Chapter 15, Software Life Cycle, Unified Process

Object-Oriented Software Engineering
Using UML, Patterns, and Java

Review of Definitions

- Software life cycle:
    - Set of activities and their relationships to each other to support the development of a software system

- Software development methodology:
    - A collection of techniques for building models - applied across the software life cycle

Software Life Cycle Questions (Review)

- Which activities should I select?
- What are the dependencies between activities?
- How should I schedule the activities?
- Questions to ask:
  - What is the problem?
  - What is the solution?
  - What are the mechanisms that best implement the solution?
  - How is the solution constructed?
  - Is the problem solved?
  - Can the customer use the solution?
  - How do we deal with changes that occur during the development? Are enhancements needed?

Life Cycle Modeling

- So far we have discussed the life cycle models
  - Waterfall model
  - V-model
  - Spiral model
  - Issue-based model
- Today we will introduce another life cycle model
  - Unified Software Process

Methodology issues

Methodologies provide general principles and strategies for selecting methods. High-level guidance is also useful when dealing with new situations. In this section, the issues for which methodologies provide guidance include:

- How much planning should be done in advance?
- How much of the design should result from reusing past solutions?
- How much of the system should be modeled before it is coded?
- In how much detail should the process be defined
- How often should the work be controlled and monitored?
- When should the project goals be redefined?

How Much Planning?

- Software project management plans are one of the keys in running a successful project. However, plans have limitations.
- If unexpected events occur, the plan is altered and corrective activities are introduced to still achieve the planned goal. This type of planning and controlling works well for assembly productions similar to those described by Taylor in the early 1920s for the production of many identical cars, but not for building systems that require creative problem solving and reactivity to change.
- In general, software project management plans are especially useful to kick off a software project and to stay on course when no major changes are expected. They are limited in helping the project manager or developers control the project when unexpected change occurs.

How Much Reuse?

- Reusing architectures, design patterns, and off-the-shelf components can significantly reduce the development effort required to deliver a system. This assumes, however, that finding the appropriate solution or component to reuse can be done efficiently and reliably, and that participants understand the reused elements sufficiently well to assess their impact on the rest of the system. Moreover, while reuse decreases development time, it does not decrease significantly the time needed to test the system and may increase the time to repair defects. For commercial components, reuse also introduces risks associated with the amount and length of support provided by the vendor.

How Much Reuse?_2

- In general, the project manager faces three different trade-offs:

- **Architecture reuse.** The project participants adopt a preexisting system design and adapt it to the system. As no actual software is reused, assessing an existing architecture is not more difficult than assessing an ad hoc architecture built from scratch. Given an architecture that fits the needs of the system, selecting an existing architecture is always more advantageous than recreating one from scratch. The challenge in this approach, however, is to select an architecture that will enable participants to meet the nonfunctional requirements of the system.
- **Design pattern reuse**. Does not involve existing software. However, design patterns provide solutions to partial design problems. Developers will reuse, adapt, and combine many different design patterns during object design. This can lead to an over-engineered system when design patterns are used only for the sake of using design patterns.

How Much Reuse?_3

- In particular, the project manager and the chief architect should ensure that the nonfunctional requirements and design goals are not hurt by the introduction of specific patterns.
- **Framework and component reuse**. Constrains the system design and may force developers to use specific design patterns. Moreover, components are usually black boxes provided by a vendor, and they cannot be modified. The project manager must carefully assess the risks associated with the reuse. In general, framework assessments are shared among several projects in the organization, enabling projects to reduce the overhead associated with selecting a framework.

For well-defined application domains and flexible client requirements, a component-based approach may be the most cost effective. For new systems, building from scratch may be the only solution.

How Much Modeling?

Modeling constructs an abstraction of the system focusing only on the relevant aspects and ignoring all other. Modeling enables developers to deal with complexity by reasoning with simpler concepts. As development goes on, models are often refined to include more and more details, and if one is not careful, the complexity of the models may become comparable to the complexity of the system being modeled.

Modeling makes implicit knowledge about the system, explicit and formalizes it so that a number of participants can share this knowledge without misunderstanding. Models can support three different types of activities:

How Much Modeling?_2

Models can support three different types of activities:

•**Design**. Models provide a representation that enables developers to reason about the system. The models also form the basis for coding, as CASE tools are able to convert models into source code templates.

•**Communication**. Models provide a common vocabulary that enables developers to discuss specific design details, even though the corresponding system has not yet been realized. Models can support a broad range of communication activities, from exchanging models on paper napkins during lunch to formal inspections in the meeting room.

•**Archive**. Models provide a compact representation for storing the design and its rationale for future reference. Successful systems have extended life cycles, requiring several generations of developers to become familiar with early decisions.

How Much Modeling?_3

A project manager can use modeling to support one or more of the activities above. In general, modeling presents several challenges to the manager:

•**Formalizing knowledge is expensive**. It takes time and effort from developers, and requires validation and consensus so that every participant agrees on the same meanings.

•**Models introduce redundancy**. If changes are made to the system, the models must be updated as well. If several models depict the same aspects of the system, all must be updated. If one model becomes out of sync, it loses its value.

•**Models become complex**. As the model complexity becomes similar to the complexity of the system, the benefit of having a model is reduced significantly

How Much Process?

- Modeling the software life cycle presents similar benefits and challenges as modeling the system. On the one hand, it enables participants to reason over a simplified representation of the process, devise and implement improvements, and, consequently, share this knowledge across projects with the rest of the organization. On the other hand, process knowledge is difficult to make explicit and to keep up to date.

- The trade-off faced by the project manager ranges from no modeling to a detailed and continuously refined model, depending on how many resources can be expended in this activity. The project manager selects a specific solution based on the project environment. A well-known application or solution domain leads to repeatable processes increasing the benefit of modeling the process and attempting to share it across projects.

How Much Control and Monitoring?

- A detailed software life cycle model can lead to a detailed plan that can then be checked against actual progress. On one end of the spectrum, a plan could be so detailed as to plan and monitor the progress of each participant on each day. At the other end of the spectrum, a plan could include a single milestone indicating the delivery, leaving participants to plan their own work accordingly. Although both approaches are unreasonable, the project manager must choose a point somewhere in between that fits the project environment.

- In general, both planning and assessing the actual progress requires experience. Consequently, novice participants are poor judges of their progress. Similarly, the lack of precedents make it difficult to plan innovative systems. To deal with novices, a project manager can select a hierarchical reporting organization with more experienced participants in the middle levels.

How Much Control and Monitoring?_2

- For a project with a mixture of experienced and novice participants, the project manager can select peer-reviews, walkthroughs, and peer programming techniques to facilitate the transfer of knowledge from experts to novices and to create a web of informal communication channels.

When to Redefine Project Goals?

- Sometimes a project goal is too ambitious or the original goal is simply wrong. When this situation occurs, it is best to admit the failure and do a thorough analysis to understand what went wrong. It requires a certain psychological capability from the manager to admit that the goal cannot be achieved. After all, from the beginning of the project to the point of failure, the main job of the manager was to convince everybody that the project was doable.

- One way to deal with this is to declare the outcome of the project as a successful failure. Instead of focusing on the failure, the manager focuses on an analysis of the project and determines lessons learned.

- Many lessons cannot be learned in a successful project. The insights resulting from this analysis can often be used to improve the next project, or to recognize dangerous obstacles in the future that are suddenly now so clearly visible.

A Spectrum of Methodologies

A software engineering methodology is a coherent set of principles for addressing the issues described in the previous courses.

- Different styles of software development consist of the combinations of different project organizations and software life cycles. Traditionally, software engineering methodologies have evolved out of large complex, one-time projects, leading to an extensive planning phase, detailed modeling (i.e., documentation), a hierarchical organization, and fine grained planning. This requires high overhead that is not always justified for shorter or routine projects.

- As a reaction to these heavier methodologies, agile methodologies, such as Extreme Programming, Feature-Driven Design and Scrum.

A Spectrum of Methodologies_2

- In these more agile methodologies, planning is accomplished incrementally, the source code is treated as the only model of interest, and loose teams of experienced programmers are given the freedom to plan and organize their own work.

- Another advantage of agile methodologies is that they can adapt to fundamental changes in requirements.

- A well-accepted definition of agility is “the ability to both create and respond to change in order to profit in a turbulent business environment”.

“Processes“ in the Unified Process

The term **Process** is overloaded in the Unified Process

- Micro process: Policies & practices for building an artifact
  - Focus: Intermediate baselines with adequate quality and functionality as economically and rapidly as practical
  - Same as “Process” in the IEEE 1074 Standard
- Macro process: A set of micro processes and the dependencies among them
  - Focus: Production of a software system within cost, schedule and quality constraints
  - Also called: Life cycle model
- Meta process
  - Focus: Organizational improvement, long-term strategies, and return on investment (ROI)
  - Also called: Business process.

The Unified Process

- The Unified Process supports the following
  1. Evolution of project plans, requirements and software architecture with well-defined synchronization points
  2. Risk management
  3. Evolution of system capabilities through demonstrations of increasing functionality

It emphasizes the difference between *engineering* and *production.*

Difference: Engineering vs. Production

- Engineering Stage
  - Driven by less predictable but smaller teams, focusing on design and synthesis activities
- Production Stage
  - Driven by more predictable but larger teams, focusing on construction, test and deployment activities

Focus	**Engineering Stage Emphasis**	**Production Stage Emphasis**
Risk	Technical feasibility, Schedule	Cost
Artifacts	Planning, Requirements, System Design Documents	Baselines, Releases
Activities	Planning, Analysis, Design	Implementation, Integration
Quality Assessment	Demonstration, Inspection	Testing

Phases in the Unified Process

- The two stages of the Unified Process are decomposed into four distinct phases
- Engineering stage
  - Inception phase
  - Elaboration phase
- Production phase
  - Construction phase
  - Transition phase.

Transitioning from Engineering to Production

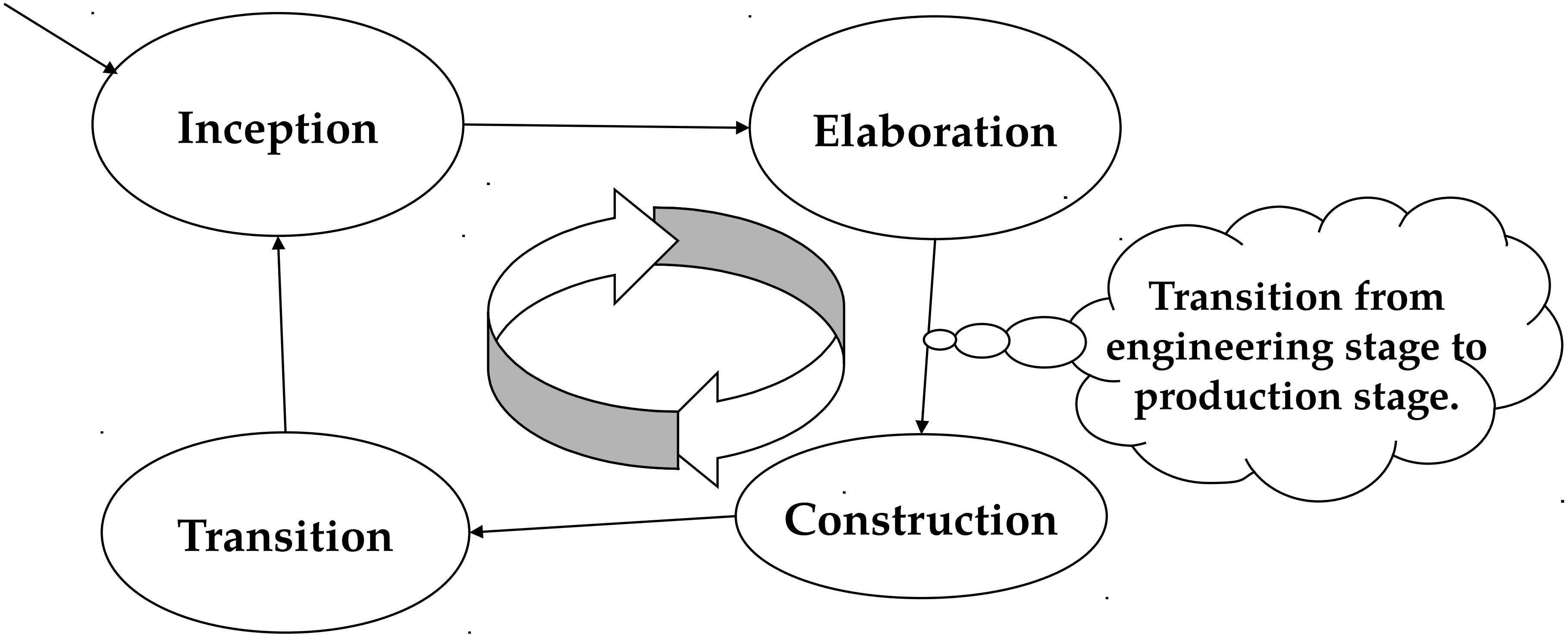
When the “engineering” of the system is complete, a decision must be made:

- Commit to production phase?
- Move to an operation with higher cost risk and inertia (i.e. bureaucracy)

Main questions:

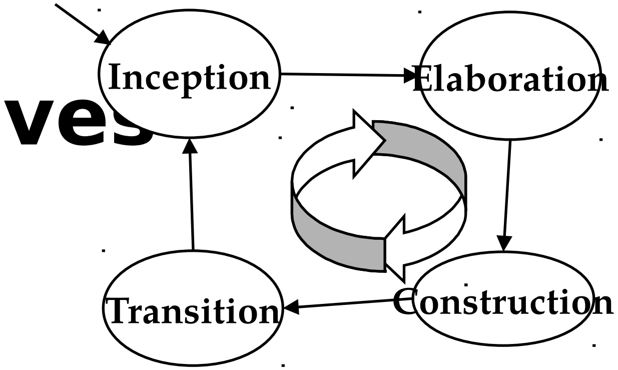
- Are the system models and project plans stable enough?
- Have the risks been dealt with?
- Can we predict cost and schedule for the completion of the development for an acceptable range?

States of a Software System in the UP



Transition from engineering stage to production stage.

Inception Phase: Objectives



- Establish the project scope
- Identify the critical use cases and scenarios
- Define acceptance criteria
- Demonstrate at least one candidate software architecture
- Estimate the cost and schedule for the project
- Define and estimate potential risks.

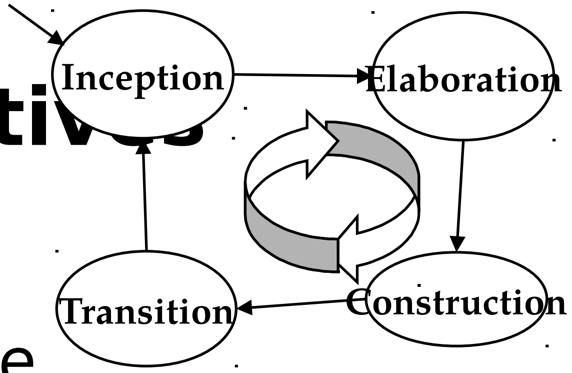
Inception Phase: Activities

- Formulate the scope of the project
  - Capture requirements
  - Result: problem space and acceptance criteria are defined
- Design the software architecture
  - Evaluate design trade-offs, investigate solution space
  - Result: Feasibility of at least one candidate architecture is explored, initial set of build vs. buy decisions
- Plan and prepare a business case
  - Evaluate alternatives for risks, staffing problems, plans.

Inception Phase: Evaluation Criteria

- Do all stakeholders concur on the scope definition and cost and schedule estimates?
- Are the requirements understood, are the critical use cases adequately modeled?
- Is the software architecture understood?
- Are cost, schedule estimates, priorities, risks and development processes credible?
- Is there a prototype that helps in evaluating the criteria?

Elaboration Phase: Objectives



- Baseline the software architecture
    - Establish a configuration management plan in which all changes are tracked and maintained
- Baseline the problem statement
- Base line the software project management plan for the construction phase
- Demonstrate that the architecture supports the requirements at a reasonable cost in a reasonable time

Question: Why does the Unified process not recommend the establishment of a configuration management plan during the inception phase?

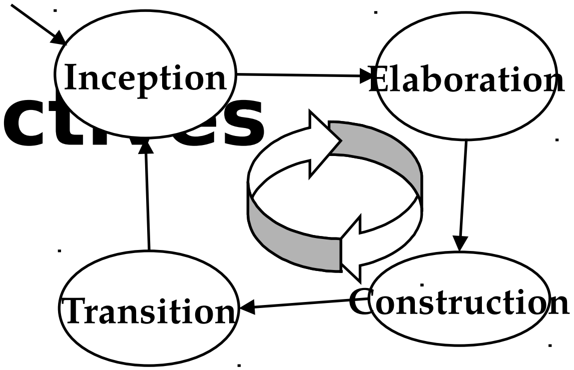
Elaboration Phase: Activities

- Elaborate the problem statement (“vision”) by working out the critical use cases that drive technical and managerial decisions.
- Elaborate the infrastructure.
- Tailor the software process for the construction stage, identify tools.
- Establish intermediate milestones and evaluation criteria for these milestones.
- Identify buy/build (“make/buy”) problems and make decisions.
- Identify lessons learned from the inception phase to redesign the software architecture if necessary (“always necessary”:-)

Elaboration Phase: Evaluation Criteria

- Apply the following questions to the results of the inception phase:
  - Is the problem statement stable?
  - Is the architecture stable?
  - Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
  - Is the construction plan credible? By what claims is it backed up?
  - Do all stakeholders (project participants) agree that the vision expressed in the problem can be met if the current plan is executed?
  - Are actual resource expenditures versus planned expenditures so far acceptable?

Construction Phase: Objectives



- Minimize development costs by optimizing resources
- Achieve adequate quality as rapidly as practical
- Achieve useful version (alpha, beta, and other test releases) as soon as possible

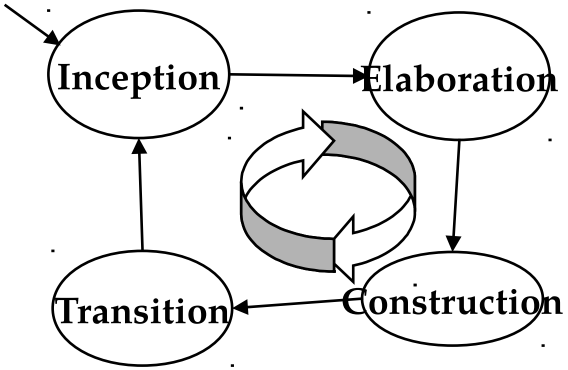
Construction Phase: Activities

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

Construction Phase: Evaluation Criteria

- Apply the following questions to the results of the construction phase:
  - Is the product baseline *mature* enough to be deployed in the user community?
    - Existing faults are not obstacles to do the release
  - Is the product baseline *stable* enough to be deployed in the user community?
    - Pending changes are not obstacles to do the release
  - Are the stakeholders ready for the transition of the software system to the user community?
  - Are actual resource expenditures versus planned expenditures so far acceptable?

Transition Phase



- The transition phase is entered when a baseline is mature
  - A usable subset of the system has been built with acceptable quality levels and user documents
  - It can be deployed to the user community
- For some projects the transition phase means the starting point for another version of the software system
- For other projects the transition phase means the complete delivery of the software system to a third party responsible for operation, maintenance and enhancement.

Transition Phase: Objectives

- Achieve independence of user (users can support themselves)
- Deployment baseline is complete and consistent with the criteria in the project agreement
- The final baseline can be built as rapidly and cost-effectively as possible.

Transition Phase: Activities

- Synchronization and integration of concurrent development increments into one consistent deployment baseline
- Commercial packaging and production
- Sales rollout kit development
- Field personnel training
- Test of deployment baseline against the acceptance criteria.

Transition Phase: Evaluation Criteria

- Is the user satisfied?
- Are actual resource expenditures versus planned expenditures so far acceptable?

Iterations in the Unified Process

- Each of the four phases introduced so far (inception, elaboration, construction, transition) consists of one or more iterations
- An iteration represents a set of activities for which there is a milestone (“well-defined intermediate event”)
  - The scope and results of the iteration are captured via work products (called artifacts in the UP).

Phase vs. Iteration

- *A phase* creates a formal, stake-holder approved version of artifacts
  - It leads to a “major milestone”
  - Phase to phase transition:
    - triggered by a significant business decision (not by the completion of a software development activity)
- An *iteration* creates an informal, internally controlled version of artifacts
  - It leads to a “minor milestone”
  - Iteration to iteration transition:
    - Triggered by a specific software development activity.

Artifact Sets in the Unified Process

- Artifact: A work product in a uniform representation format (natural language, UML, Java, binary code,…)
- Artifact set:
  - A set of artifacts developed and reviewed as a single entity
- The Unified Process distinguishes five artifact sets
  - Management set
  - Requirements set
  - Design set
  - Implementation set
  - Deployment set

Also called the engineering set.

Artifact Sets in the Unified Process

Engineering Set

Requirements Set	Design Set	Implementation Set	Deployment Set
1. Vision document	1. Design model(s)	1. Source code baselines	1. Integrated product executable
2. Requirements model(s)	2. Test model	2. Compile-time files	2. Run-time files
	3. Software architecture	3. Component executables	3. User documentation

Management Set

Planning Artifacts

1 Software Project Management Plan (SPMP)
2. Software Configuration Management Plan (SCMP)
3. Work breakdown structure
4. Business Case
5. Release specifications

Operational Artifacts

1. Release descriptions
2. Status assessments
3. Change Management database
4. Deployment documents
5. Environment.

Representation of Artifact Sets (1)

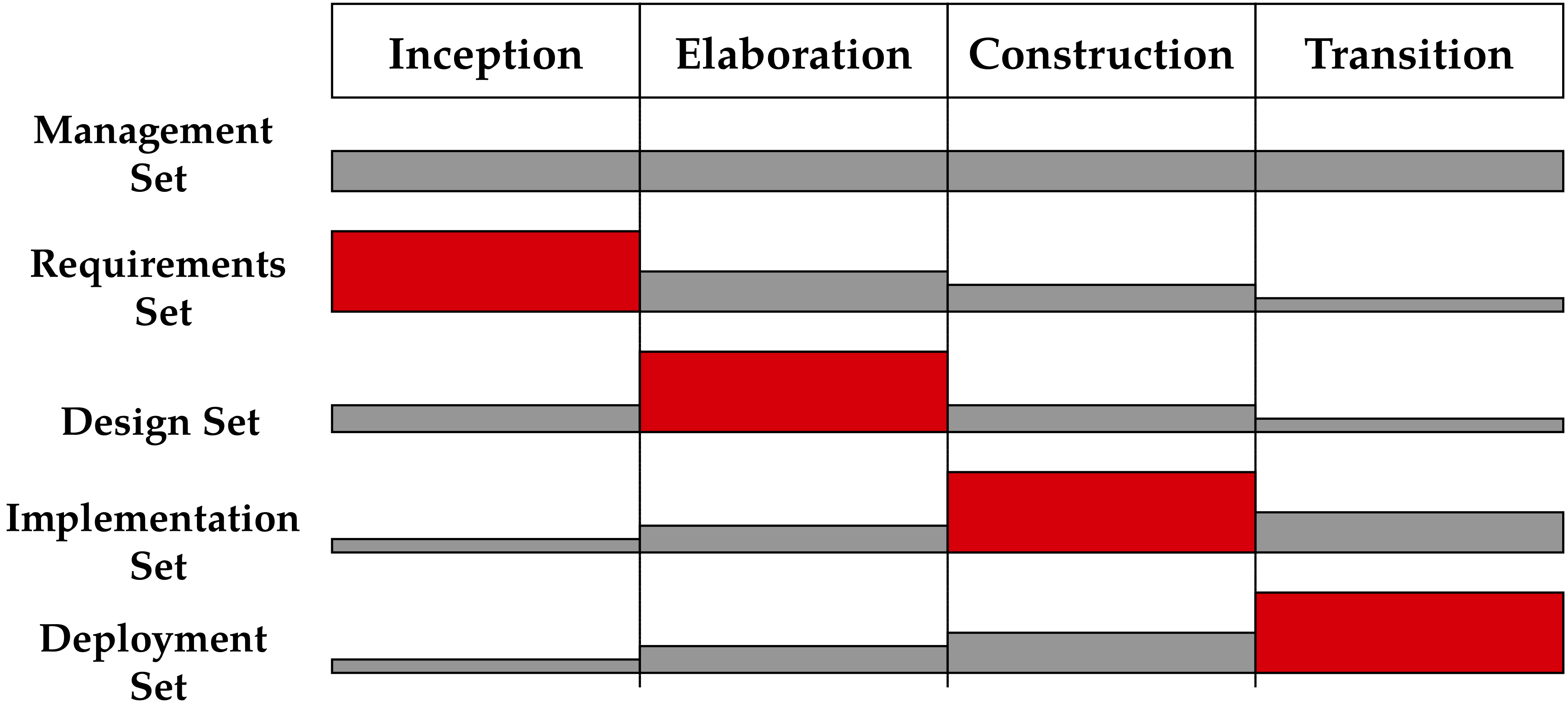
- Management Set
  - Goal: Capture plans, processes, objectives, acceptance criteria
  - Notation: Ad hoc text, graphics, textual use cases.
- Requirements set
  - Goal: Capture problem in language of problem domain
  - Notation: Structured text, UML models
- Design set
  - Goal: Capture the engineering blueprints
  - Notation: Structured text, UML models.

Rationale for Selection of Artifact Sets (2)

- Implementation set
  - Goal: Capture the building blocks of the solution domain in human-readable format
  - Notation: Programming language
- Deployment set
  - Goal: Capture the solution in machine-readable format
  - Notation: Machine language.

Life-cycle Focus on Artifact Sets

- Each artifact set is the predominant focus in one stage of the unified process.



Inception

Elaboration

Construction

Transition

Management Set

Requirements Set

Design Set

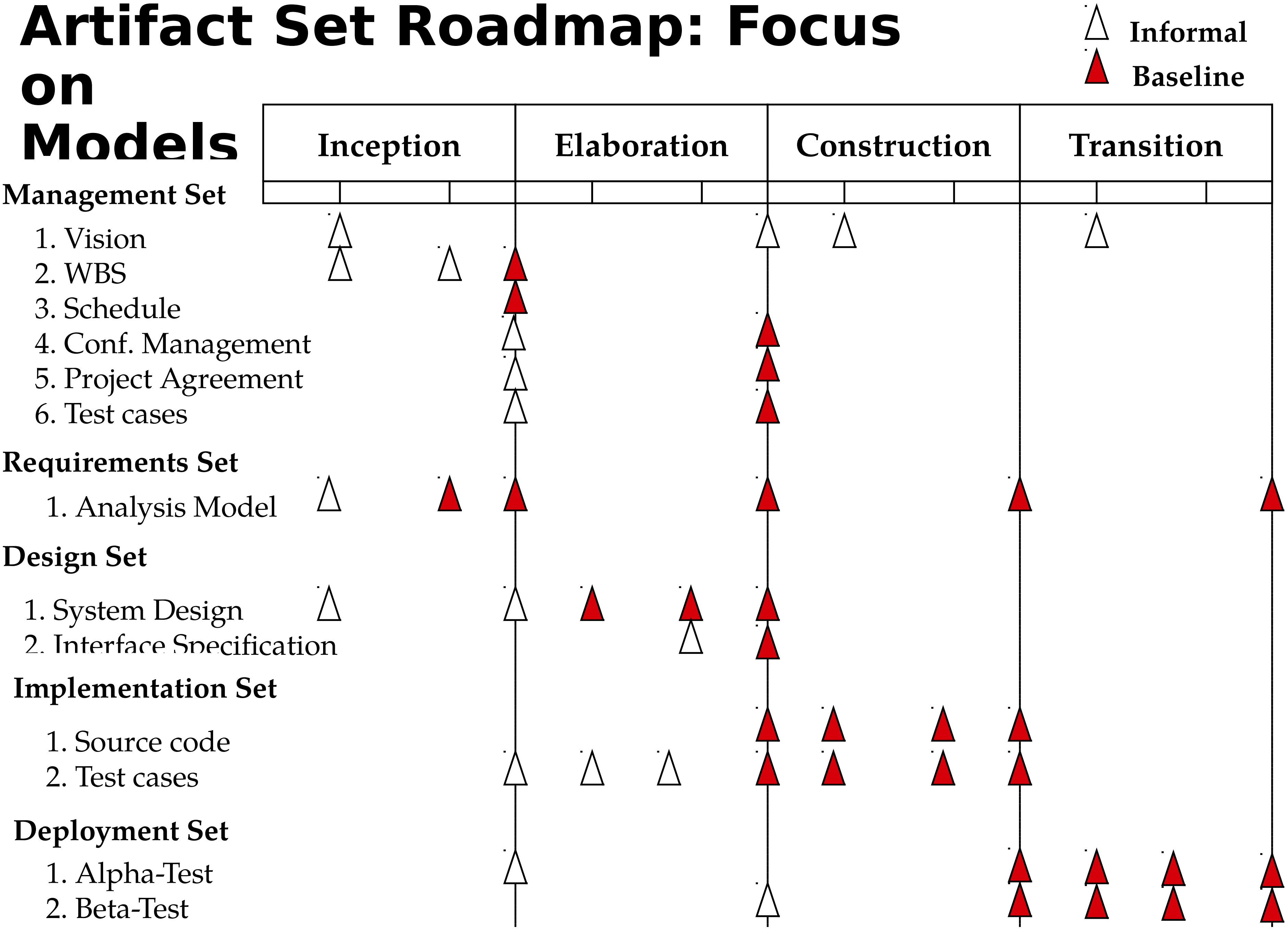
Implementation Set

Deployment Set

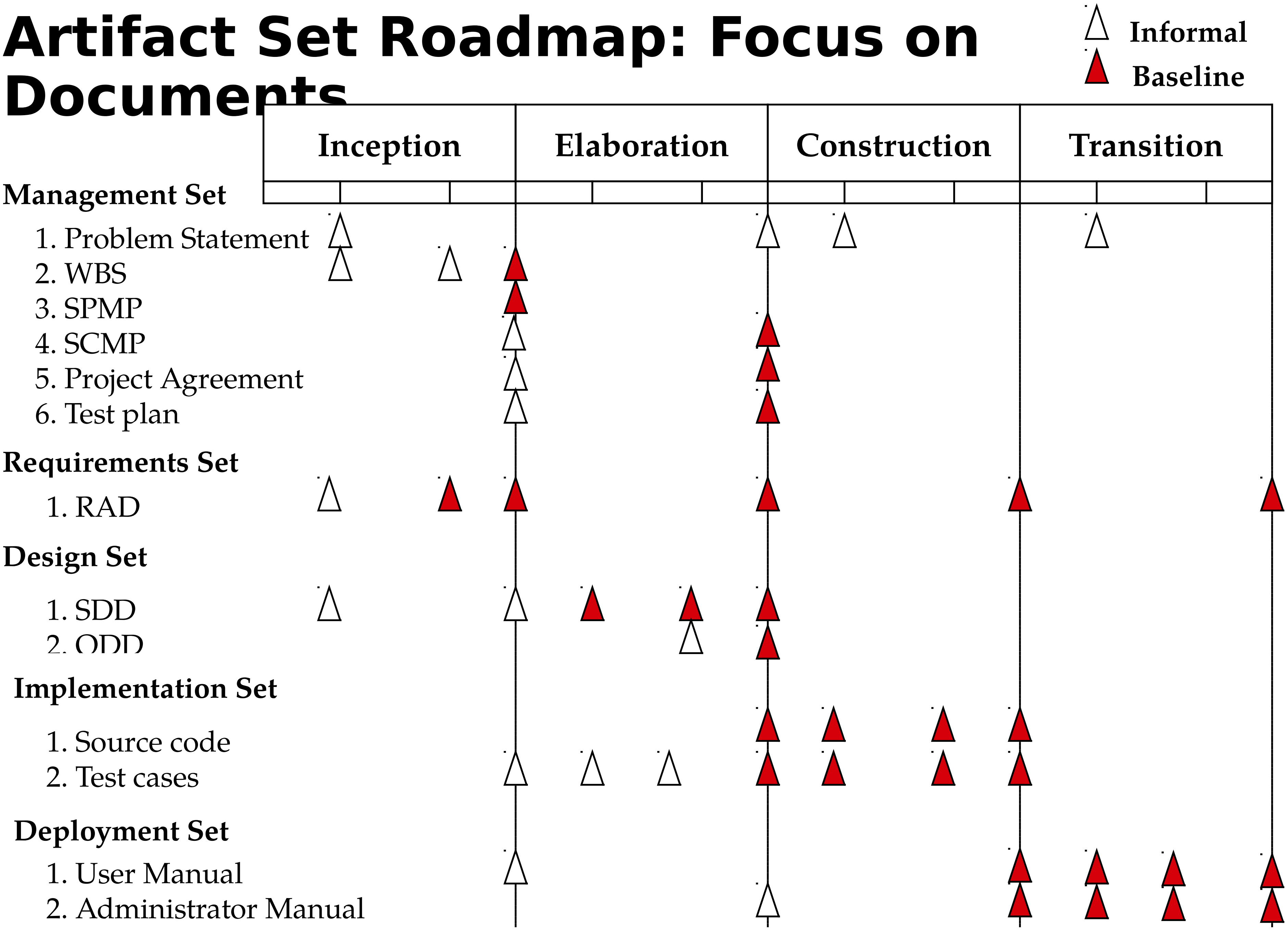
Managing the Artifact Sets

- Some artifacts need to be updated at each major milestone (after a phase)
- Other artifacts must be updated at each minor milestone (after an iteration)
- Artifact set roadmap
  - Visualization of the updates of artifacts across the software life-cycle
- The software project manager is responsible for managing the artifact set roadmap
  - Artifact set roadmap: Focus on models
  - Artifact set roadmap: Focus on documents.

Artifact Set Roadmap: Focus on Models



Artifact Set Roadmap: Focus on Documents



△ Informal

🔺 Baseline

Inception | Elaboration | Construction | Transition

Management Set

1. Problem Statement
2. WBS
3. SPMP
4. SCMP
5. Project Agreement
6. Test plan

Requirements Set

1. RAD

Design Set

1. SDD
2. ODD

Implementation Set

1. Source code
2. Test cases

Deployment Set

1. User Manual
2. Administrator Manual

Models vs. Documents

- Many software project managers pay too much attention on the production of documents
- Documentation-driven approach
  - The production of the documents drives the milestones and deadlines
- Model-driven approach
  - The production of the models drive the milestones deadlines
- Main goal of a software development project:
  - Creation of models and construction of the software system
- The purpose of documentation is to support this goal.

Historical Reasons for Documentation-Driven Approach

- People wanted to review information, but did not understand the language of the artifact
- People wanted to review information, but did not have access to the tools to view the information
- No rigorous engineering methods and languages were available for analysis and design models
  - Therefore paper documents with ad hoc text were used
- Conventional languages for implementation and deployment were highly cryptic
  - A more human-readable format was needed
- Managers needed “status”
  - Documents seemed to be a good mechanism for demonstrating progress.

Artifact-Driven Approach

- Provide templates for documents at the start of the project
- Instantiate documents automatically from these templates
  - Enrich them with modeling and artifact information generated during the project
- Tools automatically generate documents from the models. Examples:
  - Generation of analysis and design documents (Commercial CASE tools)
  - Generation of the interface specification (Javadoc)
  - Test case generation (J_Unit)
  - Schedule generation (Microsoft Project).

Micro Processes in the Unified Process

- The Unified Process distinguishes between macro and micro process:
  - The macro process models the software lifecycle
  - The micro process models activities that produce artifacts

- The micro processes are also called workflows in the Unified Process.

Workflows in the Unified Process

- Management workflow
- Environment workflow
- Requirements workflow
- Design workflow
- Implementation workflow
- Assessment workflow
- Deployment workflow.

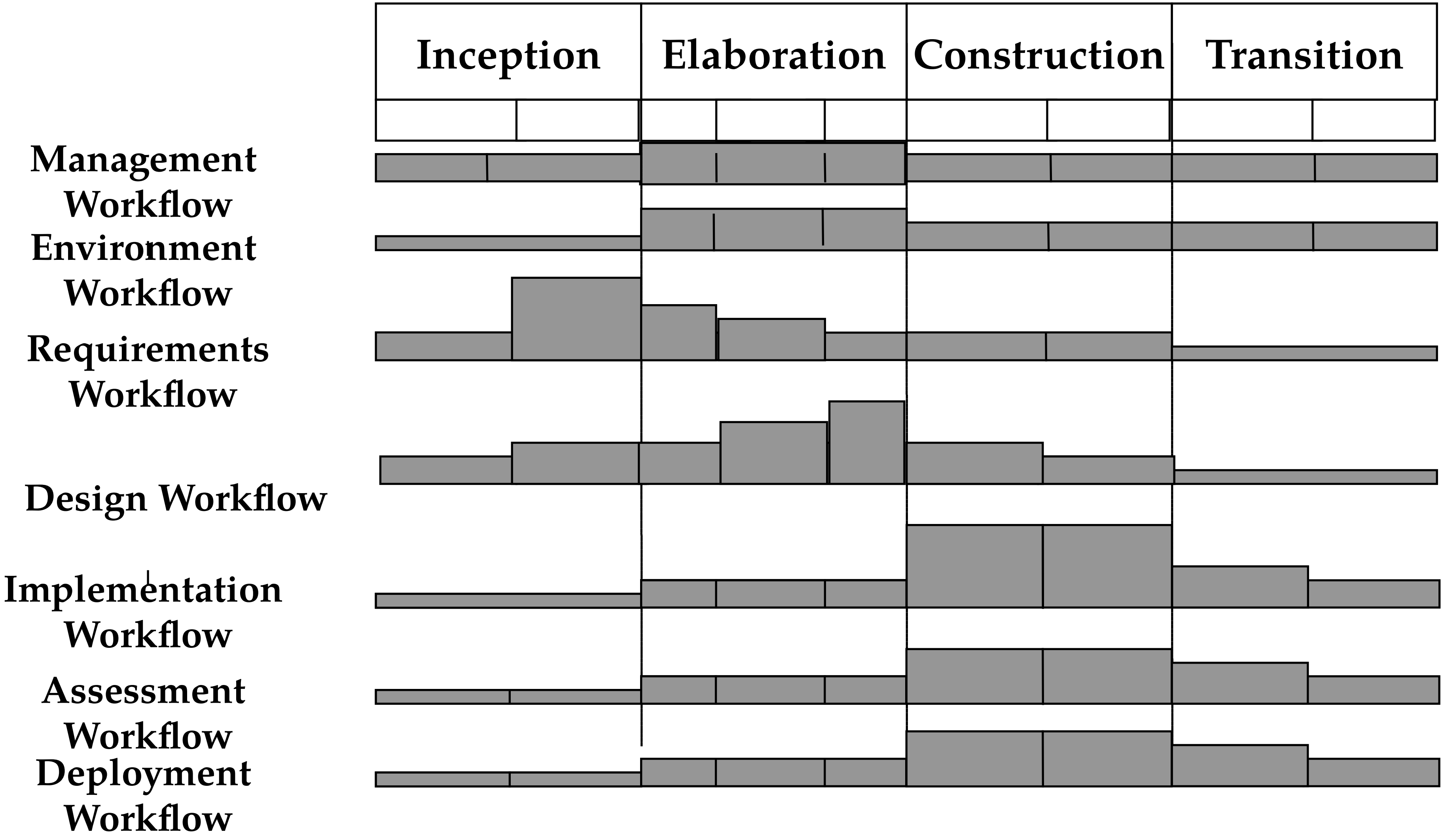
Workflows in the Unified Process

- Management workflow
  - Planning the project (Problem statement, SPMP, SCMP, test plan)
- Environment workflow
  - Automation of process and maintenance environment. Setup of infrastructure (Communication, configuration management, ...)
- Requirements workflow
  - Analysis of application domain and creation of requirements artifacts (analysis model)
- Design workflow
  - Creation of solution and design artifacts (system design model, object design model).

Workflows in the Unified Process (2)

- Implementation workflow
  - Implementation of solution, source code testing, maintenance of implementation and deployment artifacts (source code)
- Assessment workflow
  - Assess process and products (reviews, walkthroughs, inspections, testing…)
- Deployment workflow
  - Transition the software system to the end user.

Workflows work across Phases



Inception

Elaboration

Construction

Transition

Management Workflow

Environment Workflow

Requirements Workflow

Design Workflow

Implementation Workflow

Assessment Workflow

Deployment Workflow

- Workflows create artifacts (documents, models)
- Workflows consist of one or more iterations per phase.

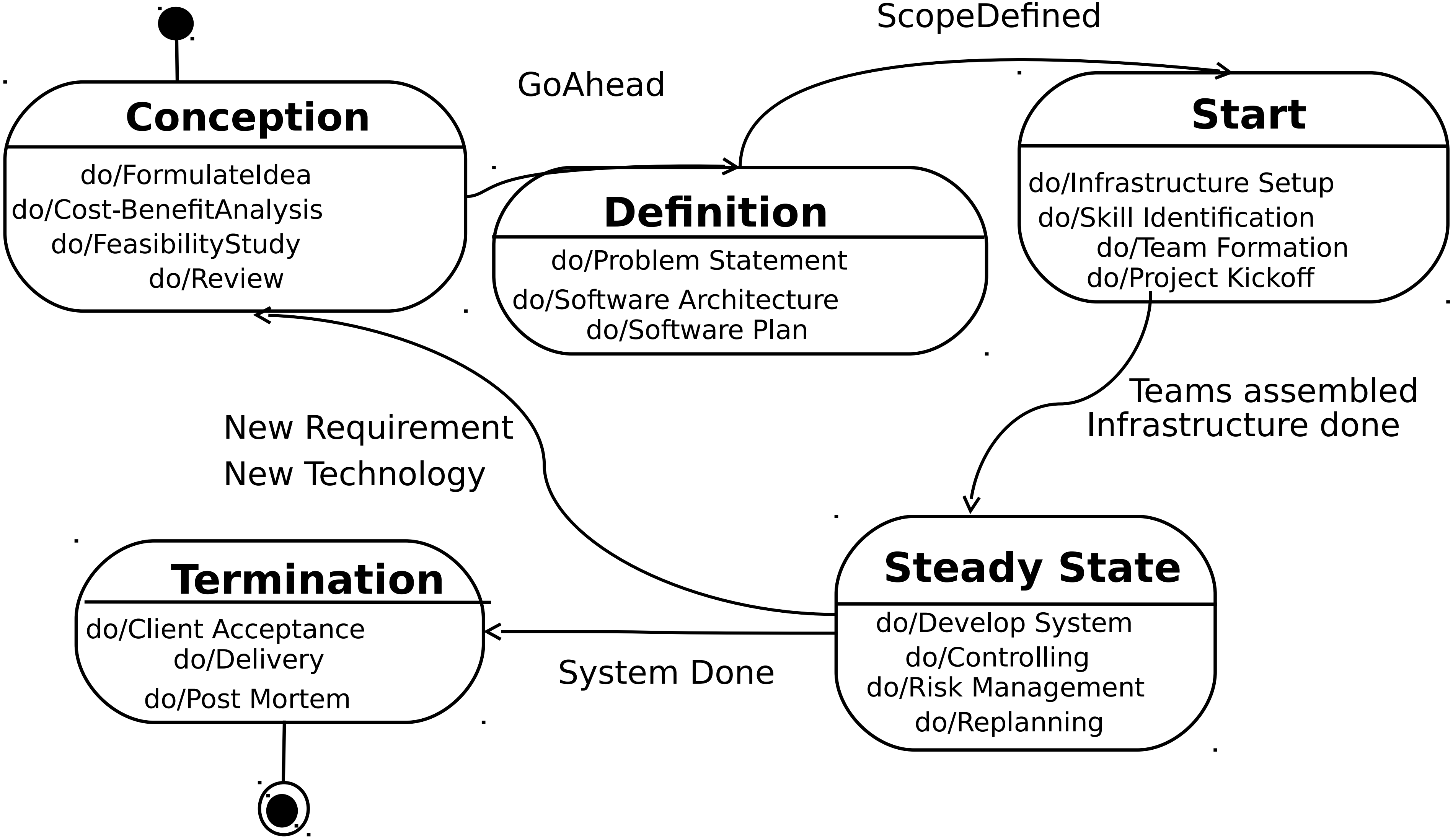
Managing Projects in the Unified Process

- How should we manage the construction of software systems with the Unified Process?
- Approach
  - Treat the development of a software system with the Unified Process as a set of several iterations
    - Some of these can can be scheduled in parallel, others have to occur in sequence
  - Define a single project for each iteration
  - Establish work break down structures for each of the 7 workflows.

Project Phases vs. Unified Process Phases

- Every project has at least 5 states
  - Conceiving: The idea is born
  - Defining: A plan is developed
  - Starting: Teams are formed
  - Performing: The work is being done
  - Closing: The project is finished.

Phases of a Software Project



ScopeDefined

GoAhead

Conception

do/FormulateIdea
do/Cost-BenefitAnalysis
do/FeasibilityStudy
do/Review

Definition

do/Problem Statement
do/Software Architecture
do/Software Plan

Start

do/Infrastructure Setup
do/Skill Identification
do/Team Formation
do/Project Kickoff

Teams assembled
Infrastructure done

New Requirement

New Technology

Termination

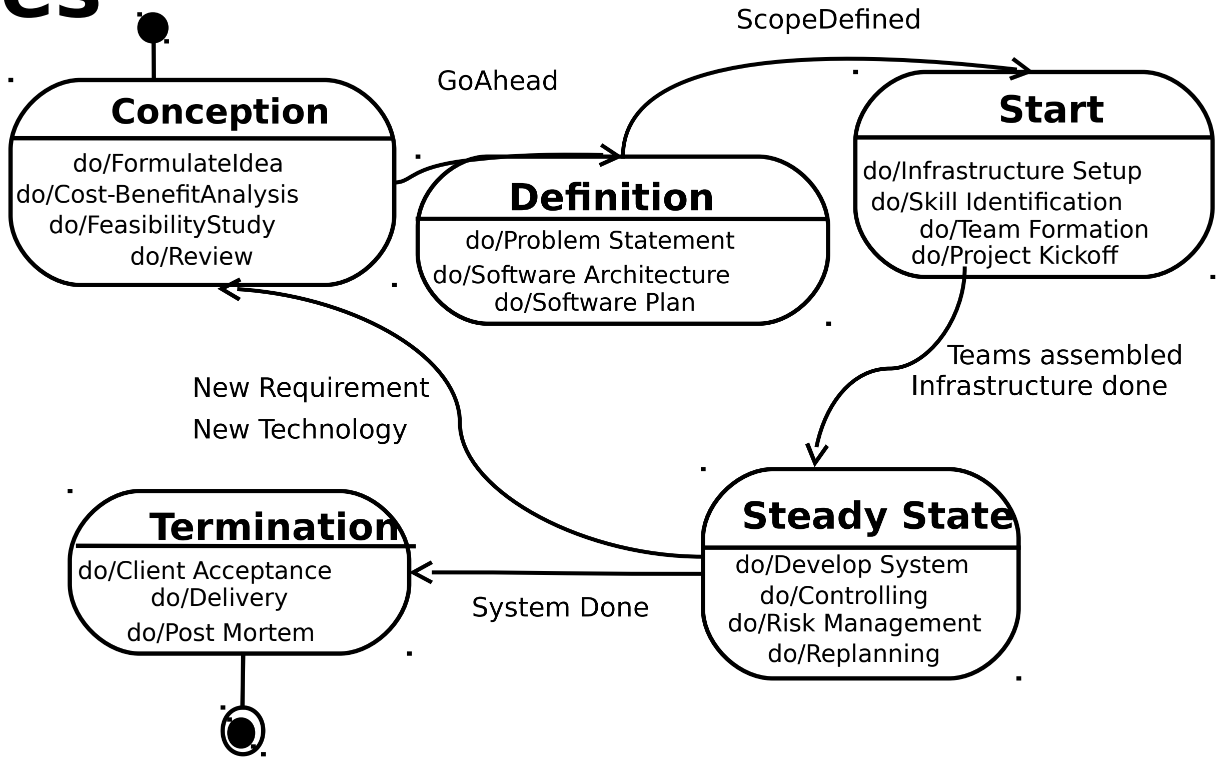
do/Client Acceptance
do/Delivery
do/Post Mortem

System Done

Steady State

do/Develop System
do/Controlling
do/Risk Management
do/Replanning

Project Phases vs. Unified Process Phases

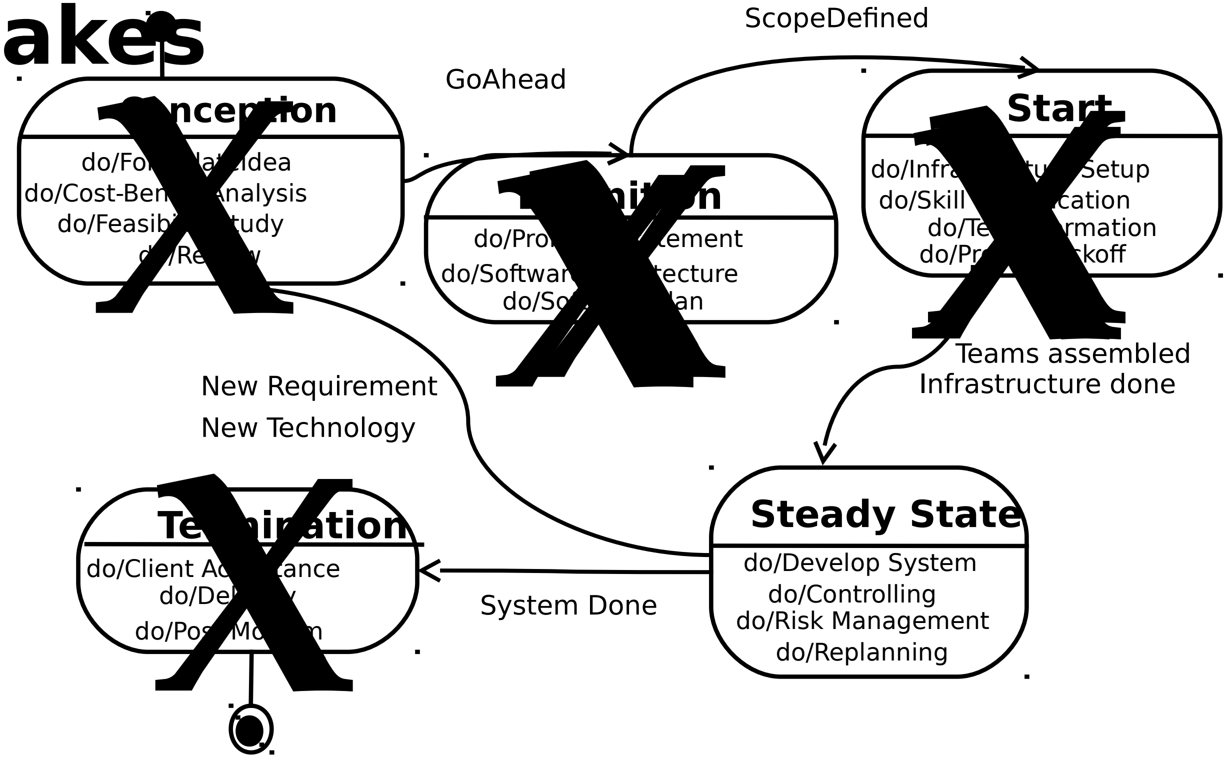


Each iteration in the unified process phases

Inception, Elaboration, Construction, Transition

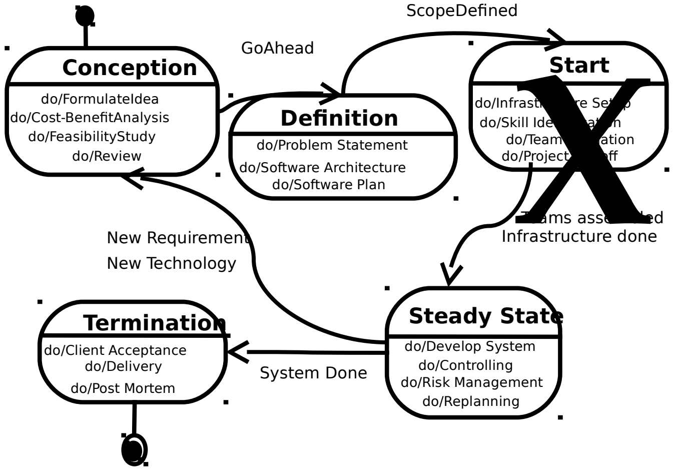
should go through each of these 5 project phases!

Unified Process Management Mistakes



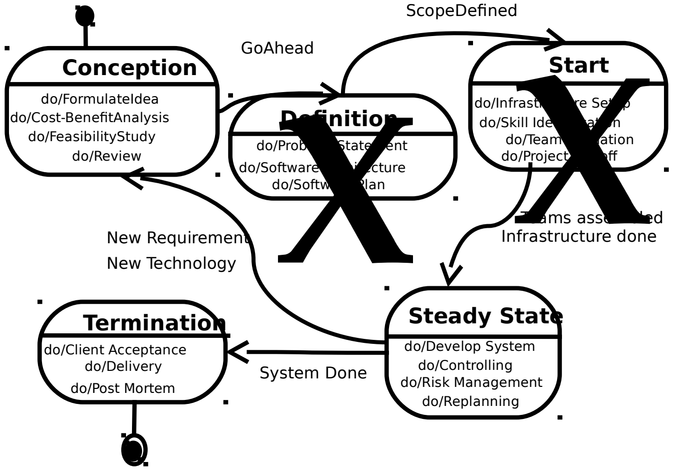
- Project manager skips the start phase
- Project manager skips the definition and start phase
- Project manager jumps straight to the steady state phase after joining the project late
- Project manager cancels the termination phase.

Mistake: Skipping the Start Phase



- Main reason: Time pressure
- Reasons for start phase
  - Inform stakeholders that the project has been approved and when work will start
  - Confirm that stakeholders are able to support the project
  - Reevaluate and reconfirm work packages with developers
  - Explain your role as manager to stakeholders and developers.

Mistake: Skipping Definition and Start Phase



- Known territory argument
  - “I have done this before, no need to waste time”
    - Even though a project may be similar to an earlier one, some things are always different
- Unknown territory argument
  - “My project is different from anything I have ever done before, so what good is it to plan?”
    - It is better to create a map if you are attempting to travel into unknown territory.

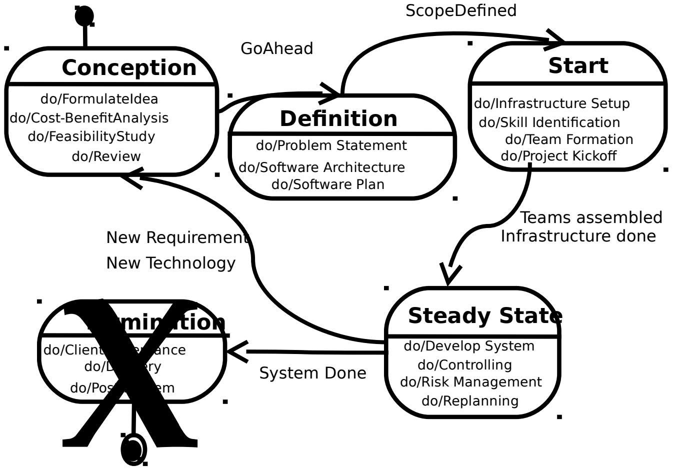
Problem: Joining a Project Late

Joining a project late is not that uncommon

- Often the planning has been performed by another person, usually a high level manager, and you are asked to take the project over
- Or the project is in such a bad state, that the current project manager needs to be replaced

- Reason to jump right into steady state phase
  - "The plan has already been developed, so why should I go back to the conception and definition phases?"
- Reasons to reevaluate the conception and definition phase:
  1. See if you can identify any issues that may have been overlooked
  2. Try to understand the rationale behind the plan and to decide if you feel the plan is achievable.

Mistake: No Termination Phase



- Reasons for skipping or not completing the termination phase:
  - You leave a project to move on right to the next one. (Because you are a successful manager:-)
  - Scarce resources and short deadlines
  - A new project is always more challenging than wrapping up an old one
- Take the time to ensure that all tasks are completed or identified as open issues:
  - Otherwise you never really know how successful your project was
- Try to learn from your mistakes ("lessons learned"):
  - If you don't, you will make the the same mistakes again, and may even fail.

Summary

- Unified Process: Iterative software lifecycle model
  - Emphasis on early construction of a software architecture
  - Emphasis on early demonstrations of the system
- Definitions
  - Phase: Status of the software system.
    - 4 phases: Inception, Elaboration, Construction, Transition
  - Workflow: Mostly sequential activity that produces artifacts
    - 7 workflows: Management, environment, requirements, design, implementation, assessment, deployment.
    - 5 artifact sets: Management set, requirements set, design set, implementation set, deployment set
  - Iteration: Repetition within a workflow.
- Each unified process iteration is a software project.

Additional References

- Walker Royce
  - Software Project Management, Addison-Wesley, 1998.
- Ivar Jacobsen, Grady Booch & James Rumbaugh
  - The Unified Software Development Process, Addison Wesley, 1999.
- Jim Arlow and Ila Neustadt
  - UML and the Unified Process: Practical Object-Oriented Analysis and Design, Addison Wesley, 2002.
- Philippe Kruchten
  - Rational Unified Process, Addison-Wesley, 2000.

Additional Slides

Component Based Software Development

- Buy
  - Commercial of the shelf components (COTS), reusable objects, …
- Build
  - Custom development, build everything from scratch,…
- Comparision: Buy vs. Build

Commercial Components (“Buy”)

- ☺ Predictable license costs
- ☺ Broadly used, mature technology
- ☺ Available now
- ☺ Dedicated support organization
- ☺ Hardware/software independence (sometimes)
- ☺ Rich in functionality

- ☹ Frequent upgrades
- ☹ Up-front license fees
- ☹ Recurring maintenance fees
- ☹ Dependency on vendor
- ☹ Run-time efficiency sacrifices
- ☹ Functionality constraints
- ☹ Integration not always trivial
- ☹ No control over upgrades and maintenance
- ☹ Unnecessary features that consume extra resources
- ☹ Often inadequate reliability and stability
- ☹ Multiple-vendor incompatibilities.

Custom Components (“Build”)

- ☺ Complete change freedom
- ☺ Smaller, often simpler implementations
- ☺ Often better performance
- ☺ Control of development and enhancement

- ☹ Expensive, unpredictable development
- ☹ Unpredictable availability date
- ☹ Undefined maintenance model
- ☹ Often immature and fragile
- ☹ Single-platform dependency
- ☹ Drain on expert resources.

Model of the Unified Process (Analysis)

- Inputs:
  - Problem Statement
- Functional Requirements:
  - Top level use case: Develop software system that implements the problem statement.
- Outputs:
  - Requirements analysis document
  - Software project management plan
  - Software configuration management plan
  - System design document
  - Object design document
  - Test plan and test cases
  - Source code
  - User manual and administrator manual

Model of the Unified Process: System Design

- Design Goals:
  - High performance, dependability, low cost, maintainability, usability
- Subsystems:
  - The workflows Management, Environment, Requirements, Design, Implementation, Assessment, Deployment
- Hardware/Software mapping:
  - Each subsystem is running on its own node.
- Concurrency:
  - The threads can run concurrently.
- Global control flow:
  - Event-driven. The subsystems communicate via events. Typical events are: „Requirement has changed“, „Review comments available“, „Time has expired“)

Model of the Unified Process: System Design (ctd)

- Persistent Data:
  - Vision, Process Model, Configuration Items, Analysis Model, System Design Model, Object Design Model, Communication data.
- Access control:
  - Stakeholders (End users, managers, customers, developers, …) have access to the persistent data with access rights defined dynamically by environment workflow.
- Boundary Conditions
  - Startup of workflows: All workflows start simultaneously
  - Steady state of workflows: Workflows wake up on an event, process the event, and go to sleep afterwards.
  - Terminal conditions of workflows: A risk has occurred that cannot be dealt with

Lifecycle Improvement

- There are 3 possibilities to improve a multi-step process
  - Quality improvement: We take an n-step process and improve the efficiency of each step
    - Example: TQM (Total Quality Management)
  - Overhead reduction: We take an n-step process and eliminate some of the steps
    - Example: Extreme Programming
  - Concurrency: We take an n-step process and parallelize some of the seps or use more concurrency in the resources being used
    - Example: Unified Process.