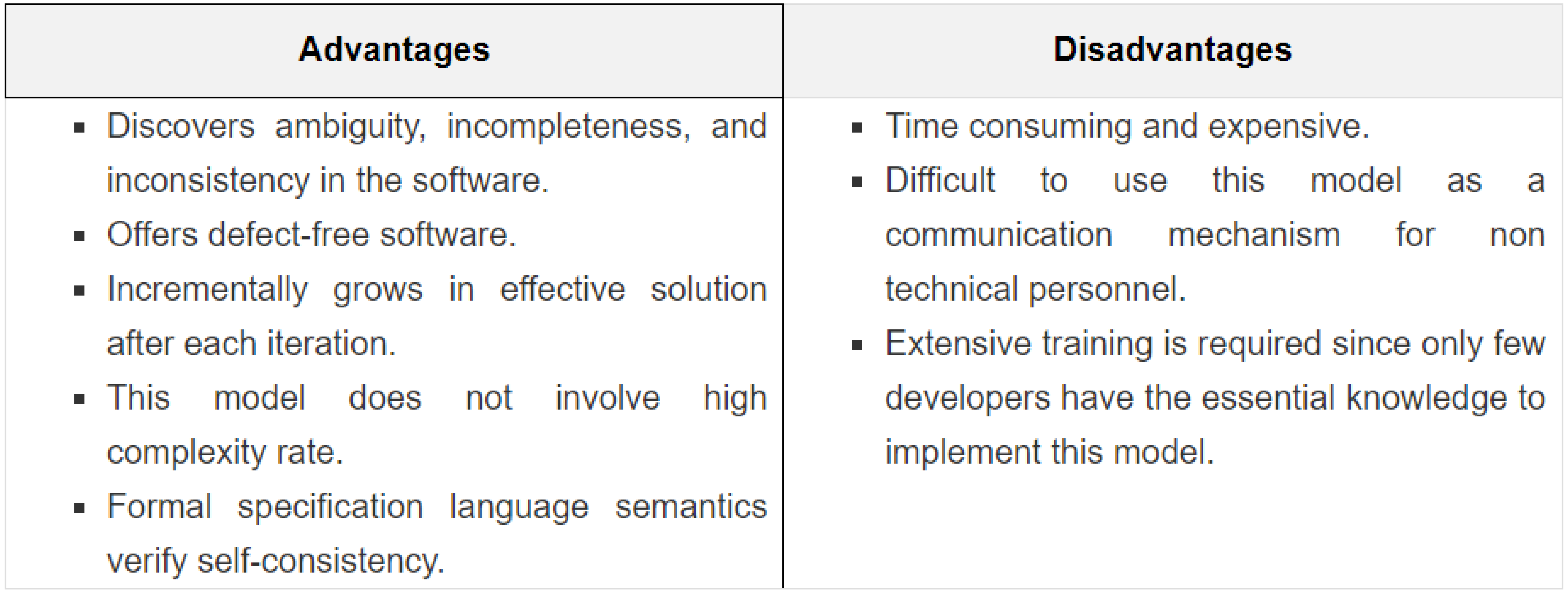
The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software.

Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.

A variation on this approach, called cleanroom software engineering

They rely on math and logic to model and analyze system behavior, making systems more reliable and secure.

THE FORMAL METHODS MODEL



Advantages	Disadvantages
▪ Discovers ambiguity, incompleteness, and inconsistency in the software. ▪ Offers defect-free software. ▪ Incrementally grows in effective solution after each iteration. ▪ This model does not involve high complexity rate. ▪ Formal specification language semantics verify self-consistency.	▪ Time consuming and expensive. ▪ Difficult to use this model as a communication mechanism for non technical personnel. ▪ Extensive training is required since only few developers have the essential knowledge to implement this model.

ASPECT ORIENTED SOFTWARE DEVELOPMENT

An approach to software development based around a new type of abstraction - an aspect.

Used in conjunction with other approaches - normally object-oriented software engineering.

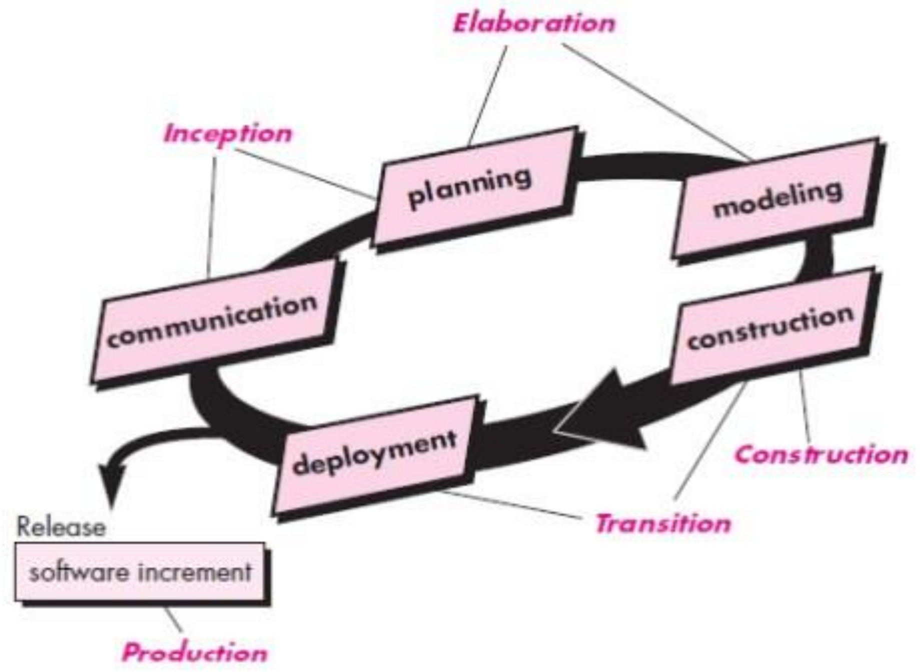
Aspects encapsulate functionality that cross-cuts and co-exists with other functionality.

Aspects include a definition of where they should be included in a program as well as code implementing the cross-cutting concern.

THE UNIFIED PROCESS

The inception phase of UP encompasses both customer communication and planning activities. Fundamental business requirements are described through a series of use-cases that describe what features and functions are desired by each major class of users.

Elaboration refines and expands preliminary use-cases to include five diff. views of the software , use-case model analysis model, design model, implementation model and deployment model

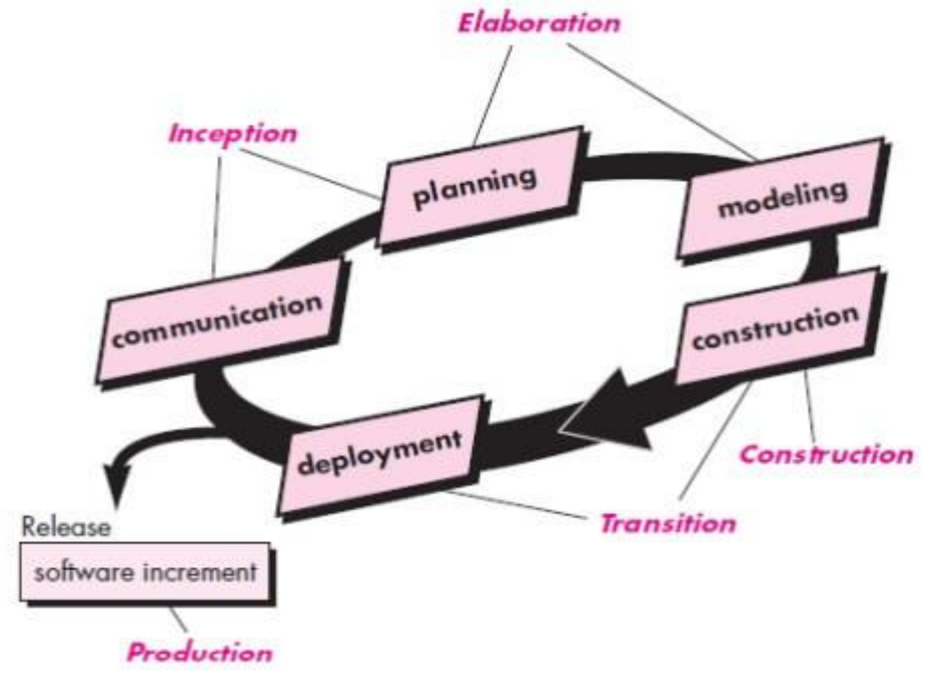


THE UNIFIED PROCESS

The construction phase is identical to the construction activity defined for the generic software process. Using architectural model as inputs, the construction phase develops or acquires software components that will make use-case operational for end-users

In the transition phase, software is given to the users for beta testing, and user feedback reports both defects and necessary changes.

During the Production phase, the on-going use of the software is monitored, support for the operating environment is provided



THE UNIFIED PROCESS

Inception

The main goal of this phase involves delimiting the project scope. This is where we define why we are making this product in the first place. It should have the following:

- What are the key features?
- How does this benefit the customers?
- Which methodology will we follow?
- What are the risks involved in executing the project?
- Schedule and cost estimates.

THE UNIFIED PROCESS

Elaboration

We build the system given the requirements, cost, and time constraints and all the risks involved. It should include the following:

- Develop with the majority of the functional requirements implemented.
- Finalize the methodology to be used.
- Deal with the significant risks involved.

Construction

This phase is where the development, integration, and testing take place. We build the complete architecture in this phase and hand the final documentation to the client.

Transition

This phase involves the deployment, multiple iterations, beta releases, and improvements of the software. The users will test the software, which may raise potential issues. The development team will then fix those errors.

SUMMARY – WATERFALL

Strength	Weakness	Types of Projects
Simple Easy to execute Intuitive and logical Easy contractually	All or nothing – too risky Requirement frozen early May chose outdated hardware/tech Disallows changes No feedback from users Encourages requirement creep	Well understood problems, short duration projects, automation of existing manual systems

SUMMARY – PROTOTYPING

Strength	Weakness	Types of Projects
Helps requirement elicitation Reduces risk Better and more stable final system	Front heavy Possibly higher cost and schedule Encourages requirement creeping Disallows later change	Systems with novice users; or areas with requirement uncertainty. Heavy reporting based systems can benefit from UI proto

SUMMARY – ITERATIVE

Strength	Weakness	Types of Projects
Regular deliveries, leading to biz benefit Can accommodate changes naturally Allows user feedback Avoids requirement creep Naturally prioritizes requirement Allows reasonable exit points Reduces risks	Overhead of planning each iteration Total cost may increase System arch and design may suffer Rework may increase	For businesses where time is imp; risk of long projects cannot be taken; requirement not known and evolve with time