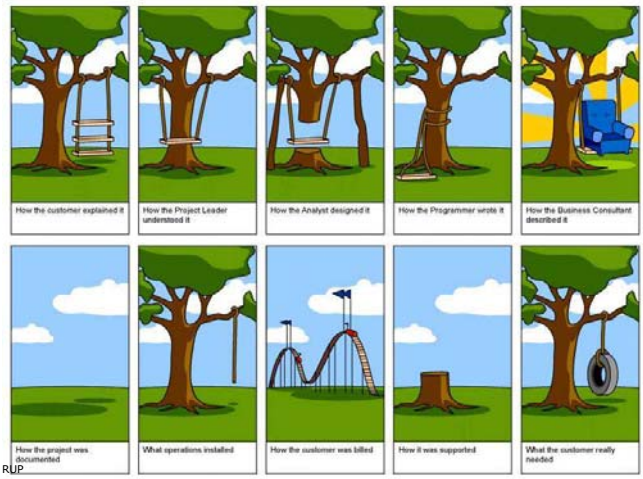
AGENDA

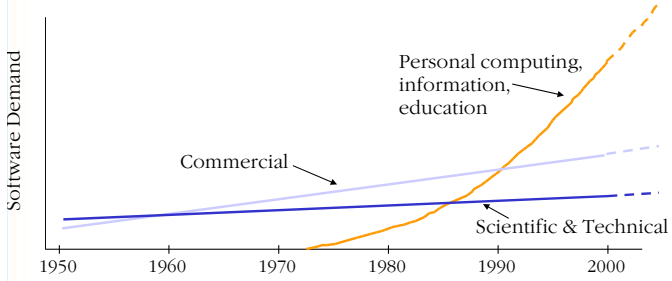
- Software Development Methodology
- Introduction into the RUP
- Iterative Development
- Basic Principles of RUP

MOTIVATION



SOFTWARE DEVELOPMENT METHODOLOGY

TRENDS — SOFTWARE DEMAND



- Several hundred billion dollars/year spent worldwide and growing
- Essential part of almost all organizations
- Key part of many products -> embedded systems
- Essential for designing many products

PROBLEMS — SOFTWARE DEVELOPMENT

1. Schedule and cost **estimates** often grossly **inaccurate**
   - Over schedule
   - Over budget
2. **Productivity** of software developers **has not kept pace** with demand for their services
3. **Quality** of software is sometimes **less than adequate**
   - Unreliable → Ariane 5 rocket
   - Unsafe → London Ambulance System; Therac-25
   - Inflexible → Hard to change/maintain
   - Abandoned → London Stock Exchange

WHY?

PROBLEMS — SOFTWARE DEVELOPMENT

- **Lack of historical data** on software development process (1)
- No good way to measure **productivity** (1, 2)
- **Poor communication** with customer/other developers (1, 3)
- **No systematic approach** to software development (1, 2, 3)
- **Vague/unclear specification** of customer requirements (1, 3)
- Lack of systematic and complete **software testing** (3)
- Little emphasis on **software maintenance** (3)

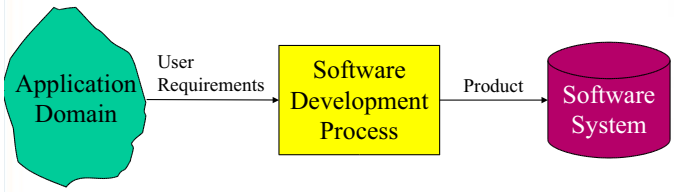
SOME PROBLEMS

- Wrong understanding user's needs
- Inability deal with changing requirements
- Modules uncompatibility
- Time-consuming support and expanding
- Lately project bug recognizing
- Low quality
- Unacceptable performance
- Unable find out, who, when and why makes change
- Unreliable „build-and-release" process
- …

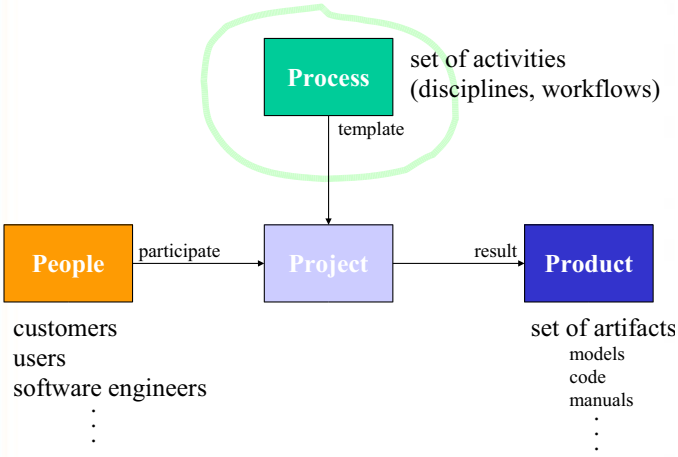
PROBLEM'S CAUSES

- Imperfect requirements management
- Unclear communication
- Fragile architecture
- Complexity
- Hidden inconsistency among requirements, desings and implementation
- Incommensurate testing
- Subjective judgement of project state
- Risks unpreceding
- Uncontroled change making
- Unsufficient automatization
- …

DEVELOPING SOFTWARE SYSTEMS



THE FOUR P's IN SOFTWARE DEVELOPMENT



PROCESS – GENERIC VIEW

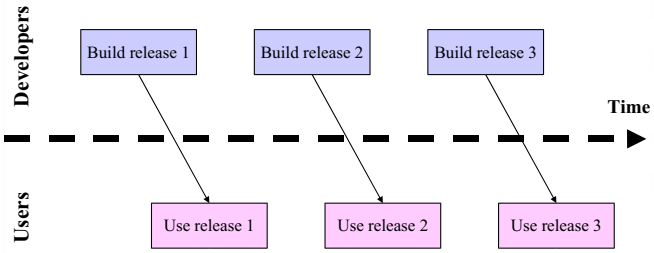
- Project definition – concentrates on WHAT
  - Project planning
  - Requirements capture
  - Analysis
- Software production – concentrates on HOW
  - Desing
  - Programming
  - Testing
  - Deployment
- Software operation – concentrates on CHANGES
- + documentation, change management, revision, ….

PROCESS – SYSTEMIC VIEW

- Activities — **Things we do**
  - Concrete guideline to perform activity
- Methodics (working process) — **Order in which we do things**
  - Order of perfoming activities
  - Created products
  - Quality management
  - Milestones for track the progress
- Tools (support) — **Things we use**
  - To do activites more effective

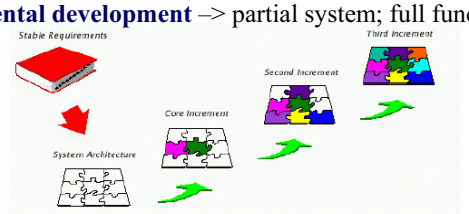
➡**Project Life Cycle**

PHASED PARADIGM

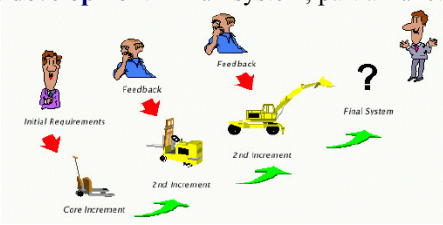


PHASED PARADIGM — INCREMENTS & ITERATIONS

Incremental development –> partial system; full functionality



Iterative development –> full system; partial functionality



PHASED PARADIGM — PROS

- **Early training and feedback**
  - Allows user training and provides real-use feedback ‘ on early releases of the system
- **Can create new markets**
  - For functionality never before offered
- **Frequent releases**
  - Allows bugs to be fixed globally and frequently
- **Apply appropriate expertise**
  - Development team can focus on different areas of expertise with each release

PROCESS — PRINCIPLES

- Rigor and formality
  - **Allows us to produce better quality software**
- Separation of concerns
  - **Allows us to divide responsibilities/work**
- Modularity
  - **Allows us to work on parts independently**
- Abstraction
  - **Allows us to concentrate on important aspects**
- Anticipation of change
  - **Allows us to prepare for maintenance**
- Generality
  - **Allows us to reuse software**
- Incremental
  - **Allows us to evolve to desired solution**

WHY IS PROCESS IMPORTANT?

- Allows divison of labour
  - Team members know their roles and tasks
- Communication support
  - Everyone knows his tasks
- Easier project management
  - Manger understands what is the matter
- Universal developers
  - Opportunity to use knowledge in others projects
- Productivity and quality of work
  - Software development process is repeatable

BEST PRACTICES FOR SOFTWARE DEVELOPMENT

BEST PRACTICES

- „Best practices for software development“

  Ideas, principles and methods for software development quality and efficiency improvement

- Develop iteratively
- Manage requirements
- Use component architectures
- Model visually
- Continuously verify quality
- Manage change

DEVELOP ITERATIVELY

- Mitigate project risks as soon as possible
- Initial design is likely to be flawed
- Lately discovery of serious design defects is highly costly
- Benefits
  - Risks are mitigated earlier, because elements are integrated progressively
  - Changing requirements and tactics are accommodated
  - Improving and refining the product is facilitated, resulting in a more robust product
  - Organizations can learn from this approach and improve their process
  - Reusability is increased

DEVELOP ITERATIVELY II

MANAGE REQUIREMENTS

- Requirement - "a condition or capability to which the system must conform“
- Requirements management is a systematic approach to finding, documenting, organizing, and tracking a system's changing requirements
  - Managed process
  - During the whole project lifecycle

USE COMPONENT ARCHITECTURES

- Make an agreement about the software system internal organization
  - Structural things and interfaces in the system
  - Specification of behavior and collaboration of these things
  - Composition into more complex things
  - Standards
  - Inner style
- Control the project complexity and integrity
- Team development
- Efficiency
- Ensure future maintainability

USE COMPONENT ARCHITECTURES II

- Flexible architecture
  - Fulfils the today‘s and tomorrow‘s requirements
  - Enhances extensibility
  - Allows reusibility
  - Encapsulates the system dependencies
- Component architerure
  - Reusability and enhancement of components
  - Using commercialy available components
  - Iterative and incremental modular development

MODEL VISUALLY

- Use semantically rich, graphical and textual design notations to capture software designs
  - Requirements, analysis, design
  - Communication
  - Lower the complexity
  - Raise the level of abstraction
- Various models, various views, various abstractions
- Examples:
  - Use Cases to unambiguously specify behavior
  - Class Diagrams and Data Model Diagrams to capture design
  - State Transition Diagrams and Sequence Diagrams to model dynamic behavior

MODEL VISUALLY II

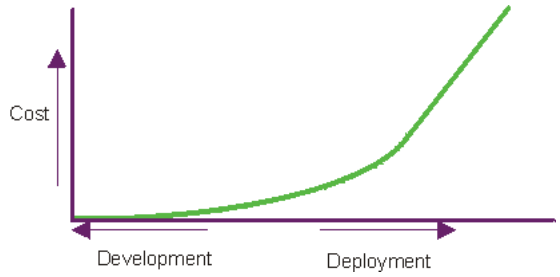
- UML is the standard
- UML is a language for software system
  - Visualization
  - Specification
  - Construction
  - Documentation

CONTINUOUSLY VERIFY QUALITY

- Quality - the capability of the product to be used
- Continuously verify quality
  - Discrepancies between specification, design and implementation
  - Functionality, performance, reliability
  - Verify the most risky areas
  - Automatization (tools)
- The responsibility of the whole team
- Managing quality means to:
  - Identify appropriate indicators (metrics) of acceptable quality
  - Identify appropriate measures to be used in evaluating and assessing quality
  - Identify and appropriately address issues affecting quality as early and effectively as possible

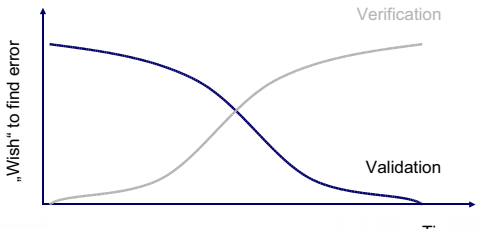
CONTINUOUSLY VERIFY QUALITY II

- Software problems are 100 to 1000 times more costly to find and repair after deployment. Verifying and managing quality throughout the project's lifecycle is essential to achieving the right objectives at the right time



CONTINUOUSLY VERIFY QUALITY III

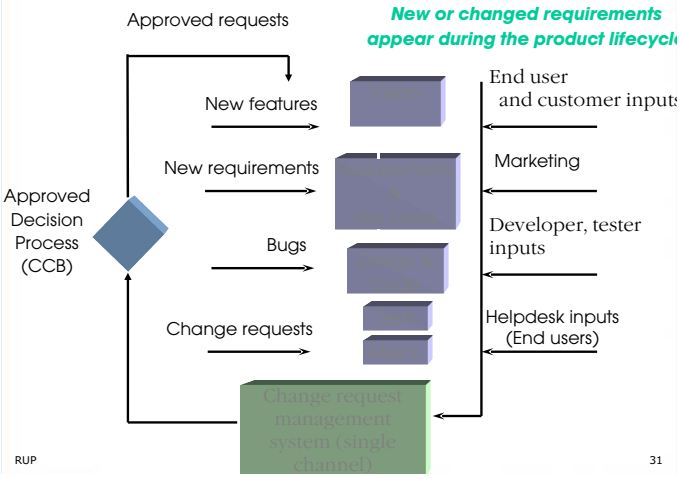
- Verification - to prove the specification was followed
  - Can be performed by development team
  - Can be automatized
  - Testing discipline
- Validation - to prove the specification was right
  - The end user is needed
  - The end user has to have a chance to „touch“ the software
  - Requirements management discipline



MANAGE CHANGE

- Define the Change Request Management
  - Enhancement requests, defects
  - Who is responsible for what change
  - The change request lifecycle
- Configuration and version management
  - Control the project files and assets
  - Build management
  - Parallel development
- The idea is: Avoid chaos!

MANAGE CHANGE II



RATIONAL UNIFIED PROCESS

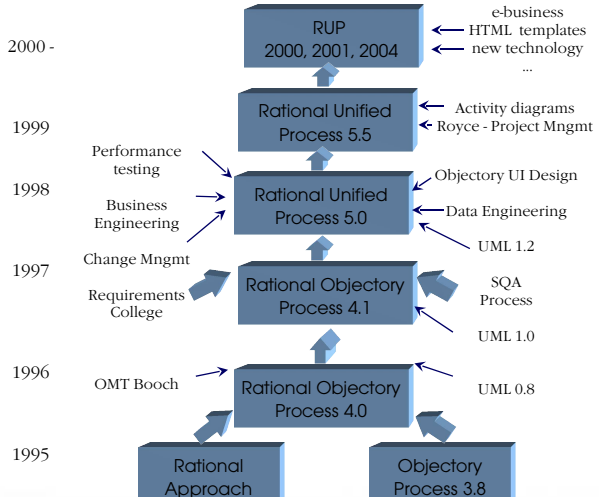
- Solution, that covers best practices

INTRODUCTION INTO THE RUP

WHAT IS PROCESS?

- Process describes **WHO** does **WHAT**, **WHEN** and **HOW** to achieve specific goal
- Goal of software development is create new SW product or extensive the existing product
  - Gives a instruction for activity of working team
  - Defines, which artifacts and when must be created
  - Task management
  - Criteria for tracking and evaluating activites and results of project

BRIEF HISTORY OF RUP



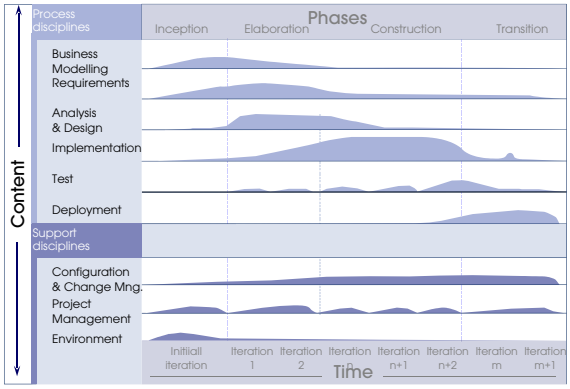
MODEL OF RUP

- Process describes WHO does WHAT, WHEN and HOW to achieve specific goal
- Roles – WHO
  - One or more people in the role, person in one or more roles
- Activites – HOW
  - A unit of work a role may be asked to perform
- Artifacts – WHAT
  - A piece of information than is produced, modified, or used by a process
- Disciplines – WHEN
  - Collection of related activities that are related to a major 'area of concern'

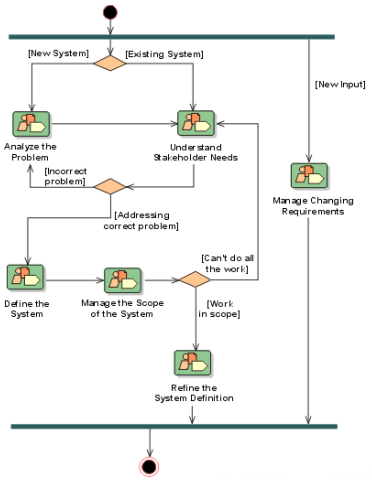
DISCIPLINES I

- Workflow is a succession of activites, that make observable result
- Organization of RUP activities by means of
  - Basic disciplines
  - Details of workflows
  - Iterational workflow

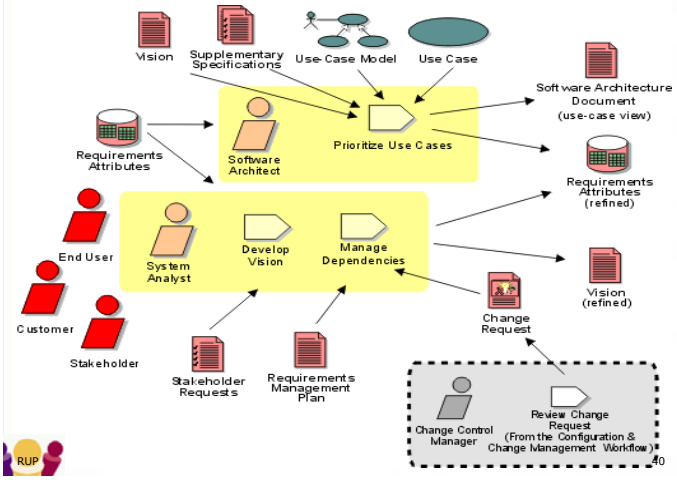
DISCIPLINES II



DISCIPLINE – ACTIVITY DIAGRAM



DISCIPLINE DETAIL - WORKFLOW



RUP - ACTIVITY

Purpose

- To outline the functionality of the system.
- To define what will be handled by the system and what will be handled outside the system.
- To define who and what will interact with the system.
- Divide the model into packages with actors and use cases.
- Create diagrams of the use-case model.
- Develop a survey of the use-case model.

Input Artifacts:	Resulting Artifacts:
Business Use-Case Model Business Object Model Use-Case Modeling Guidelines Stakeholder Requests Vision Glossary	Use Case Actor Use-Case Model Supplementary Specifications

Role: System Analyst

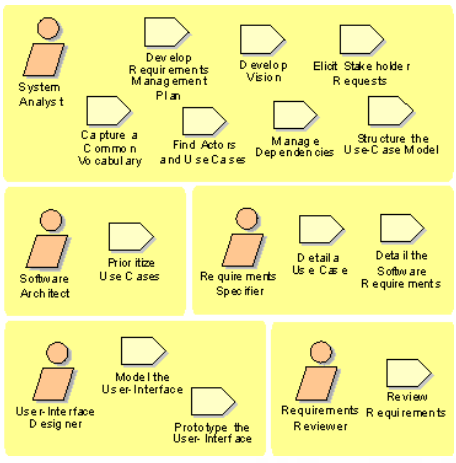
Work Guidelines:

- Use-Case Workshop
- Storyboarding

Tool Mentors

- Finding Actors and Use Cases Using Rational Rose
- Finding Actors and Use Cases Using Rational Rose
- Managing Use Cases Using Rational Rose and Rational RequisitePro

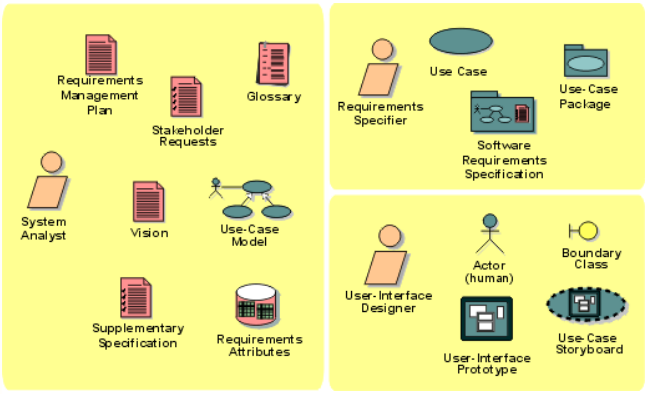
DISCIPLINE'S ACTIVITIES SUMMARY



RUP - ARTIFACT

Vision	The **Vision** defines the stakeholders view of the product to be developed, specified in terms of the stakeholders key needs and features. Containing an outline of the envisioned core requirements, it provides the contractual basis for the more detailed technical requirements.
Role:	System Analyst
Templates:	■ HTML Template: Vision ■ HTML Template: Vision (for small projects) ■ Others
Examples:	■ Vision, from the Course Registration System example website ■ Vision, from the Collegiate Sports Paging System (e-business)
More Information:	■ Checkpoints:Vision ■ Checkpoints: Stakeholder Requests ■ Checkpoints: Requirements Attributes ■ Artifact: Requirements Management Plan
■ Purpose ■ Timing ■ Responsibility ■ Additional Information ■ Tailoring	

DISCIPLINE'S ARTIFACTS SUMMARY



ITERATIVE DEVELOPMENT

ITERATIVE DEVELOPMENT

- The initial requirements and design are most probably faulty
- Finding faults late means additional costs and/or project cancelation

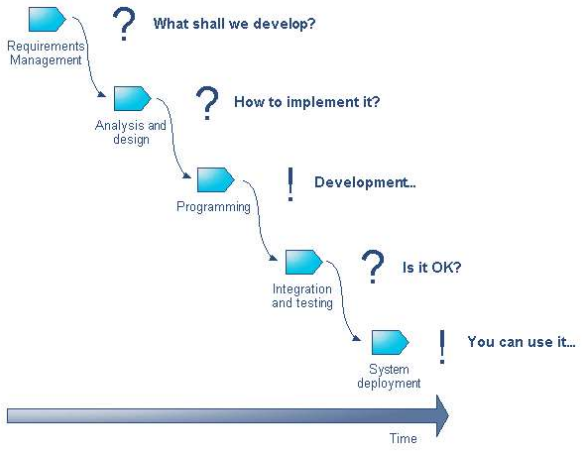
$$$

Time and money spent for a bad design implementation are lost

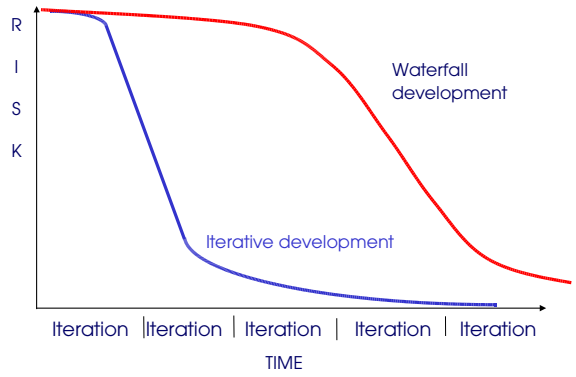
WRONG ASSUMPTIONS

- The requirements won't change
- We can design the system with paper&pencil
- The initial design is the best one
- Our programmers don't make mistakes
- The integration is not a problem
- The integration doesn't cause new defects
- The test are useful only when the system is complete
- ...

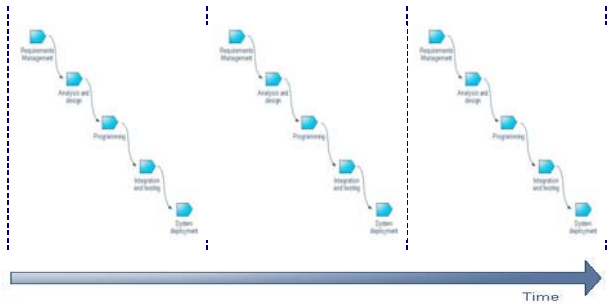
SOFTWARE DEVELOPMENT



ITERATIVE DEVELOPMENT



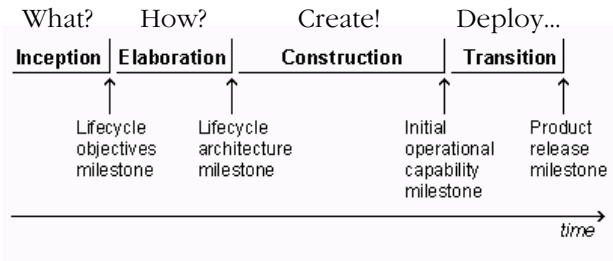
ITERATION = SMALL WATERFALL



- First iteration finds the biggest risks
- Next iterations eliminate the risks as soon as possible
- Each iteration contains executable software
- Each iteration contains integration and testing

PHASES AND MILESTONES

- We change the focus during the project lifecycle
- Four phases in UP



INCEPTION PHASE

- Specify the real project scope
- Milestone: Lifecycle Objectives Milestone (LCO)
  - Stakeholder concurrence on scope definition and cost/schedule estimates
  - Agreement that the right set of requirements have been captured and that there is a shared understanding of these requirements
  - Agreement that the cost/schedule estimates, priorities, risks, and development process are appropriate
  - All risks have been identified and a mitigation strategy exists for each

ELABORATION PHASE

- Establish the right software architecture and prove it works
- Milestone: Lifecycle Architecture Milestone (LCA)
  - The product Vision and requirements are stable
  - The architecture is stable
  - The key approaches to be used in test are proven
  - Test and evaluation of executable prototypes have demonstrated that the major risk elements have been addressed and have been credibly resolved
  - The iteration plans for the construction phase are defined and supported by credible estimates
  - All stakeholders agree that the current vision can be met if the current plan is executed to develop the complete system, in the context of the current architecture
  - Actual resource expenditure versus planned expenditure are acceptable

CONSTRUCTION PHASE

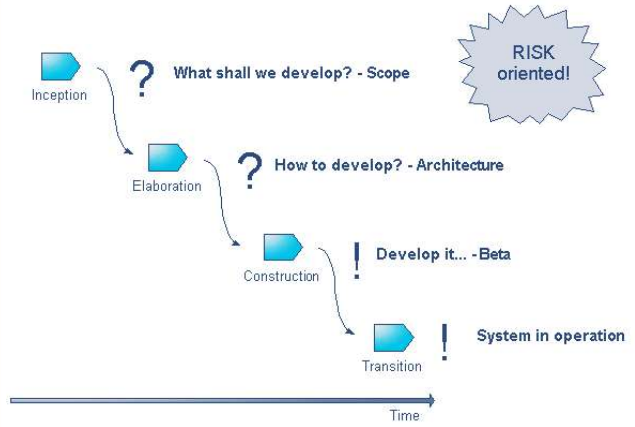
- „Mass production“
- Milestone: Initial Operational Capability (IOC)
  - Is this product release stable and mature enough to be deployed in the user community?
  - Are all the stakeholders ready for the transition into the user community?
  - Are actual resource expenditures versus planned still acceptable?

TRANSITION PHASE

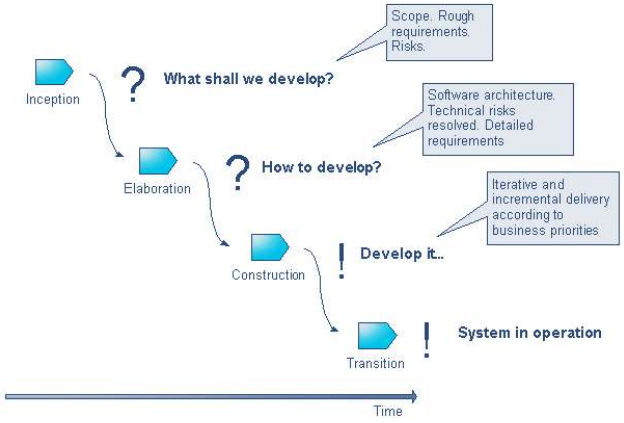
- Product in operational environment
- Documentation and installation media
- Training and support

- Milestone: Product Release
  - Is the user satisfied?
  - Are actual resources expenditures versus planned expenditures acceptable?

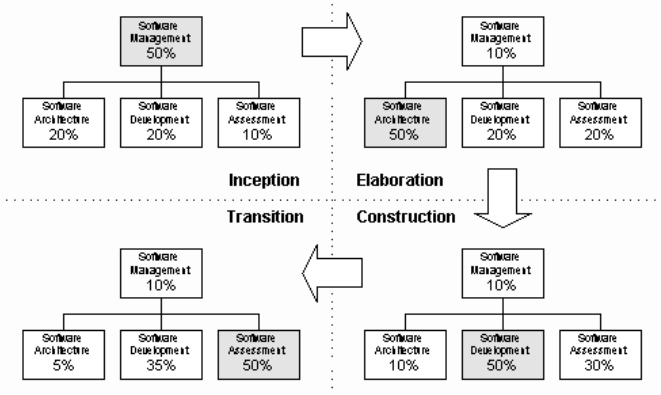
ITERATIVE PROJECT LIFECYCLE



ITERATIVE PROJECT LIFECYCLE



SPREADING THE EFFORTS IN TEAM



UNIFIED PROCESS — USE-CASE DRIVEN

Actor: represents the users

Use case: an interaction of an actor (user) with the system

Use-case model: the collection of all user interactions (use cases)

➡ **Use-case model represents all system user functionality**

(functions that add value for the users)

➡ **Use cases drive the development process**

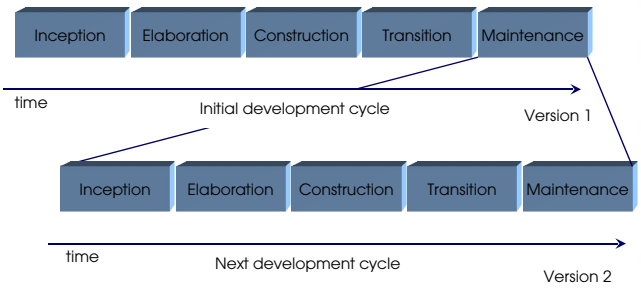
(transform use case model into analysis, design and implementation models)

➡ **Supports seamless traceability between models**

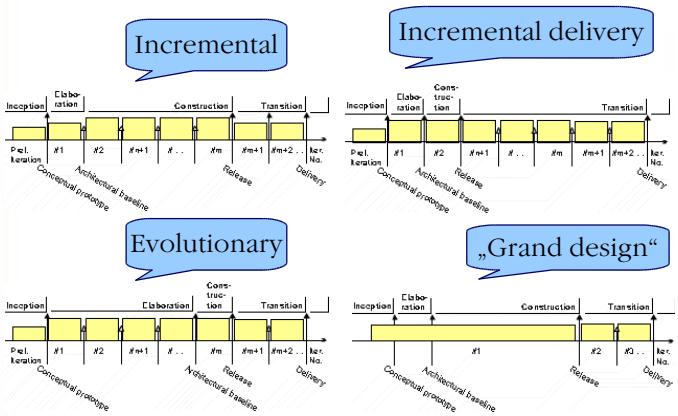
PLANNING WITH USE CASES

- Iterations can be planned according to use case model
  - Choose a set of use cases (or scenarios) for the iteration
  - Incrementaly enhance the functionality in subsequent iterations
- Start with the use cases which represent a risk
  - Use the use case attributes
  - Don‘t forget the other requirements (usability, reliability, performance, supportability, technological constraints)
- Criteria for planning with use cases:
  - We don‘t know how to implement the functionality
  - Repeating task (standardized solution)
  - Important functionality

DEVELOPMENT CYCLES



ITERATIVE DEVELOPMENT STRATEGIES



HOW MANY AND HOW LONG?

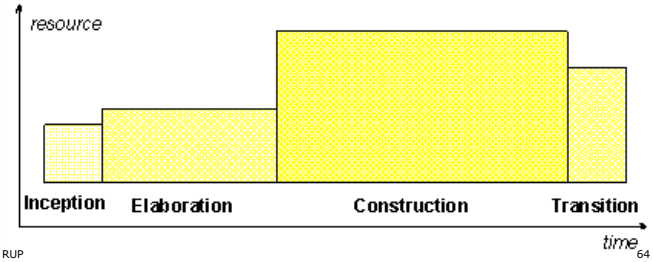
- How many iterations for a project?
- How long should the iteration be?
- There is no simple answer
  - How large the system is?
  - How many people are involved?
- 3 weeks - 3 months for iteration are advised

	Amount	P	E	K	Z
Few	3	0	1	1	1
Typical	6	1	2	2	1
Many	9	1	3	3	2

PLANNING THE PHASES

- Incremental lifecycle strategy

	Inception	Elaboration	Construction	Transition
Effort	~5 %	20 %	65 %	10%
Schedule	10 %	30 %	50 %	10%



ITERATIVE DEVELOPMENT ADVANTAGES

- The serious misunderstandings are dicovered early
- Reflection with users and customers
- Concentration on the key problems and areas
- Objective project status assessment
  - Metrics, testing...
- The discrepancies between requirements, analysis, design and implementation are discovered early
- Balanced people workload
- Team learning

ITERATIVE DEVELOPMENT RISKS

- More work
  - More planning, more supervision, more customer involvement
- Can‘t create a detailed plan for the whole project
  - The short-term plan should be detailed
- Let‘s get started, we‘ll decide where to go later
  - The „no time to get these things right“ approach
- Acknowledge rework up-front
  - Waterfal SDLC - too much rework comes at the very end
  - Iterative SDLC - the rework is continuous (so you can plan resources for rework)
- Project not converging
  - Don‘t allow the meaningless rework
  - Be aware of new, unplanned requirements

ITERATIVE DEVELOPMENT RISKS II

- Planning only finished artifacts
  - Learn to plan iteratively - the artifacts should and will evolve
  - Don‘t think only in terms of „specifications“ (documents)
- Hitting hard problems late
  - Putting your head in the sand
  - Forgetting about new risks
- The project price
  - Fixing the price early (inception) is usualy wrong
  - Fix the price at the LCA (end of the elaboration)
- Too many iterations
  - Don‘t talk about builds as iterations
- Overlapping iterations
  - When to start to plan the next iteration?

BASIC PRINCIPLES OF RUP

HOW USE RUP?

- The goal of software development is to deliver first-rate software
  - Quality
  - Quantity (scope)
  - Term
  - Budget
- Observe Best practices

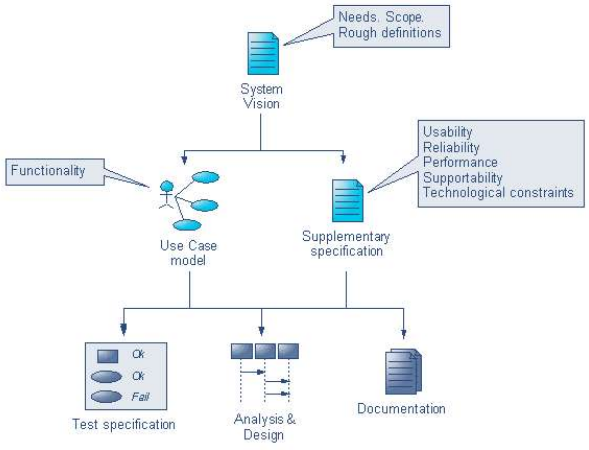
BASIC STRATEGY

- Vision definition
- Architecture
- Planning and evaluating
- Iterative implementation and testing
- Change management
- Users support

VISION DEFINITION

- Artifact Vison – scope of system
  - Plus other requirements
- Answers for these questions
  - What are the key problems (Glossary)
  - What problem we must solve (Problem statement)
  - Who are the stakeholders, users, what they need
  - Which features should have the product
  - Which functionality is needed (use cases)
  - Which other requirements we have to realize
  - Which technological constraints we have to meet

REQUIREMENTS MANAGEMENT



UNIFIED PROCESS — ARCHITECTURE-CENTRIC

Architecture: all the **decisions** made about a system that are **applied consistently** and **pervasively** throughout the system

- Represents the **most significant static** and **dynamic** aspects of the system (subsystems, interfaces, dependencies, etc.)
  - Provides **different views** of the system
  - Describes the **foundation** of the system
- **Key use cases (5-10%) determine the architecture**
  - Use cases describe the **function** of the system **WHAT**
  - Architecture describes the **form** of the system **HOW**

GOAL -> a stable architecture early in the development

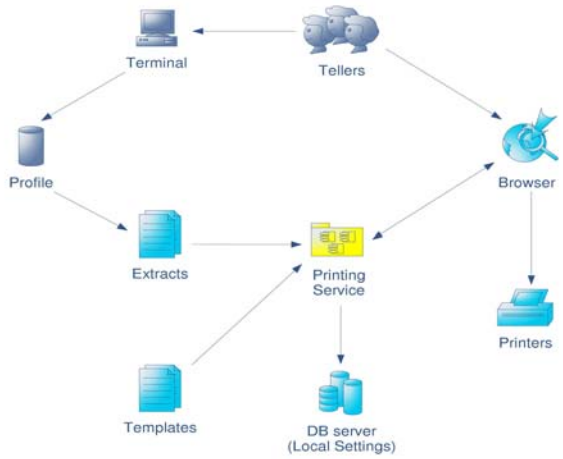
DEFINITION(S)

- Software architecture encompasses the set of significant decisions about the organization of a software system
  - Selection of the structural elements and their interfaces by which a system is composed
  - Behavior as specified in collaborations among those elements
  - Composition of these structural and behavioral elements into larger subsystems
  - Architectural style that guides this organization
- http://www.sei.cmu.edu/architecture/definitions.html
- Architecture is the 20 % of the system that is really important

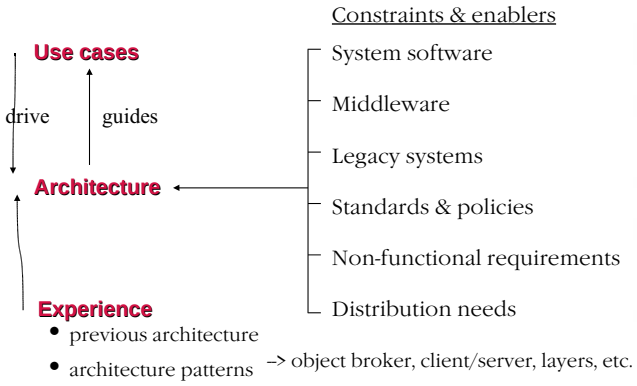
ARCHITECTURE REPRESENTATIONS

- Big-picture (M-architecture)
  - High abstraction
  - Easy to understand
    - For stakeholders (customer)
    - For team members
  - The whole system described on one paper
  - Everyone is satisfied with the great idea
- Detailed description (T-architecture)
  - Technical documentation
    - Software architecture document
    - 4+1 views
  - Relevant for technical people
    - Developers
    - Customer (IT people)

EXAMPLE - BIG PICTURE I

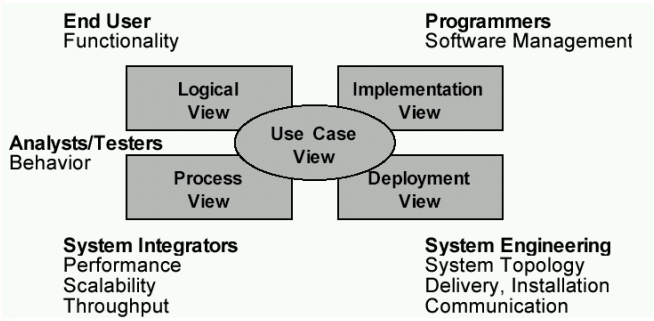


DETERMINING ARCHITECTURE



ARCHITECTURE

- System architecture must have ability to be seen from different point of view
- 4 + 1 view



PLANNING – MANAGING ACCORDING TO PLAN

- Artifact Software Development Plan contents many plans
  - Project plans
  - Iteration plans
  - Project organisation, Requirements management plan, Configuration plan, QA plan, Test plan, etc.
- Different projects can have a different way of use
  - Small project – some plans can be expressed by one sentence
- Format of plan is not important, but important is planning and thinking about project progress

PLANNING - RISKS

- Identify fundamental risks as soon as possible
- Artifact Risk List

- Risks are direct and indirect
- Attributes of risk
  - Probability of occurence
  - Impact on the project
- Percieved risk
  - Risk avoidance
  - Risk transfer
  - Risk acceptance
- When accepting a risk
  - Risk mitigation
  - Define a contingency plan

PLANNING – BUSINESS CASE

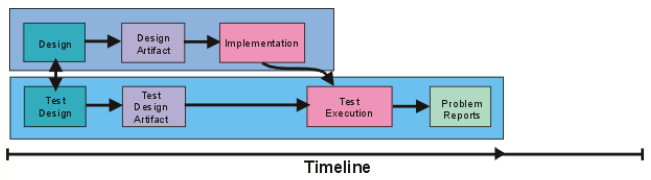
- Artifact Business Case
- The main purpose of the Business Case is to develop an economic plan for realizing the project vision presented in the Vision
- The Business Case is used to make an accurate assessment of the return on investment (ROI) provided by the project
- It provides information to the economic decision makers on the project's economic worth and is used to determine whether the project should move ahead

EVALUATION – PROJECT STATE

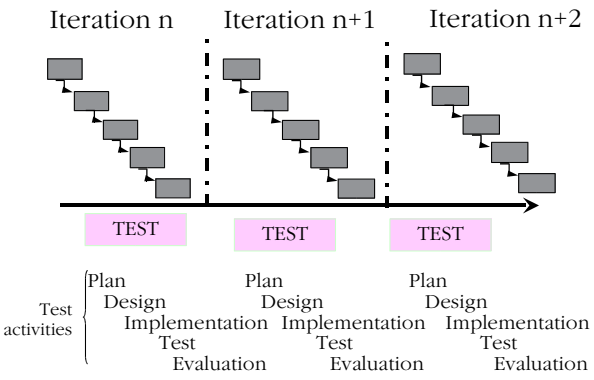
- One of the objectives of the process is to ensure that the expectations of all parties are synchronized and consistent
- Artifact Status Assessment – for example weekly status assessment
  - Problems with managing, technologies and customer
  - Defining tasks and terms
- The key role during the project are **Metrics** (measuring)
  - Primitive metrics
    - Number of worked hours
    - Number of created artifacts with the same type
    - Planned hours
  - Derived metrics
    - Plan success – ration between planed and worked hours

ITERATIVE IMPLEMENTATION & TESTING

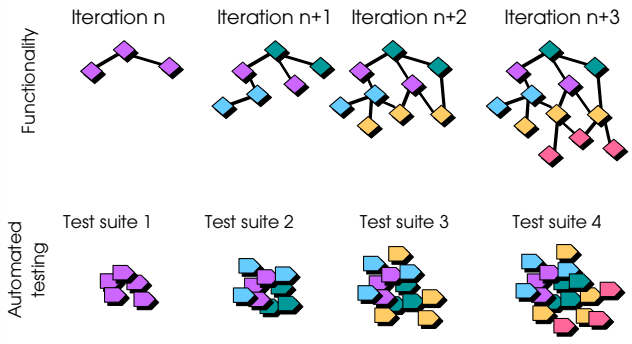
- Gradual code developing and its testing
- Executive version in the end of the iteration
- In the end of the Elaboration phase
  - Architectonic prototype
  - User-interface prototype (optional)
- Unit tests – tests of programmers



ITERATIVE (REGRESSION) TESTING



ITERATIVE TESTING II



CHANGE MANAGEMENT

- Changes are common
- Changes are not usually bad, bad is no change management
- Artifact Change Request
  - Defined workflow
  - Evaluating, implementing
  - Record of decisions
- Configuration management and Release management
  - Depends on project size

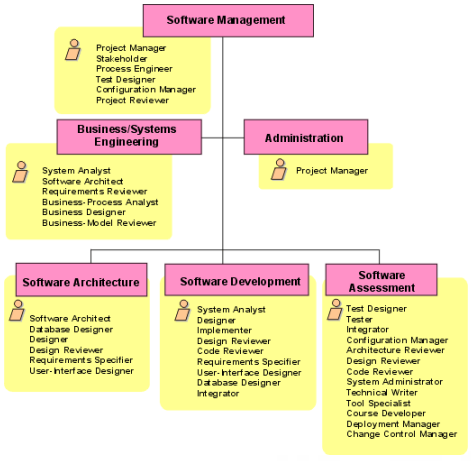
USERS SUPPORT

- Documentation
  - User's documentation
  - Help
  - Release notes
  - Install's documentation
  - Programmer's documentation
- Training materials
- …

WHAT DOES HAPPEN, WHEN WE HAVE ...

- No vision?
  - We don't know, where we go and we don't know, if we go in a right way
- No plan?
  - We are unable to track progress and benefits
- No risk list?
  - Dangerous in concentrating on unimportant things and unexpecting suprise
- No Business case?
  - We risk wasting with money and time
- No architecture?
  - We risk bad communication, synchronization and integration, performance and scalability

ORGANISATION STRUCTURE



KEY ROLES

- Requirements - System Analyst
  - Business-Process Analyst
- Analysis and design - Software Architect
  - Database Designer
- Implementation - Implementer
- Testing - Tester
- Managing - Project Manager
- Others - Stakeholder
  - Configuration Manager
  - Change Control Manager

RESOURCES

BOOKS

- Rational Unified Process 2004.05.00 - Rational Software Corporation
- Rational Unified Process - An Introduction; Philippe Kruchten, 2000
- Process Paterns; Scott Ambler, 1999
- More Process Patterns; Scott Ambler, 1999
- Unified Process Inception Phase; Scott Ambler, 2000
- Unified Process Elaboration Phase; Scott Ambler, 2000
- Unified Process Construction Phase; Scott Ambler, 2000
- Unified Process Transition Phase; Scott Ambler, 2000
- Unified Software Development Process; Jacobson, Booch, Rumbaugh, 1999
- Software Project Management: A Unified Framework; Walker Royce, 1998

INTERNET

- www.rational.com
- www.therationaledge.com
- www.cetus-links.org