Software Engineering

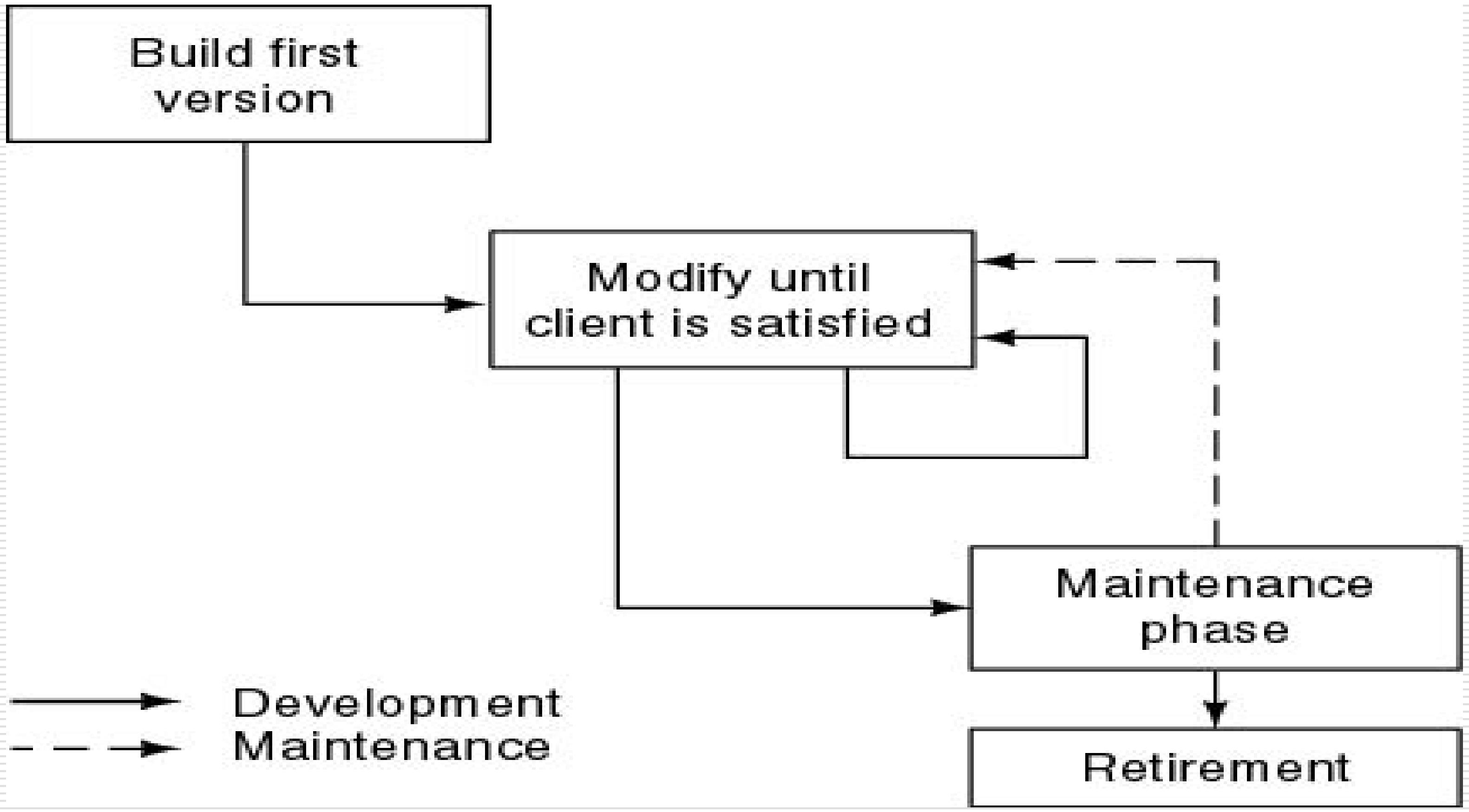
Lecture 02

Software Process Model

Software process model

- Process models prescribe a distinct set of activities, actions, tasks, milestones, and work products required to engineer high quality software.
- Process models are not perfect, but provide roadmap for software engineering work.
- Software models provide stability, control, and organization to a process that if not managed can easily get out of control
- Software process models are adapted to meet the needs of software engineers and managers for a specific project.

Build and Fix Model



Build and Fix Model

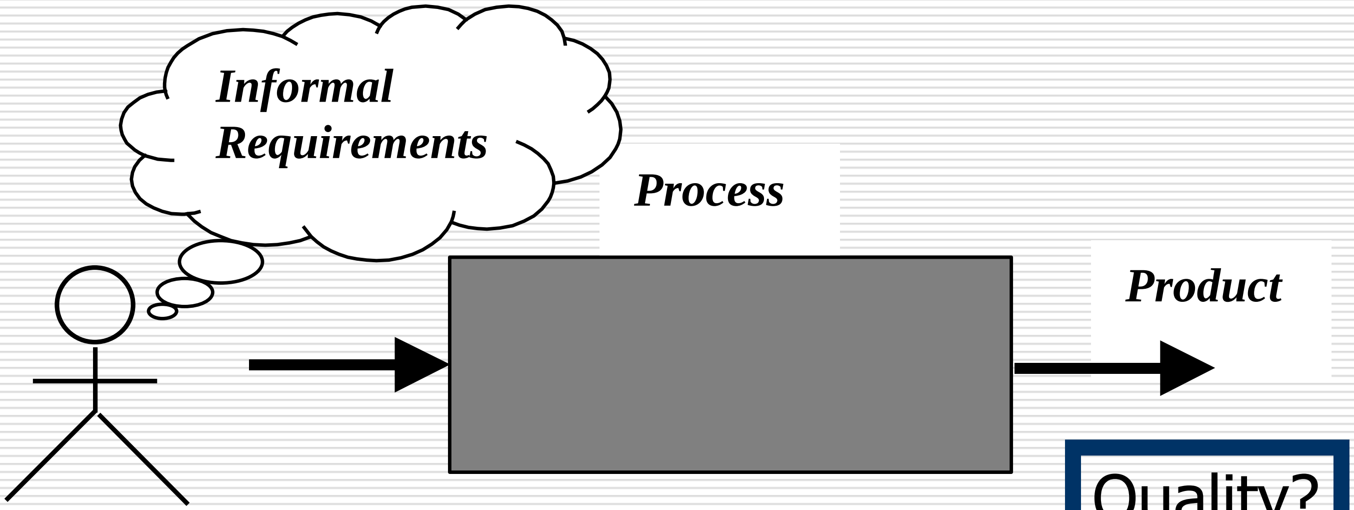
The earlier approach

- Product is constructed without specification or any attempt at design.
- developers simply build a product that is reworked as many times as necessary to satisfy the client.
- model may work for small projects but is totally unsatisfactory for products of any reasonable size.
- Maintenance is high.
- Source of difficulties and deficiencies
  - impossible to predict
  - impossible to manage

Why Models are needed?

- Symptoms of inadequacy: the software crisis
  - scheduled time and cost exceeded
  - user expectations not met
  - poor quality

Process as a "black box"



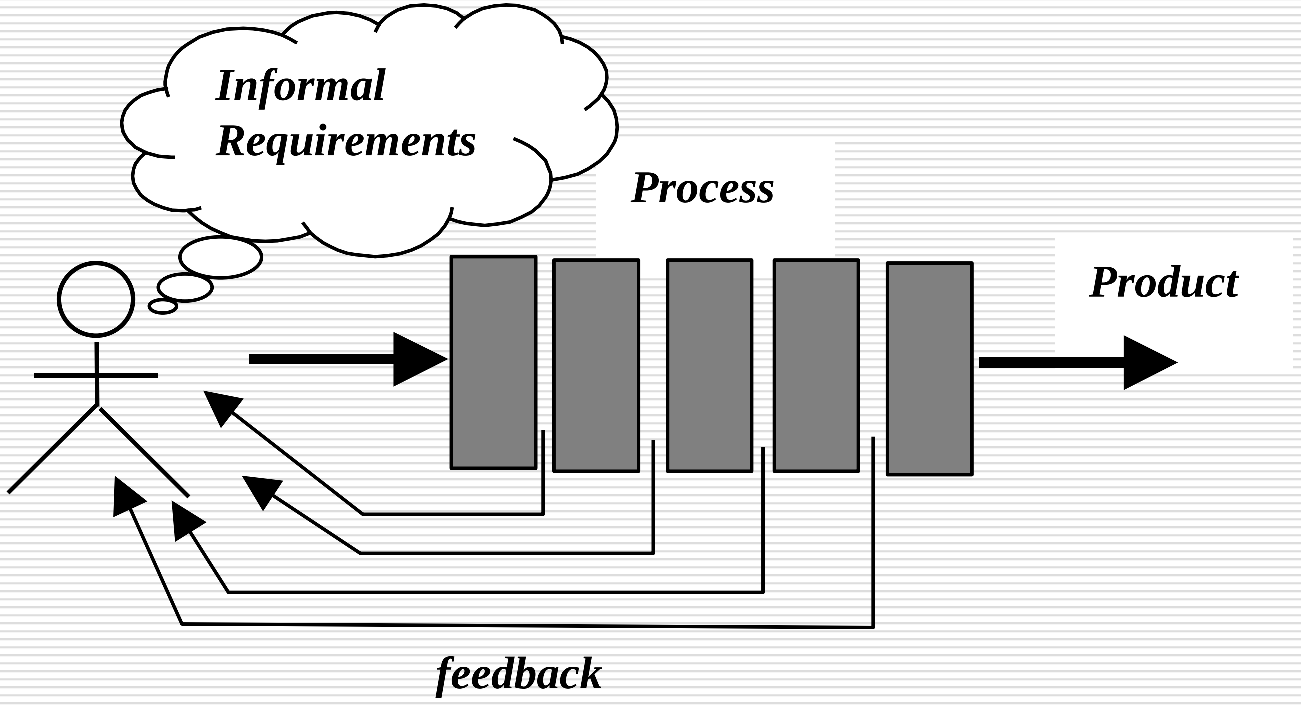
Uncertain /
Incomplete requirement
In the beginning

Quality?

Problems

- The assumption is that requirements can be fully understood prior to development
- Interaction with the customer occurs only at the beginning (requirements) and end (after delivery)
- Unfortunately the assumption almost never holds

Process as a "white box"



Advantages

- Reduce risks by improving visibility
- Allow project changes as the project progresses
  - based on feedback from the customer

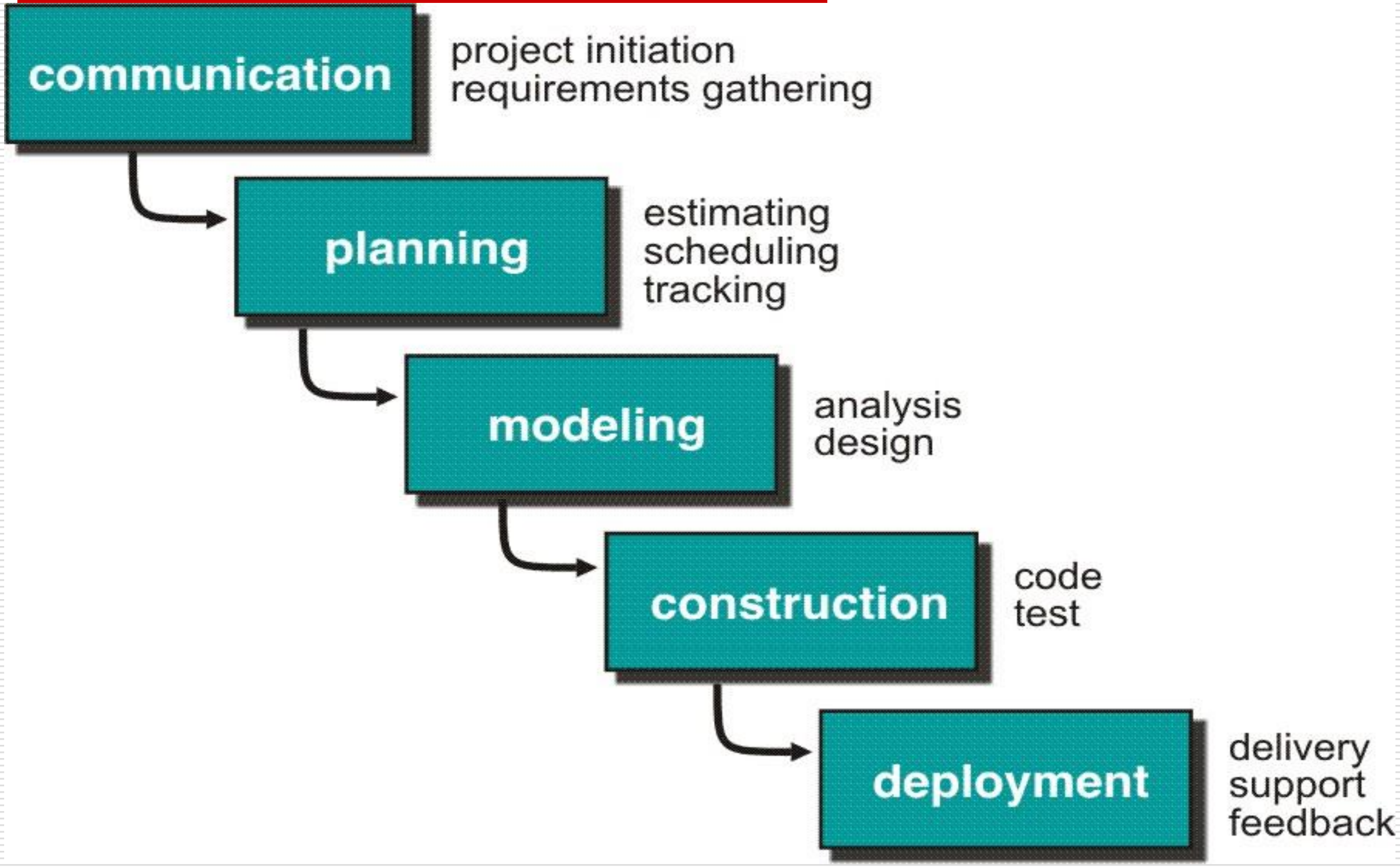
Prescriptive Model

- Prescriptive process models advocate an orderly approach to software engineering
  - Organize framework activities in a certain order
- Process framework activity with set of software engineering actions.
- Each action in terms of a task set that identifies the work to be accomplished to meet the goals.
- The resultant process model should be adapted to accommodate the nature of the specific project, people doing the work, and the work environment.
- Software engineer choose process framework that includes activities like;
  - Communication
  - Planning
  - Modeling
  - Construction
  - Deployment

Prescriptive Model

- Calling this model as “Prescribe” because it recommend a set of process elements, activities, action task, work product & quality.
- Each elements are inter related to one another (called workflow).

Waterfall Model or Classic Life Cycle



Waterfall Model or Classic Life Cycle

- Requirement Analysis and Definition: What - The systems services, constraints and goals are defined by customers with system users.
- Scheduling tracking -
  - Assessing progress against the project plan.
  - Require action to maintain schedule.
- System and Software Design: How –It establishes and overall system architecture. Software design involves fundamental system abstractions and their relationships.
- Integration and system testing: The individual program unit or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- Operation and Maintenance: Normally this is the longest phase of the software life cycle. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life-cycle.

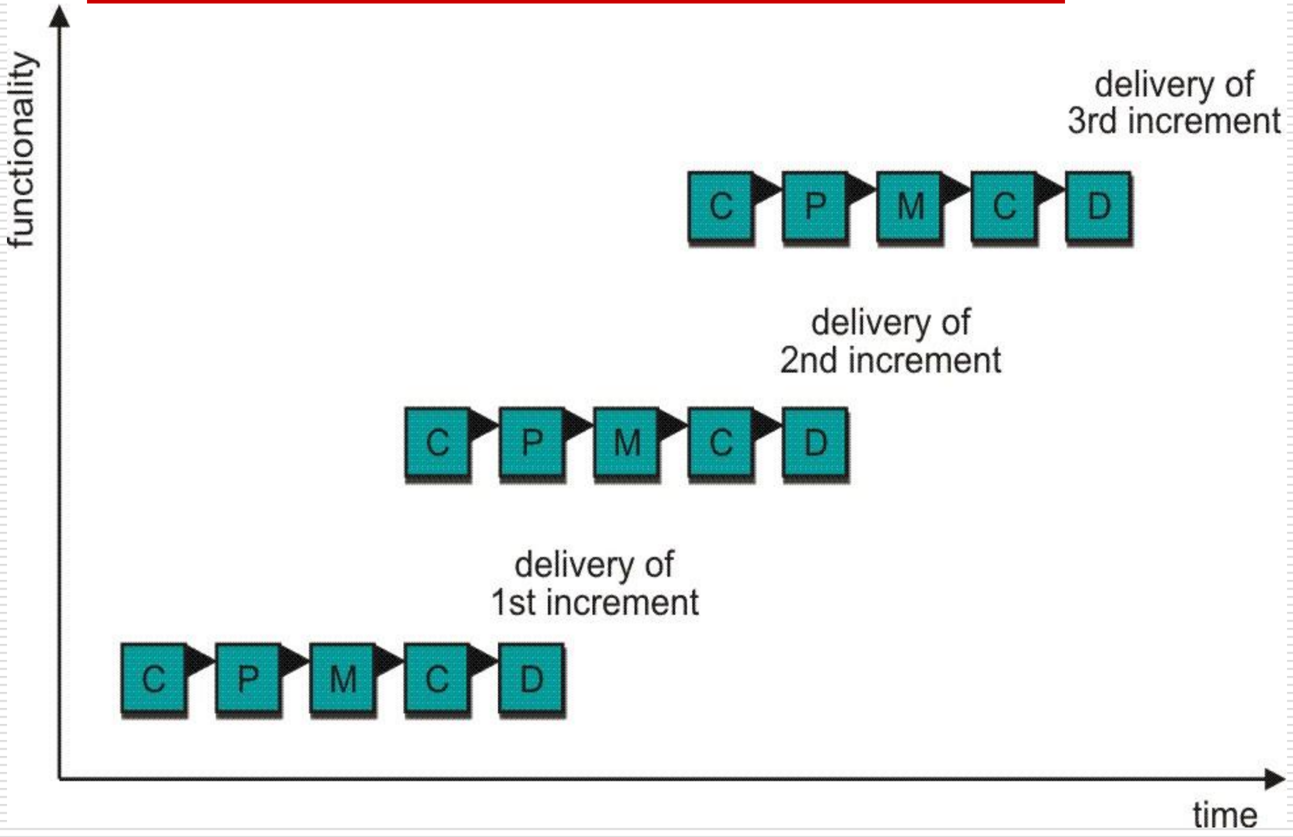
Limitations of the waterfall model

- The nature of the requirements will not change very much During development; during evolution
- The model implies that you should attempt to complete a given stage before moving on to the next stage
  - Does not account for the fact that requirements constantly change.
  - It also means that customers can not use anything until the entire system is complete.
- The model implies that once the product is finished, everything else is maintenance.
- Surprises at the end are very expensive
- Some teams sit ideal for other teams to finish
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

Problems:

1. Real projects are rarely follow the sequential model.
2. Difficult for the customer to state all the requirement explicitly.
3. Assumes patience from customer - working version of program will not available until programs not getting change fully.

Incremental Process Model


functionality
delivery of 3rd increment
C
P
M
C
D
delivery of 2nd increment
C
P
M
C
D
delivery of 1st increment
C
P
M
C
D
time

C- Communication
P - Planning
M – Modeling
C - Construction
D - Deployment

Delivers software in small but usable pieces, each piece builds on pieces already delivered

The Incremental Model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- First Increment is often core product
  - Includes basic requirement
  - Many supplementary features (known & unknown) remain undelivered
- A plan of next increment is prepared
  - Modifications of the first increment
  - Additional features of the first increment
- It is particularly useful when enough staffing is not available for the whole project
- Increment can be planned to manage technical risks.
- Incremental model focus more on delivery of operation product with each increment.

The Incremental Model

- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Rational Unified Process

- Rational Unified Process, or RUP, is a configurable software development process platform that delivers practices and a configurable architecture
- RUP is a method of managing OO Software Development
- Enables the developers to select and deploy only the process components they need for each stage of their project

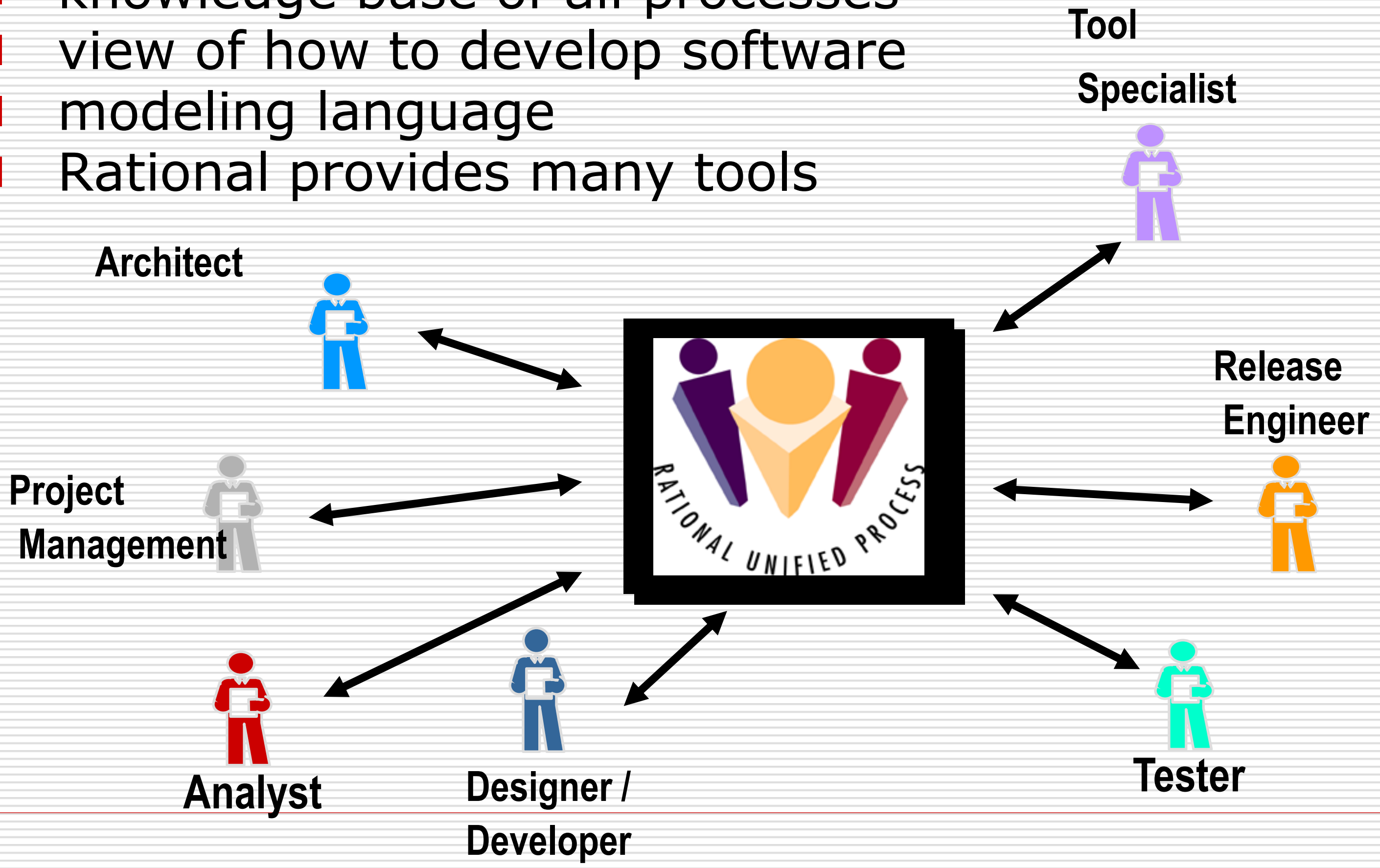
Rational Unified Process

- It can be viewed as a Software Development Framework which is extensible and features:
  - Iterative Development
  - Requirements Management
  - Component-Based Architectural Vision
  - Visual Modeling of Systems
  - Quality Management
  - Change Control Management

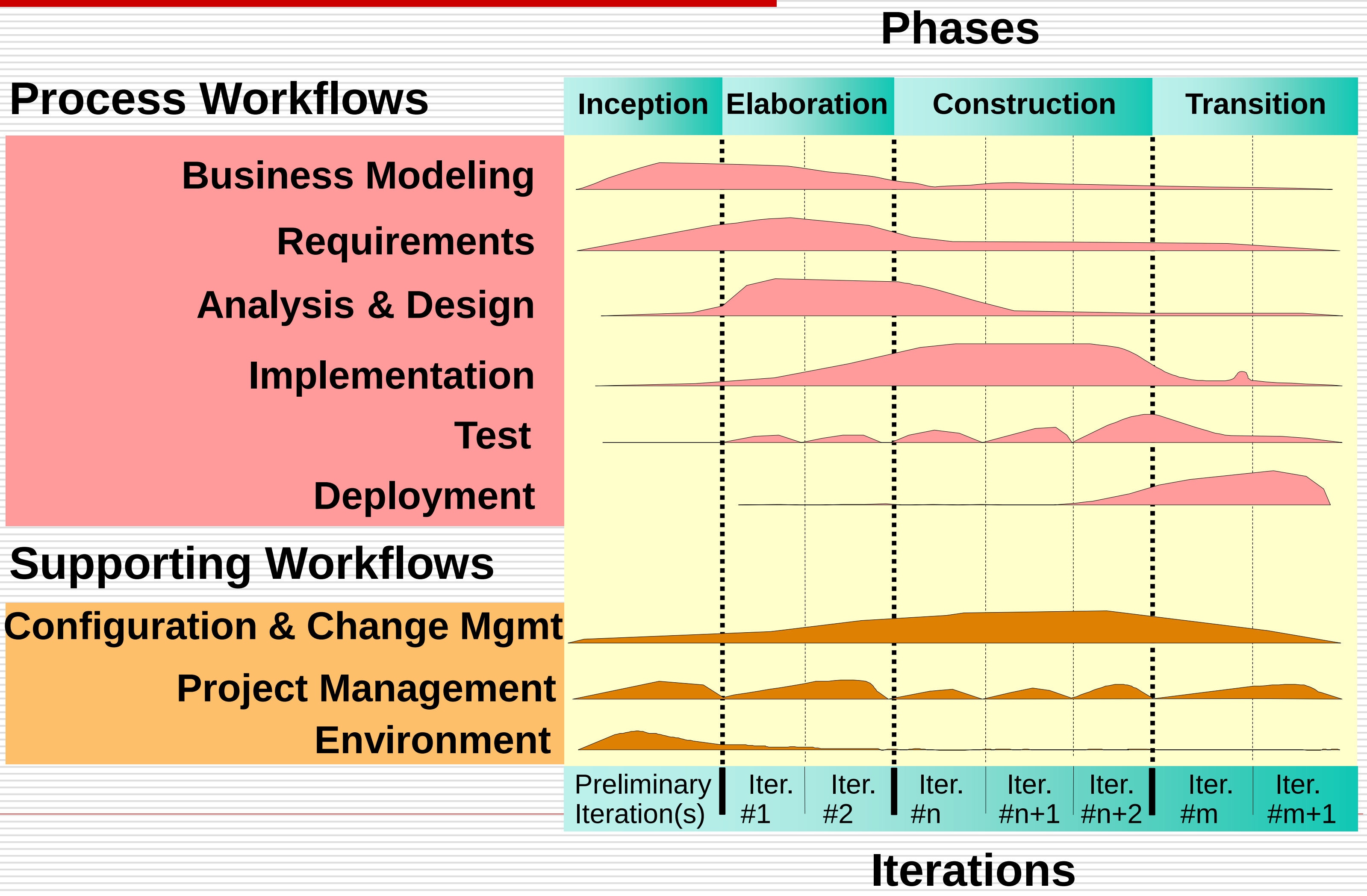
- Team-Unifying Approach

  The RUP unifies a software team by providing a common view of the development process and a shared vision of a common goal

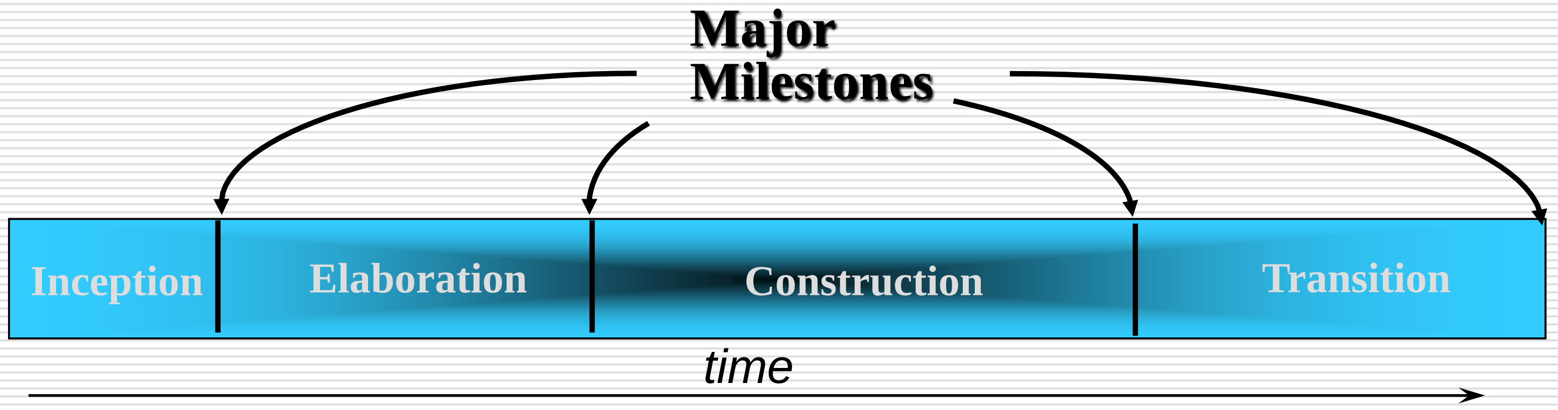
- Increased Team Productivity
  - knowledge base of all processes
  - view of how to develop software
  - modeling language
  - Rational provides many tools



Rational Unified Process (RUP)



Phases in RUP



The Rational Unified Process has four phases:

- Inception - Define the scope of project
- Elaboration - Plan project, specify features, baseline architecture
- Construction - Build the product
- Transition - Transition the product into end user community

Inception phase

- Establishing the project's software scope and boundary conditions, including an operational vision, acceptance criteria and what is intended to be in the product and what is not.
- Discriminating the critical use cases of the system, the primary scenarios of operation that will drive the major design tradeoffs.
- Exhibiting, and maybe demonstrating, at least one candidate architecture against some of the primary scenarios.
- Estimating the overall cost and schedule for the entire project (and more detailed estimates for the elaboration phase that will immediately follow).
- Estimating potential risks (the sources of unpredictability)
- Preparing the supporting environment for the project.

Elaboration phase

- Defining, validating and baselining the architecture as rapidly as practical.
- Refining the Vision, based on new information obtained during the phase, establishing a solid understanding of the most critical use cases that drive the architectural and planning decisions.
- Creating and baselining detailed iteration plans for the construction phase.
- Refining the development case and putting in place the development environment, including the process, tools and automation support required to support the construction team.
- Refining the architecture and selecting components. Potential components are evaluated and the make/buy/reuse decisions sufficiently understood to determine the construction phase cost and schedule with confidence. The selected architectural components are integrated and assessed against the primary scenarios.

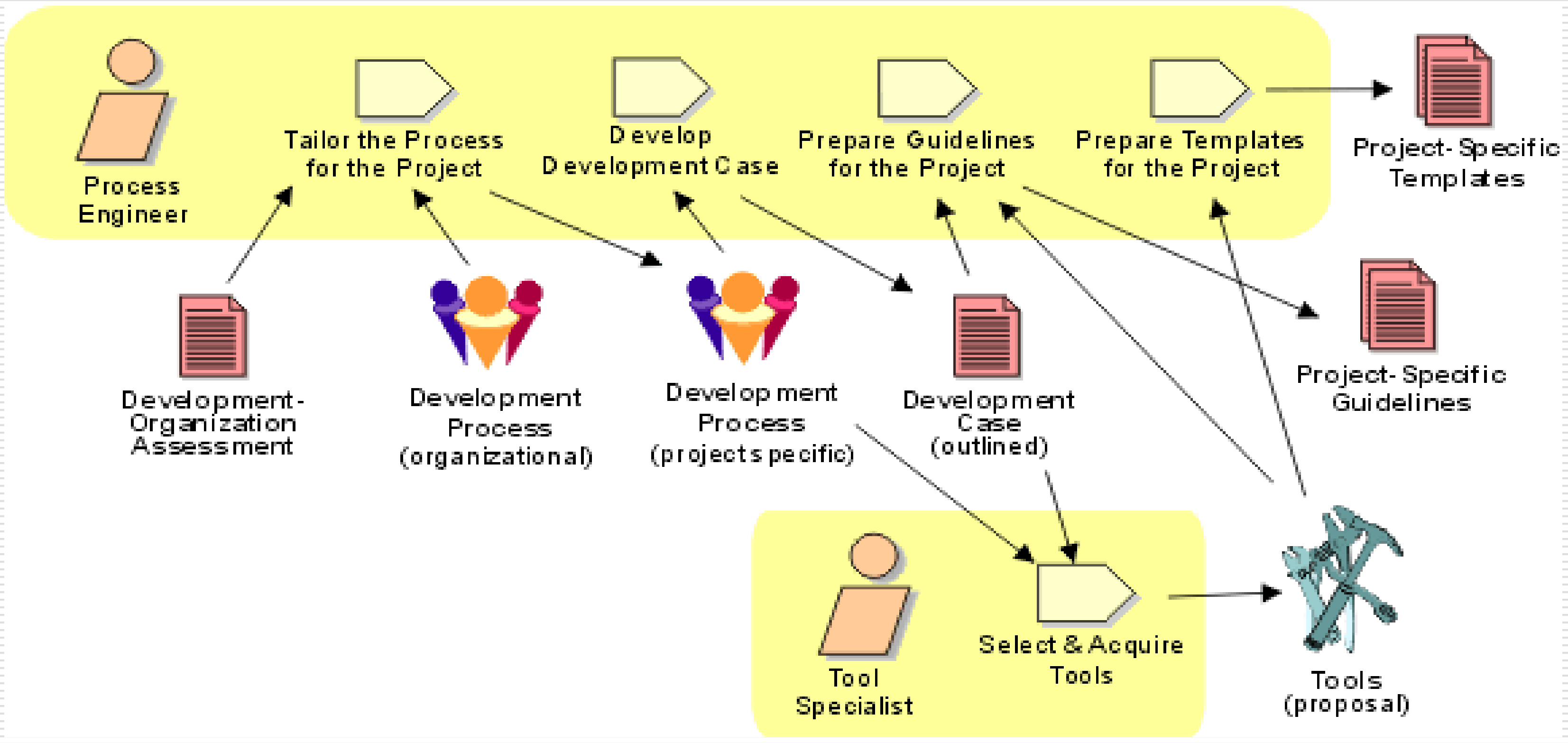
Construction phase

- Resource management, control and process optimization
- Complete component development and testing against the defined evaluation criteria
- Assessment of product releases against acceptance criteria for the vision.

Transition phase

- Executing deployment plans.
- Finalizing end-user support material.
- Testing the deliverable product at the development site.
- Creating a product release.
- Getting user feedback.
- Fine-tuning the product based on feedback.
- Making the product available to end users.

Workflow Detail: Prepare Environment for Project


Process Engineer
Tailor the Process for the Project
Develop Development Case
Prepare Guidelines for the Project
Prepare Templates for the Project
Project-Specific Templates
Development-Organization Assessment
Development Process (organizational)
Development Process (projectspecific)
Development Case (outlined)
Project-Specific Guidelines
Tool Specialist
Select & Acquire Tools
Tools (proposal)

Workflows - 3 key elements

- Three key elements of each workflows:
  - Artifacts
  - Roles
  - Activities

Artifacts

A piece of information that:

- Is produced, modified, or used by a process
- Defines an area of responsibility
- Is subject to version control.

An artifact can be a *model*, a *model element*, or a *document*. A document can enclose other documents.

Roles

- Represent a role that an individual may play on the project
- Responsible for producing artifacts
- Distinct from actors

Activities

- Tasks performed by people representing particular roles in order to produce artifacts

Brief summary of process workflows

- Business Modelling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

Business Modelling

- Understand structure & dynamics of organization in which system is to be deployed
- Understand current problems in the target organization & identify improvement potential
- Ensure customers, end users & developers have common understanding of target organisation
- Derive system requirements to support target organisation

Analysis & Design

- Transform requirements into a design of the system
- Evolve a robust architecture for the system
- Adapt design to match the implementation environment, designing it for performance

Implementation

- Define organization of the code, in terms of implementation subsystems organized in layers
- Implement classes & objects in terms of components
- Test developed components as units
- Integrate results into an executable system

Test

- Verify interaction between objects
- Verify proper integration of all components of the software
- Verify that all requirements have been correctly implemented
- Identify & ensure defects are addressed prior to deployment

Deployment

- Provide custom installation
- Provide shrink wrap product offering
- Provide software over internet

Brief summary of supporting workflows

- Configuration & Change Management
- Project Management
- Environment

Configuration & Change Management

- Supports development methods
- Maintains product integrity
- Ensures completeness & correctness of configured product
- Provides stable environment within which to develop product
- Restricts changes to artifacts based on project policies
- Provides an audit trail on why, when & by whom any artifact was changed

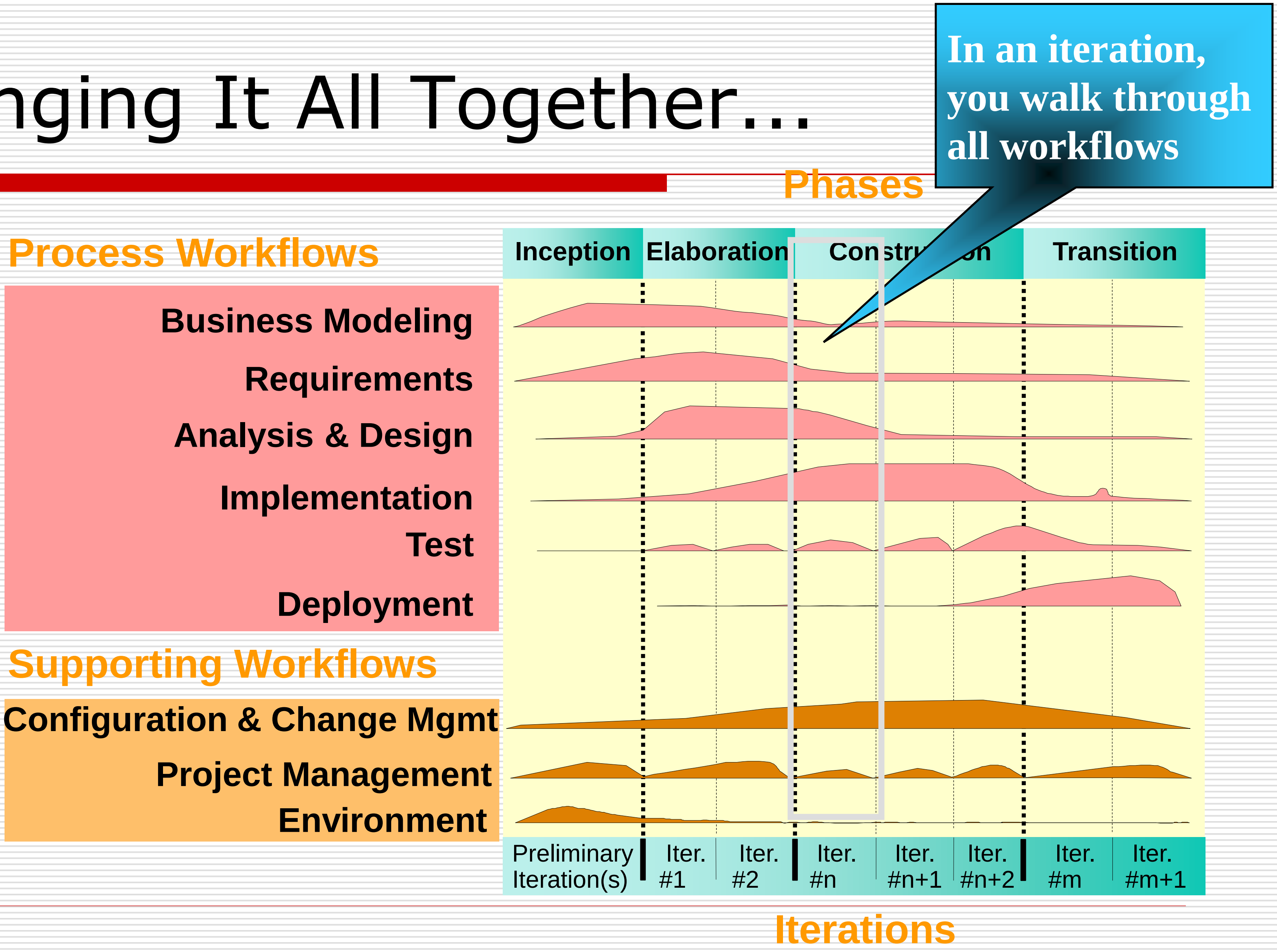
Project Management

- A framework for managing software-intensive projects
- Practical guidelines for planning, staffing, executing & monitoring projects
- A framework for managing risk

Environment

- Design, implement and manage the project’s required technical environments
- Define the technical architectures for the development, system validation, testing & staging/release management environments
- When possible, standard architectural models for given types of platforms should be utilized when defining the production environment

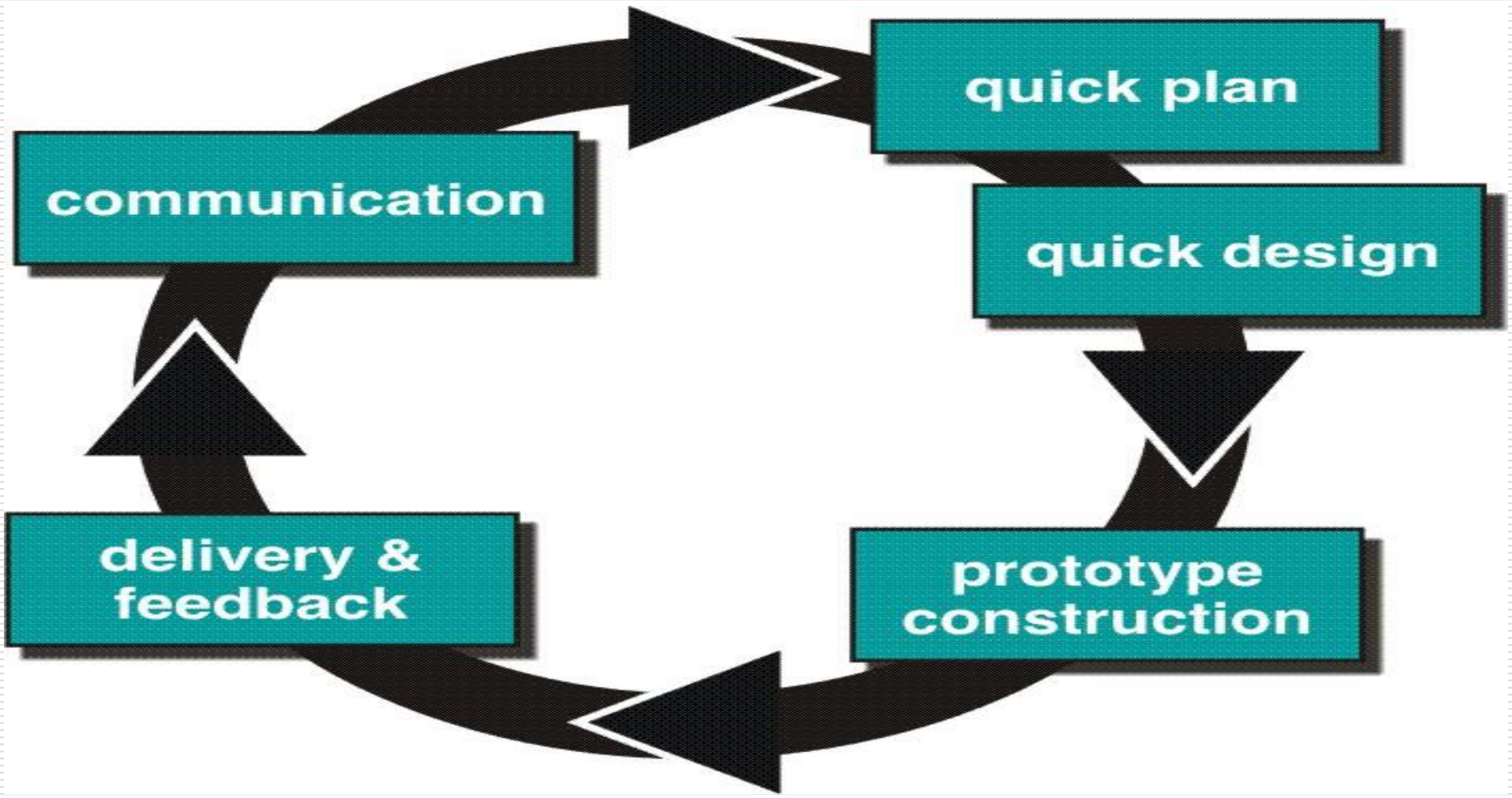
Bringing It All Together...



Evolutionary Process Model

- Produce an increasingly more complete version of the software with each iteration.
- Evolutionary Models are iterative.
- Evolutionary models are:
  - Prototyping
  - Spiral Model
  - Concurrent Development Model
  - Fourth Generation Techniques (4GT)

Evolutionary Process Models : Prototyping



Prototyping cohesive

- **Best approach when**:
  - Objectives defines by customer are general but does not have details like input, processing, or output requirement.
  - Developer may be unsure of the efficiency of an algorithm, O.S., or the form that human machine interaction should take.
- It can be used as standalone process model.
- Model assist software engineer and customer to better understand what is to be built when requirement are fuzzy.
- Prototyping start with communication, between a customer and software engineer to define overall objective, identify requirements and make a boundary.
- Going ahead, planned quickly and modeling (software layout visible to the customers/end-user) occurs.
- Quick design leads to prototype construction.
- Prototype is deployed and evaluated by the customer/user.
- Feedback from customer/end user will refine requirement and that is how iteration occurs during prototype to satisfy the needs of the customer.

Prototyping (cont..)

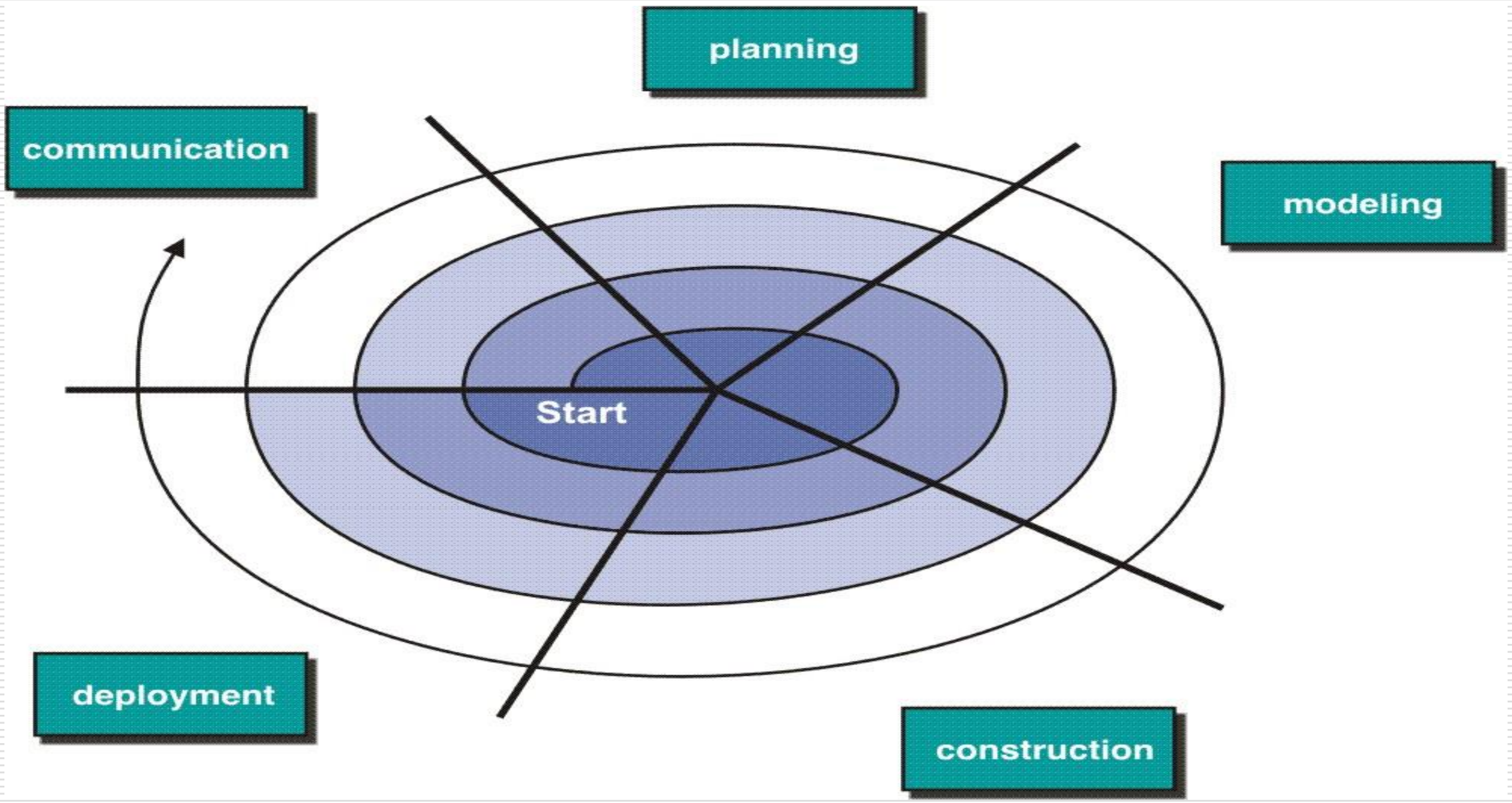
- Prototype can be serve as “the first system”.
- Both customers and developers like the prototyping paradigm.
  - Customer/End user gets a feel for the actual system
  - Developer get to build something immediately.

Problem Areas:

- Customer cries foul and demand that “a few fixes” be applied to make the prototype a working product, due to that software quality suffers as a result.
- Developer often makes implementation in order to get a prototype working quickly without considering other factors in mind like OS, Programming language, etc.

Customer and developer both must be agree that the prototype is built to serve as a mechanism for defining requirement.

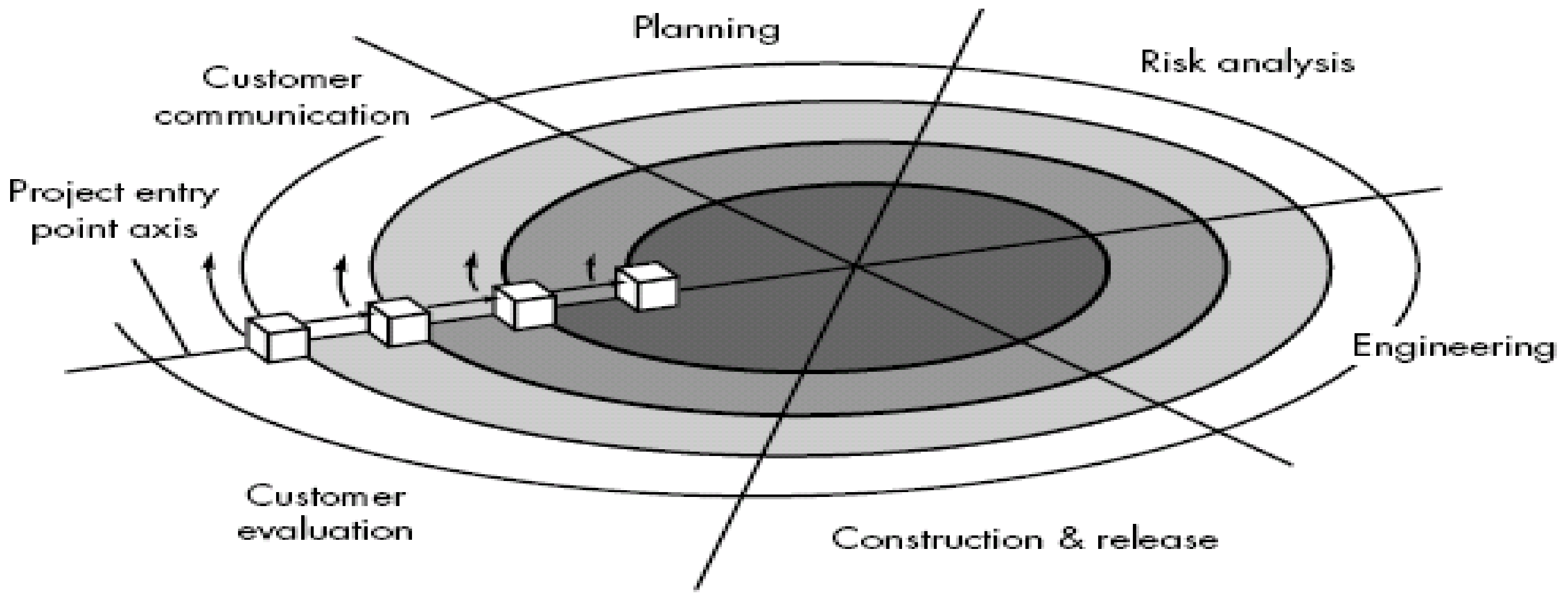
Evolutionary Model: Spiral Model



Spiral Model

- Couples iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model
- It provide potential for rapid development of increasingly more complete version of the software.
- Using spiral, software developed in as series of evolutionary release.
  - Early iteration, release might be on paper or prototype.
  - Later iteration, more complete version of software.
- Divided into framework activities (C,P,M,C,D). Each activity represent one segment.
- Evolutionary process begins in a clockwise direction, beginning at the center risk.
- First circuit around the spiral might result in development of a product specification. Subsequently, develop a prototype and then progressively more sophisticated version of software.
- Unlike other process models that end when software is delivered.
- It can be adapted to apply throughout the life of software.

Spiral Model



□	Product maintenance projects
░	Product enhancement projects
▒	New product development projects
▓	Concept development projects

Spiral Model (cont.)

Concept Development Project:

- Start at the core and continues for multiple iterations until it is complete.
- If concept is developed into an actual product, the process proceeds outward on the spiral.

New Product Development Project:

- New product will evolve through a number of iterations around the spiral.
- Later, a circuit around spiral might be used to represent a "Product Enhancement Project"

Product Enhancement Project:

- There are times when process is dormant or software team not developing new things but change is initiated, process start at appropriate entry point.

- Spiral models uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at each stage in the evolution of the product.

- It maintains the systematic stepwise approach suggested by the classic life cycle but also incorporates it into an iterative framework activity.
- If risks cannot be resolved, project is immediately terminated

Problem Area:

- It may be <u>difficult to convince customers</u> (particularly in contract situations) that the evolutionary approach is controllable.
- If a major risk is <u>not uncovered and managed</u>, problems will undoubtedly occur.

Agile Process

- What is “Agility”?
  - Effective (rapid and adaptive) response to change
  - Effective communication among all stakeholders
  - Drawing the customer onto the team
  - Organizing a team so that it is in control of the work performed

  Yielding …
  - Rapid, incremental delivery of software

An Agile Process

- Is driven by customer descriptions of what is required (scenarios)
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple ‘software increments’
- Adapts as changes occur

Agility Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

Agility Principles (cont.)

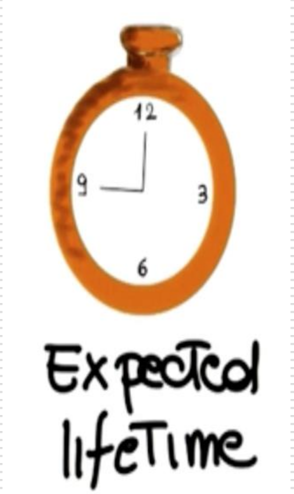
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Unified Process Model

- Unified Process is component-based, which means that the software system being built is made up of software components interconnected via well-defined interfaces in UML

Choosing a SDLC Model

Requirements Understanding



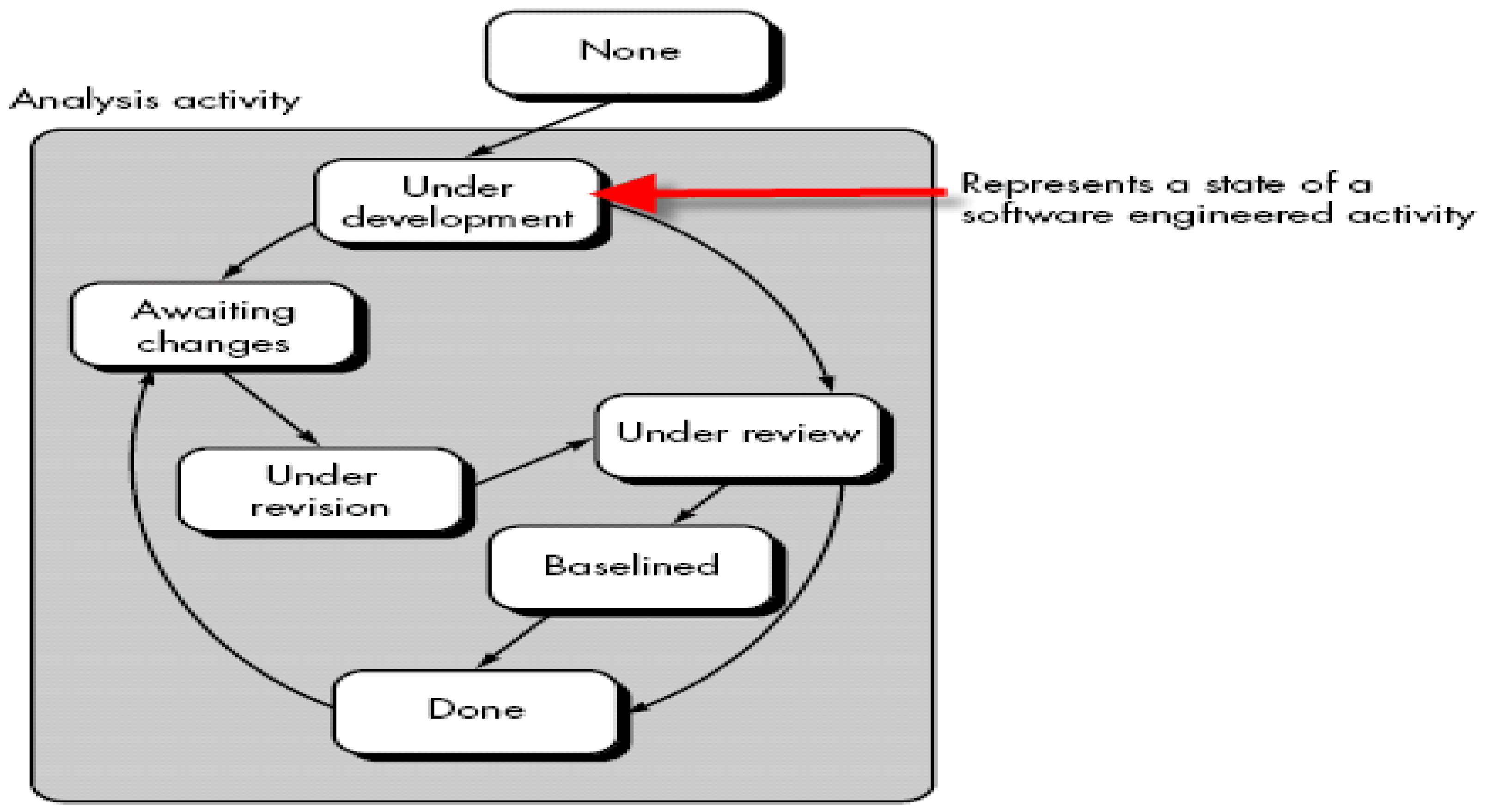
June 2021
Schedule Constraints

BLA BLA BLA
Interaction with management/customers

Choosing a SDLC Model

- Is the SDLC suitable for the size of our team and their skills?
- Is the SDLC suitable for the selected technology we use for implementing the solution?
- Is the SDLC suitable for client and stakeholders concerns and priorities?
- Is the SDLC suitable for the geographical situation (distributed team)?
- Is the SDLC suitable for the size and complexity of our software?
- Is the SDLC suitable for the type of projects we do?
- Is the SDLC suitable for our software engineering capability?
- Is the SDLC suitable for the project risk and quality insurance?

Concurrent Development Model



Concurrent Development Model

- It represented schematically as series of major technical activities, tasks, and their associated states.
- It is often more appropriate for system engineering projects where different engineering teams are involved.
- The activity-modeling may be in any one of the states for a given time.
- All activities exist concurrently but reside in different states.

E.g.

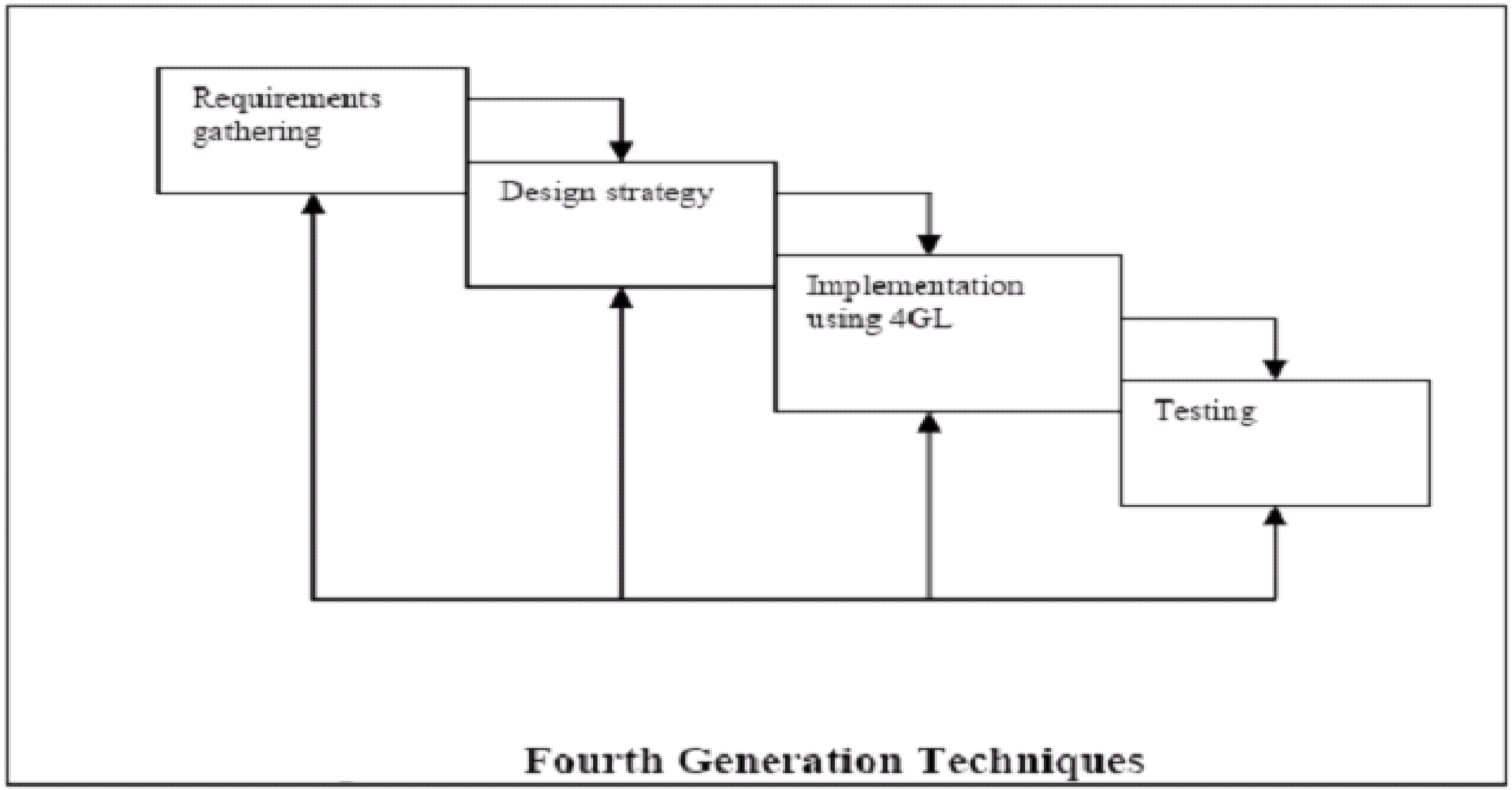
- The *analysis* activity (existed in the **none** state while initial customer communication was completed) now makes a transition into the **under development** state.
- *Analysis* activity moves from the **under development** state into the **awaiting changes** state only if customer indicates changes in requirements.
- Series of event will trigger transition from state to state.

E.g. During initial stage there was inconsistency in design which was uncovered. This will triggers the analysis action from the **Done** state into **Awaiting Changes** state.

Concurrent Development (Cont.)

- Visibility of current state of project
- It define network of activities
- Each activities, actions and tasks on the network exists simultaneously with other activities ,actions and tasks.
- Events generated at one point in the process network trigger transitions among the states.

Fourth Generation Techniques(4GT)



Fourth Generation Techniques

4GT

- Like all other models, 4GT begins with a requirements gathering phase.
- Ideally, the customer would describe the requirements, which are directly translated into an operational prototype.
- Practically, however, the client may be unsure of the requirements, may be ambiguous in his specs or may be unable to specify information in a manner that a 4GT tool can use.
- For small applications, it may be possible to move directly from the requirements gathering phase to the implementation phase using a nonprocedural fourth generation language.
- However for larger projects a design strategy is necessary. Otherwise, the same difficulties are likely to arise as with conventional approaches.

4GT

- To transform a 4GT implementation into a product, the developer must conduct thorough testing, develop meaningful documentation.
- In addition, the 4GT developed software must be built in a manner that enables maintenance to be performed quickly.

Merits:

- Dramatic reduction in software development time. (For small and intermediate application)
- Improved productivity for software developers.

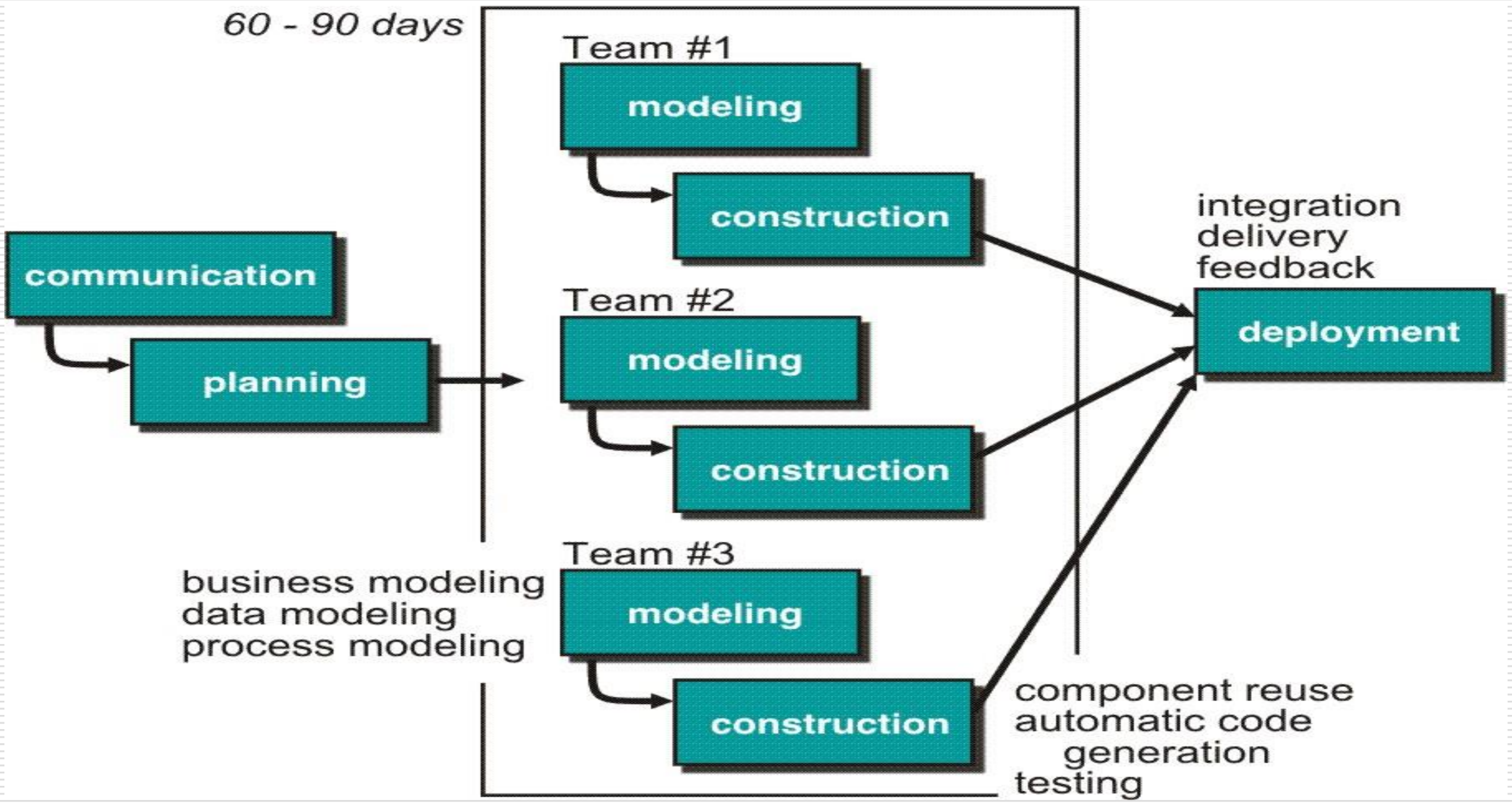
Demerits:

- Not much easier to use as compared to programming languages
- The maintainability of large software systems built using 4GT is open to question.

4GT

- 4GT Software tool is used to generate the source code for a software system from a high level specification representation
- Commonly used 4GT in development models are mentioned below:
  - Report Generation
  - Data base query language
  - Data Manipulation
  - Screen definition and interaction
  - Code Generation
  - Web engineering Tools
  - high-level graphics

Rapid Application Development (RAD) Model



Makes heavy use of reusable software components with an extremely short development cycle

RAD model

- **Communication** – to understand business problem.
- **Planning** – multiple s/w teams works in parallel on diff. system.
- **Modeling** –
  - **Business modeling** – Information flow among business is working.

    Ex. What kind of information drives?

    Who is going to generate information?

    From where information comes and goes?
  - **Data modeling** – Information refine into set of data objects that are needed to support business.
  - **Process modeling** – Data object transforms to information flow necessary to implement business.

- **Construction –** it highlighting the use of pre-existing software component.
- **Deployment –** Deliver to customer basis for subsequent iteration.
- RAD model emphasize a short development cycle.
- “High speed” edition of linear sequential model.
- If requirement are well understood and project scope is constrained then it enable development team to create “ fully functional system” within a very short time period.

RAD Model

- If application is modularized (“Scalable Scope”), each major function to be completed in less than three months.
- Each major function can be addressed by a separate team and then integrated to form a whole.

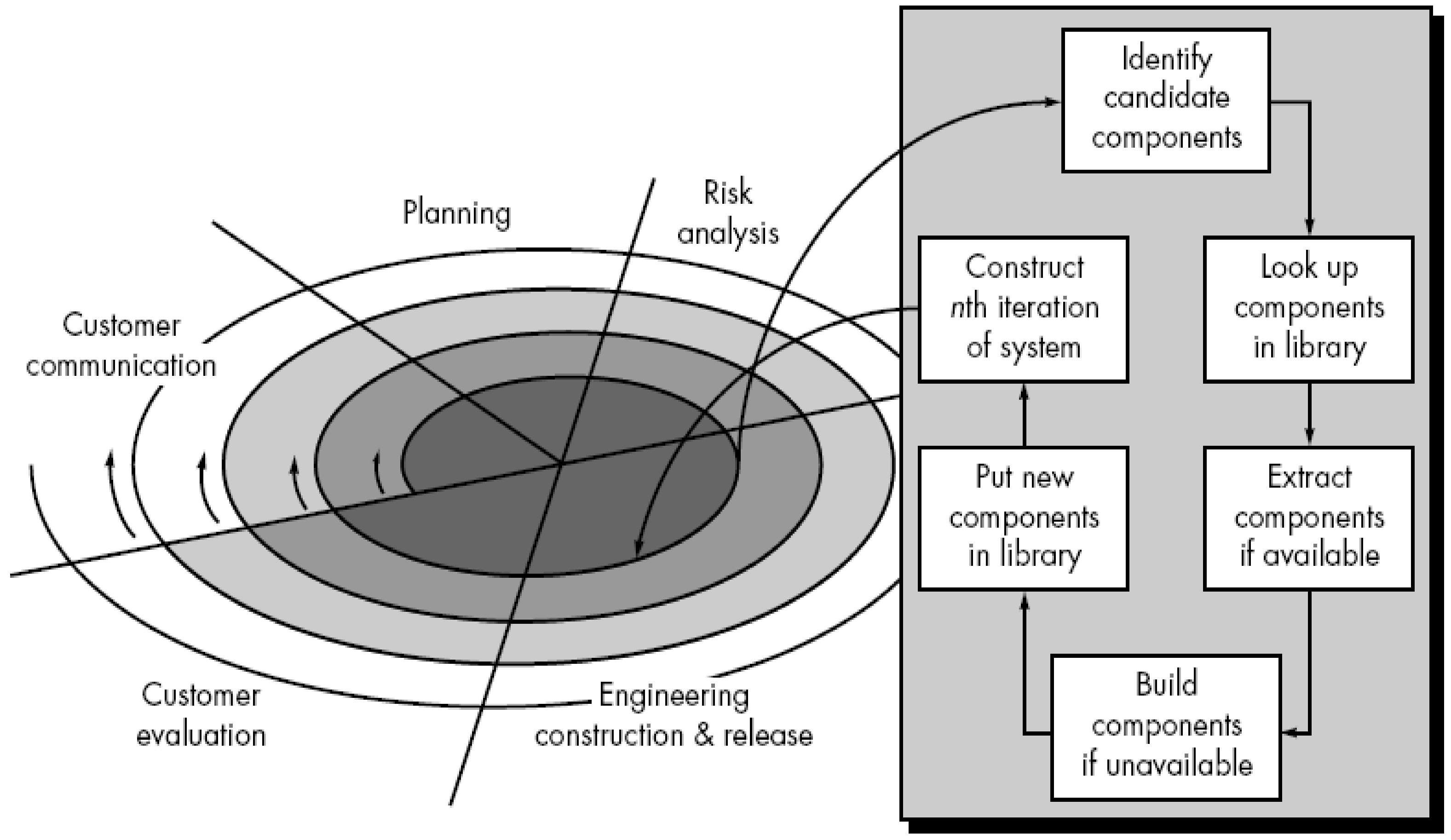
Drawback:

- For large but scalable projects
  - RAD requires sufficient human resources
- Projects fail if developers and customers are not committed in a much shortened time-frame
- Problematic if system can not be modularized
- Not appropriate when technical risks are high (heavy use of new technology)

Component Based Development

- component-based development (CBD) model incorporates many of the characteristics of the spiral model.
- It is evolutionary by nature and iterative approach to create software.
- CBD model creates applications from prepackaged software components (called *classes).*

CBD Model


Planning
Risk analysis
Customer communication
Customer evaluation
Engineering construction & release
Identify candidate components
Look up components in library
Extract components if available
Build components if unavailable
Put new components in library
Construct *n*th iteration of system

CBD model (cont.)

- Modeling and construction activities begin with identification of candidate components.
- Classes created in past software engineering projects are stored in a class library or repository.
- Once candidate classes are identified, the class library is searched to determine if these classes already exist.
- If class is already available in library extract and reuse it.
- If class is not available in library, it is engineered or developed using object-oriented methods.
- Any new classes built to meet the unique needs of the application.
- Now process flow return to the spiral activity.

CBD model (cont.)

- CBD model leads to software reusability.
- Based on studies, CBD model leads to 70 % reduction in development cycle time.
- 84% reduction in project cost.
- Productivity is very high.