Software Development

Process, Models, Methods, Diagrams

Software Development Life Cyles

Part - II

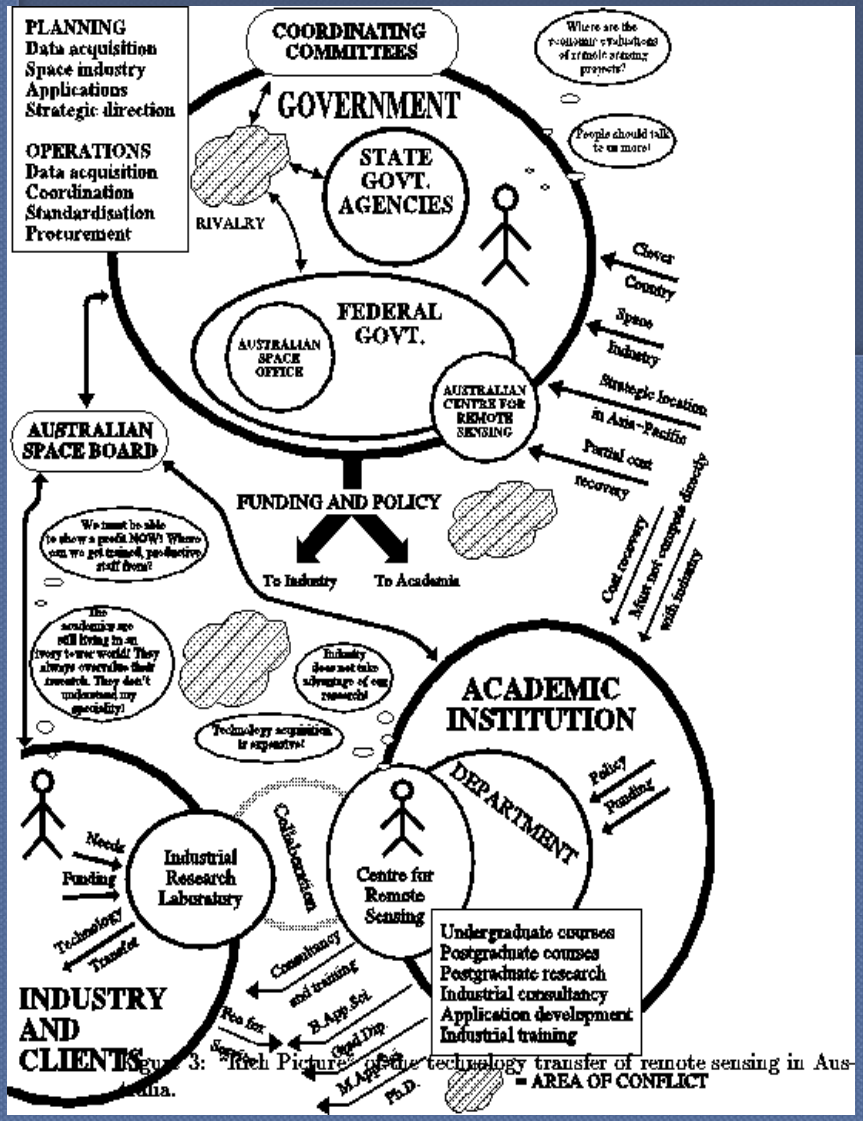
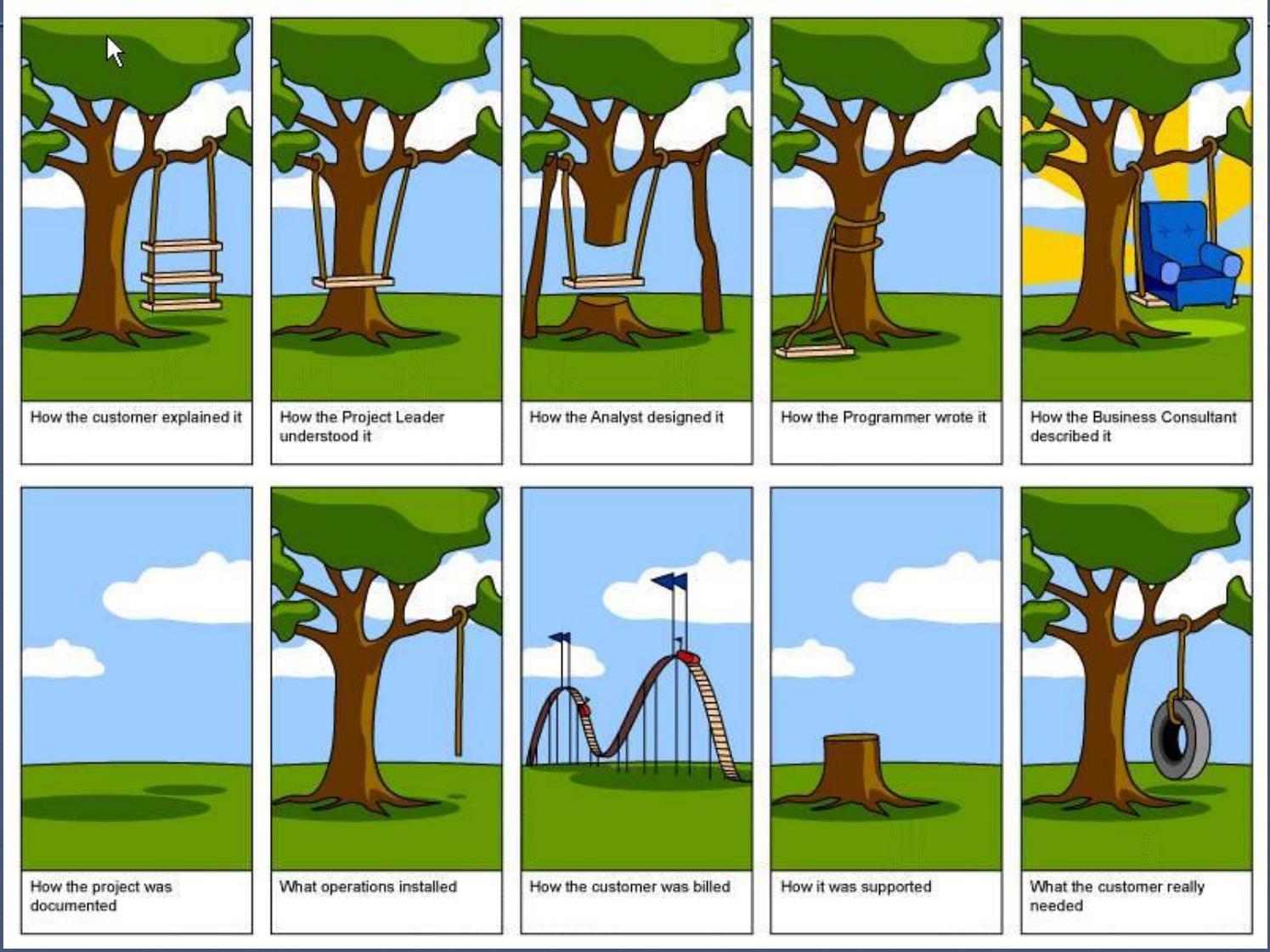


Figure 3: "Rich Picture" of the technology transfer of remote sensing in Australia.

Famous SDLC Figure



Capability Maturity Model (CMM)

- A bench-mark for measuring the maturity of an organization's software process
- CMM defines 5 levels of process maturity based on certain Key Process Areas (KPA)

CMM Levels

Level 5 – Optimizing (< 1%)

-- process change management
-- technology change management
-- defect prevention

Level 4 – Managed (< 5%)

-- software quality management
-- quantitative process management

Level 3 – Defined (< 10%)

-- peer reviews
-- intergroup coordination
-- software product engineering
-- integrated software management
-- training program
-- organization process definition
-- organization process focus

Level 2 – Repeatable (~ 15%)

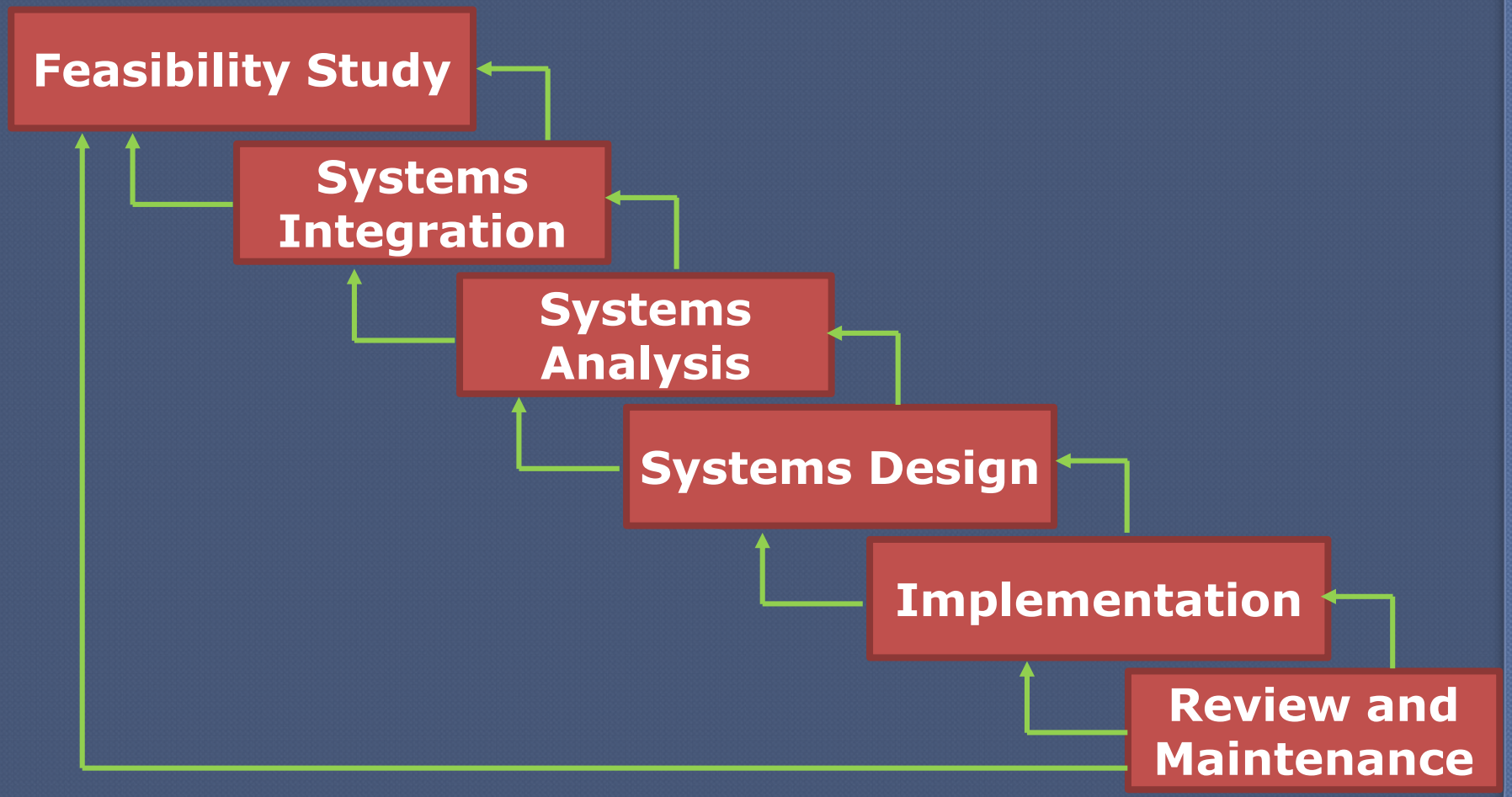
-- software configuration management
-- software quality assurance
-- software project tracking and oversight
-- software project planning
-- requirements management

Level 1 – Initial (~ 70%)

Traditional Structured Analysis

- Described by W. W. Royce, 1970, IEEE WESCON, *Managing the development of large software systems*.
- Decomposition in terms of Function and Data
- Modularity available only at the file level
  - cf. C language's *static* keyword (=="file scope")
- Data was not encapsulated:
  - Global Scope
  - File Scope
  - Function Scope (automatic, local)
- Waterfall Method of Analysis and Design

Waterfall Methods



Waterfall Process Assumptions

- Requirements are known up front before design
- Requirements rarely change
- Users know what they want, and rarely need visualization
- Design can be conducted in a purely abstract space, or trial rarely leads to error
- The technology will all fit nicely into place when the time comes (the apocalypse)
- The system is not so complex. (Drawings are for wimps)

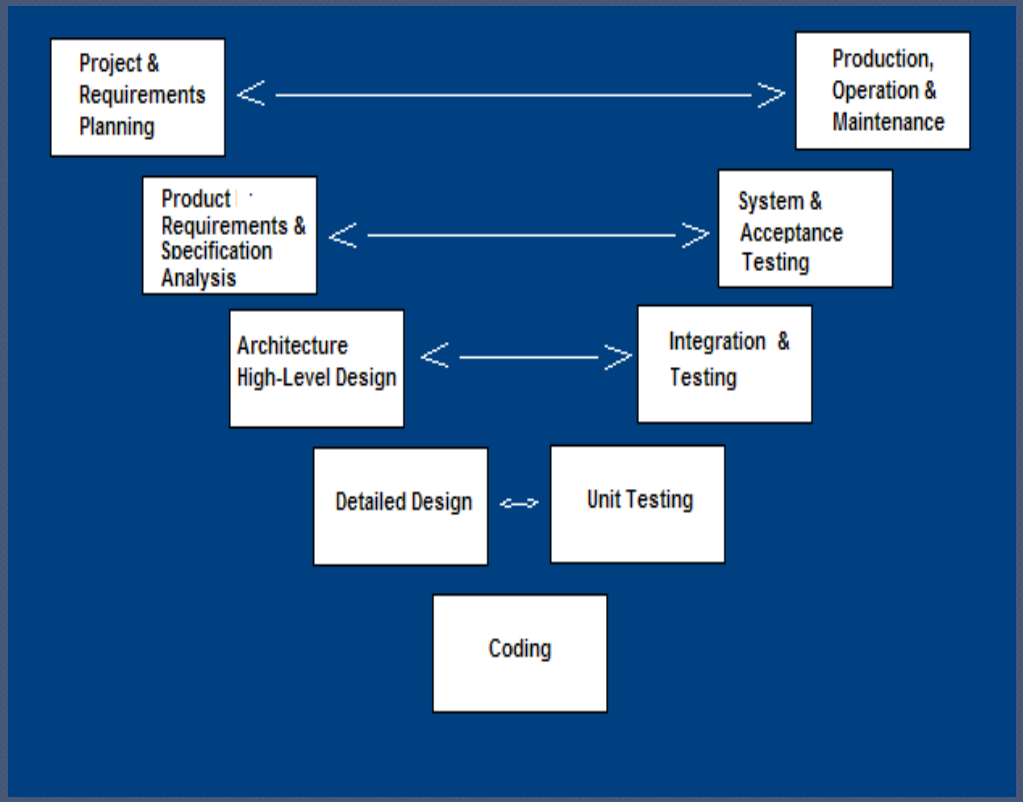
Structured Analysis Problems

- Reuse is complicated because Data is strewn throughout many different functions
  - Reuse is usually defined as code reuse and is implemented through cutting and pasting of the same code in multiple places. What happens when the logic changes?
    - coding changes need to be made in several different places
    - changing the function often changes the API which breaks other functions dependent upon that API
    - data type changes need to be made each time they are used throughout the application

Waterfall Process Limitations

- Big Bang Delivery Theory
- The proof of the concept is relegated to the very end of a long singular cycle. Before final integration, only documents have been produced.
- Late deployment hides many lurking risks:
  - technological (well, I *thought* they would work together…)
  - conceptual (well, I *thought* that's what they wanted…)
  - personnel (took so long, half the team left)
  - User doesn't get to *see* anything real until the very end, and they always *hate* it.
  - System Testing doesn't get involved until later in the process.

V-Shaped SDLC Model



- A variant of the Waterfall that emphasizes the verification and validation of the product
- Testing of the product is planned in parallel with a corresponding phase of development

V-Shaped Steps

- Project and Requirements Planning – allocate resources
- Product Requirements and Specification Analysis – complete specification of the software system
- Architecture or High-Level Design – defines how software functions fulfill the design
- Detailed Design – develop algorithms for each architectural component
- Production, operation and maintenance – provide for enhancement and corrections
- System and acceptance testing – check the entire software system in its environment
- Integration and Testing – check that modules interconnect correctly
- Unit testing – check that each module acts as expected
- Coding – transform algorithms into software

V-Shaped Strengths

- Emphasize planning for verification and validation of the product in early stages of product development
- Each deliverable must be testable
- Project management can track progress by milestones
- Easy to use

V-Shaped Weaknesses

- Does not easily handle concurrent events
- Does not handle iterations or phases
- Does not easily handle dynamic changes in requirements
- Does not contain risk analysis activities

When to use the V-Shaped Model

- Excellent choice for systems requiring high reliability – hospital patient control applications
- All requirements are known up-front
- When it can be modified to handle changing requirements beyond analysis phase
- Solution and technology are known

Iterative and evolutionary development: AGILE

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

Principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity–the art of maximizing the amount of work not done–is essential.
- The best architectures, requirements, and designs emerge from self organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

AGILE Methodologies

Methodologies share common principles, but differ in practices

- eXtreme Programming (XP)
- Scrum
- Evolutionary Project Management (Evo)
- Unified Process (UP)
- Crystal
- Lean Development (LD)
- Adaptive Software Development (ASD)
- Dynamic System Development Method (DSDM)
- Feature Driven Development (FDD)

RUP

- Rational Software Corp (now part of IBM), lead by 3 amigos: Grady Booch, James Rumbaugh, Ivar Jacobson
- Derived from several methodologies at that time
- Micro and Macro development process
- Micro deals with tactical issues (daily activities)
- Macro process has inception, elaboration, construction, and transition
- Generally viewed as heavy weight process
- Agile in sprit, but can get very ceremonial
  - Emphasizes iterative cycles, constant feedback
  - Developed along with UML which provides for several forms of documentation
- Comes from disciplined process oriented angle
- Not easy to tailor for small projects

The Rational Unified Process

- RUP is a method of managing OO Software Development
- 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

RUP Features

- Online Repository of Process Information and Description in HTML format
- Templates for all major artifacts, including:
  - RequisitePro templates (requirements tracking)
  - Word Templates for Use Cases
  - Project Templates for Project Management
- Process Manuals describing key processes
- http://sce.uhcl.edu/helm/rationalunifiedprocess/process/templates.htm

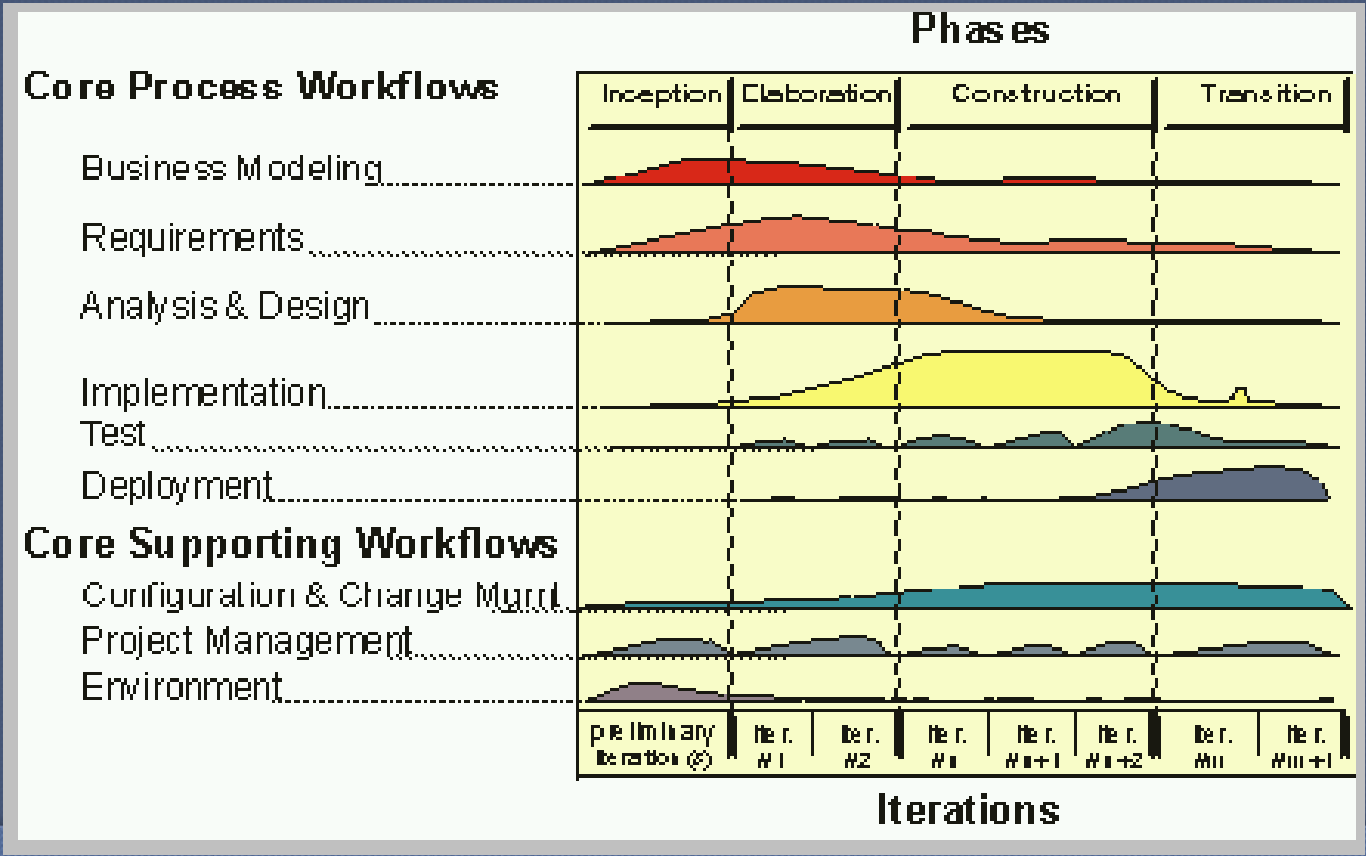
The Phases

Two Dimensions

The process can be described in two dimensions, or along two axis:

• the horizontal axis represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones.

• the vertical axis represents the static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows.



An Iterative Development Process…

- Recognizes the reality of changing requirements
  - Caspers Jones's research on 8000 projects
    - 40% of final requirements arrived after the analysis phase, after development had already begun
- Promotes early risk mitigation, by breaking down the system into mini-projects and focusing on the riskier elements first
- Allows you to "plan a little, design a little, and code a little"
- Encourages all participants, including testers, integrators, and technical writers to be involved earlier on
- Allows the process itself to modulate with each iteration, allowing you to correct errors sooner and put into practice lessons learned in the prior iteration
- Focuses on component architectures, not final big bang deployments

An Incremental Development Process…

- Allows for software to evolve, not be produced in one huge effort
- Allows software to improve, by giving enough time to the evolutionary process itself
- Allows the system (a small subset of it) to actually run much sooner than with other processes
- Allows for the management of risk, by exposing problems earlier on in the development process

Goals and Features of Each Iteration

- The primary goal of each iteration is to slowly chip away at the risk facing the project, namely:
  - performance risks
  - integration risks (different vendors, tools, etc.)
  - conceptual risks (ferret out analysis and design flaws)
- Perform a "miniwaterfall" project that ends with a delivery of something tangible in code, available for scrutiny by the interested parties, which produces validation or correctives
- Each iteration is risk-driven
- The result of a single iteration is an increment--an incremental improvement of the system, yielding an evolutionary approach

Risk Management

- Identification of the risks
- Iterative/Incremental Development
- The prototype or pilot project
  - Booch's "Tiger Team"
- Early testing and deployment as opposed to late testing in traditional methods

Major Milestones

- Inception Phase
- Elaboration Phase
- Construction Phase
- Transition Phase

Inception

During the inception phase, you establish the business case for the system and delimit the project scope.

To accomplish this you must identify all external entities with which the system will interact (actors) and define the nature of this interaction at a high-level. This involves identifying all use cases and describing a few significant ones.

The business case includes success criteria, risk assessment, and estimate of the resources needed, and a phase plan showing dates of major milestones.

Inception

The outcome of the inception phase is:

- A vision document: a general vision of the core project's requirements, key features, and main constraints.
- A initial use-case model (10% -20%) complete)
- An initial project glossary (may optionally be partially expressed as a domain model)
- An initial business case, which includes business context, success criteria (revenue projection, market recognition, and so on), and financial forecast
- An initial risk assessment
- A project plan, showing phases and iterations
- A business model, if necessary
- One or several prototypes

Inception

At the end of the inception phase is the first major project milestone. The evaluation criteria for the inception phase are:

- Stakeholder concurrence on scope definition and cost/schedule estimates.
- Requirements understanding as evidenced by the fidelity of the primary use cases.
- Credibility of the cost/schedule estimates, priorities, risks, and development process.
- Depth and breadth of any architectural prototype that was developed.
- Actual expenditures versus planned expenditures.

The project may be cancelled or considerably re-thought if it fails to pass this milestone.

Elaboration

In the elaboration phase, an executable architecture prototype is built in one or more iterations, depending on the scope, size, risk, and novelty of the project.

This effort should at least address the critical use cases identified in the inception phase, which typically expose the major technical risks of the project.

While an evolutionary prototype of a production-quality component is always the goal, this does not exclude the development of one or more exploratory, throwaway prototypes to mitigate specific risks such as design/requirements trade-offs, component feasibility study, or demonstrations to investors, customers, and end-users

Elaboration

The outcome of the elaboration phase is:

• A use-case model (at least 80% complete) — all use cases and actors have been identified, and most usecase descriptions have been developed.
• Supplementary requirements capturing the non functional requirements and any requirements that are not associated with a specific use case.
• A Software Architecture Description.
• An executable architectural prototype.
• A revised risk list and a revised business case.
• A development plan for the overall project, including the coarse-grained project plan, showing iterations" and evaluation criteria for each iteration.
• An updated development case specifying the process to be used.
• A preliminary user manual (optional).

Elaboration

At the end of the elaboration phase, you examine the detailed system objectives and scope, the choice of architecture, and the resolution of the major risks. The main evaluation criteria for the elaboration phase involves the answers to these questions:

- Is the vision of the product stable?
- Is the architecture stable?
- Does the executable demonstration show that the major risk elements have been addressed and credibly resolved?
- Is the plan for the construction phase sufficiently detailed and accurate? Is it backed up with a credible basis of estimates?
- Do all stakeholders agree that the current vision can be achieved if the current plan is executed to develop the complete system, in the context of the current architecture?
- Is the actual resource expenditure versus planned expenditure acceptable?

The project may be aborted or considerably re-thought if it fails to pass this milestone.

Construction Phase

During the construction phase, all remaining components and application features are developed and integrated into the product, and all features are thoroughly tested.

The construction phase is, in one sense, a manufacturing process where emphasis is placed on managing resources and controlling operations to optimize costs, schedules, and quality.

In this sense, the management mindset undergoes a transition from the development of intellectual property during inception and elaboration, to the development of deployable products during construction and transition.

Construction Phase

The outcome of the construction phase is a product ready to put in hands of its end-users. At minimum, it consists of:

- The software product integrated on the adequate platforms.
- The user manuals.
- A description of the current release.

Construction Phase

At this point, you decide if the software, the sites, and the users are ready to go operational, without exposing the project to high risks. This release is often called a “beta” release. The evaluation criteria for the construction phase involve answering these questions:

- Is this product release stable and mature enough to be deployed in the user community?
- Are all stakeholders ready for the transition into the user community?
- Are the actual resource expenditures versus planned expenditures still acceptable?

Transition may have to be postponed by one release if the project fails to reach this milestone.

Transition Phase

This includes:

- “beta testing” to validate the new system against user expectations
- parallel operation with a legacy system that it is replacing
- conversion of operational databases
- training of users and maintainers
- roll-out the product to the marketing, distribution, and sales teams

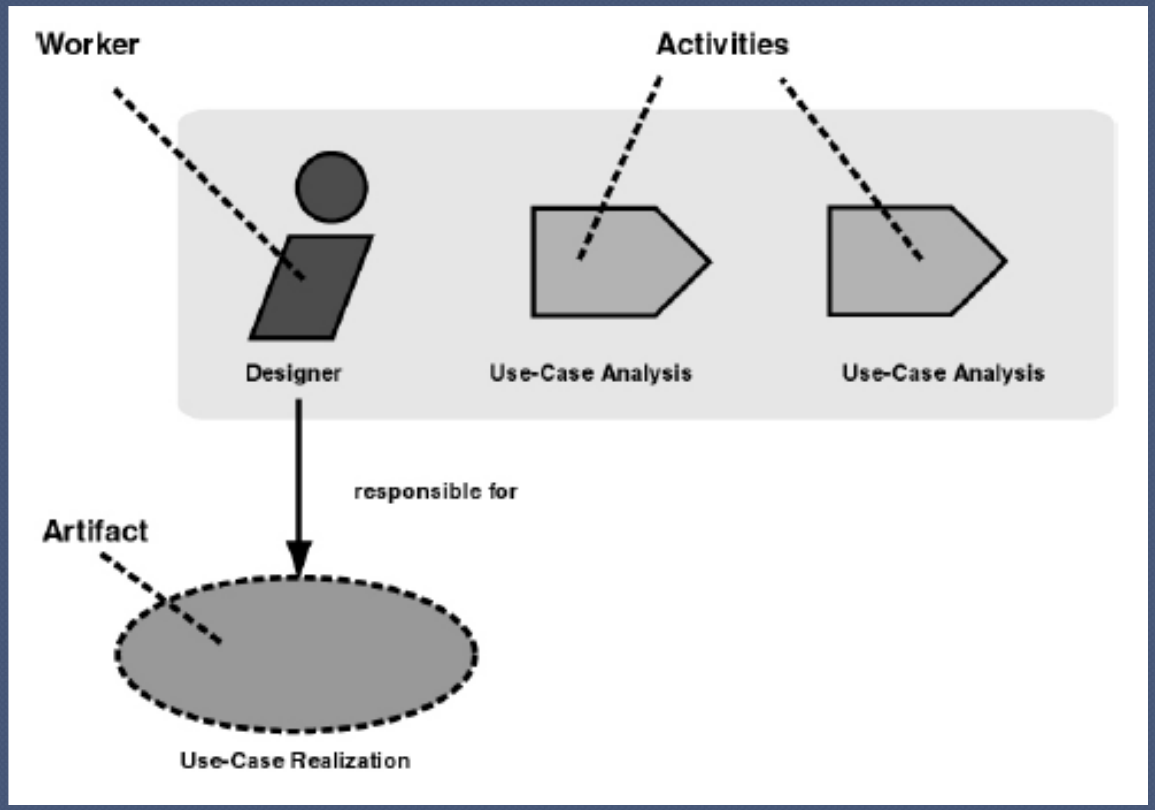
This phase can range from being very simple to extremely complex, depending on the type of product. For example, a new release of an existing desktop product may be very simple, whereas replacing a nation's air-traffic control system would be very complex.

Transition Phase

The primary evaluation criteria for the transition phase involve the answers to these questions:

- Is the user satisfied?
- Are the actual resources expenditures versus planned expenditures still acceptable?

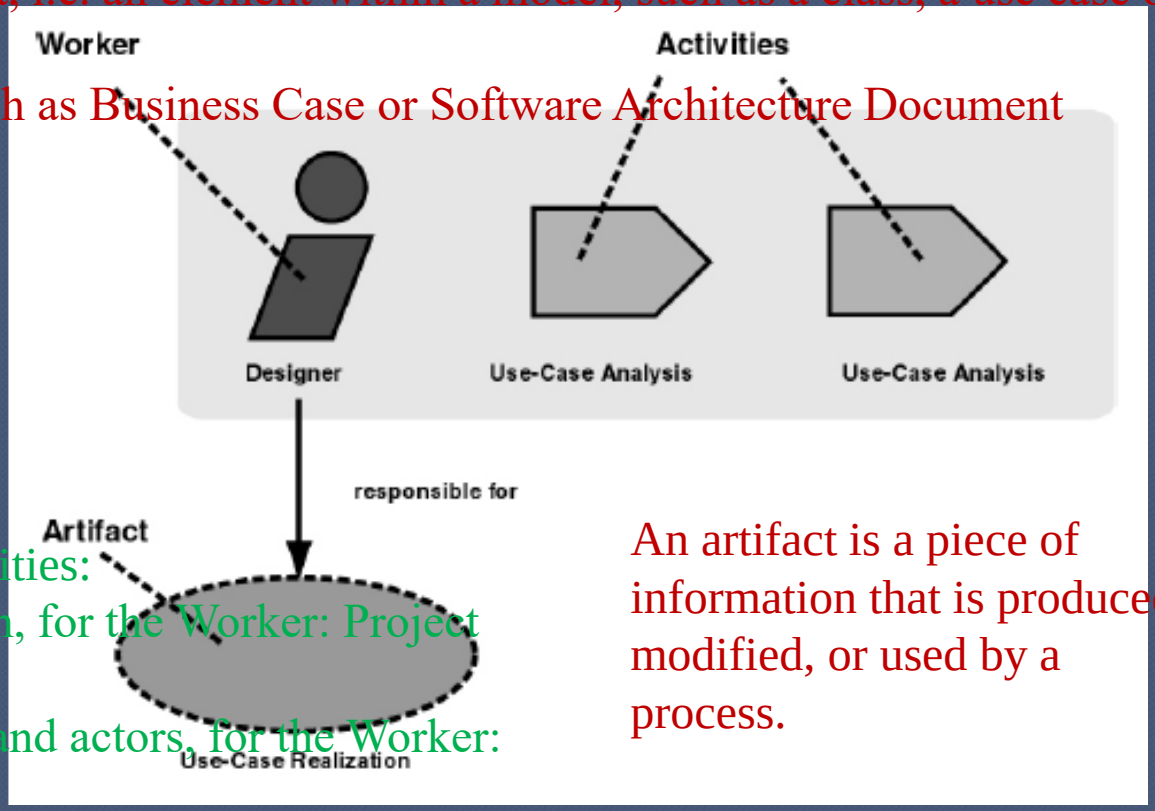
How to define your SDLC



How to define your SDLC

Artifacts may take various shapes or forms:

- A model, such as the Use-Case Model or the Design Model
- A model element, i.e. an element within a model, such as a class, a use case or a subsystem
- A document, such as Business Case or Software Architecture Document
- Source code
- Executables



An artifact is a piece of information that is produced, modified, or used by a process.

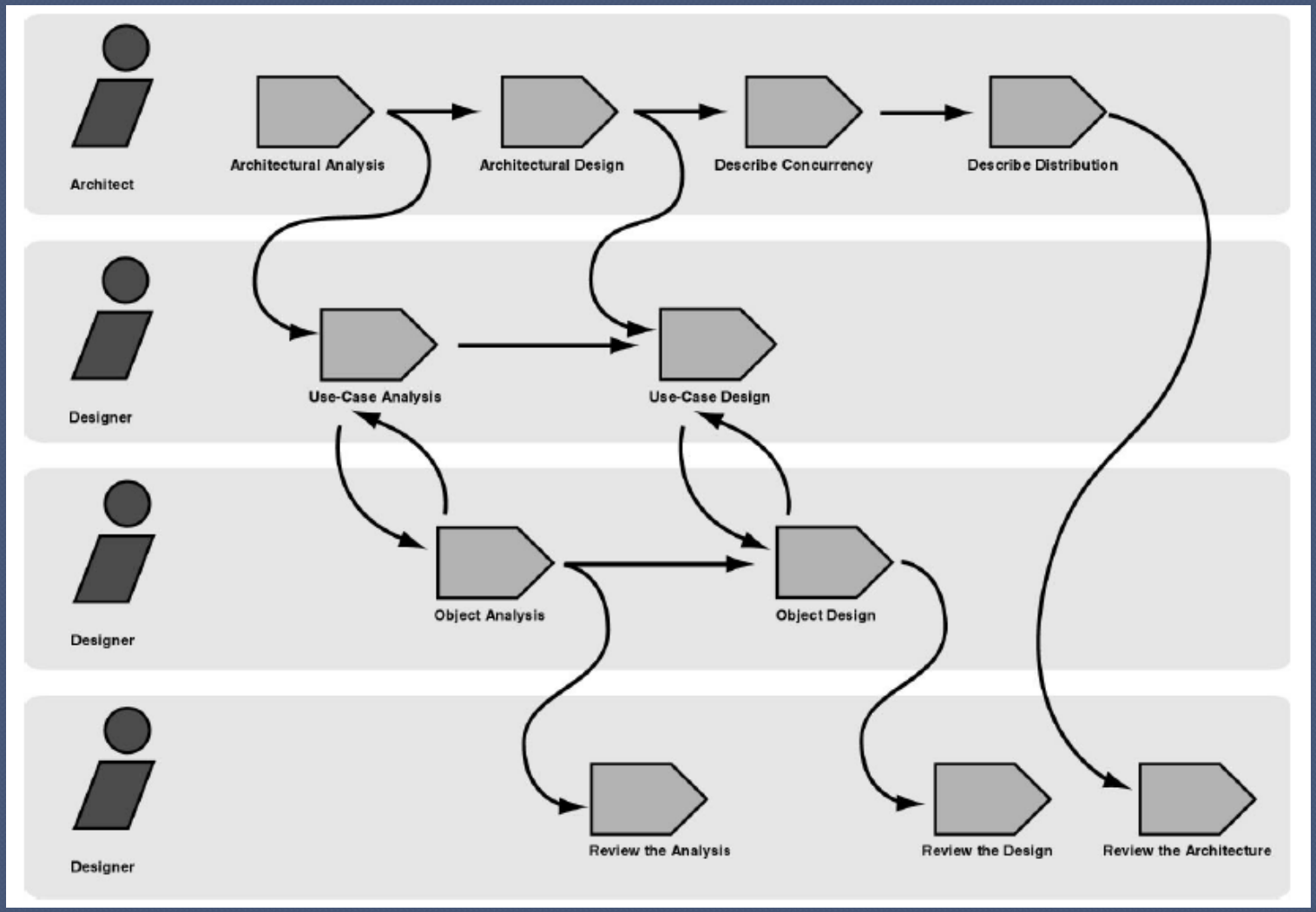
Example of activities:

- Plan an iteration, for the Worker: Project Manager
- Find use cases and actors, for the Worker: System Analyst
- Review the design, for the Worker: Design Reviewer
- Execute performance test, for the Worker: Performance Tester

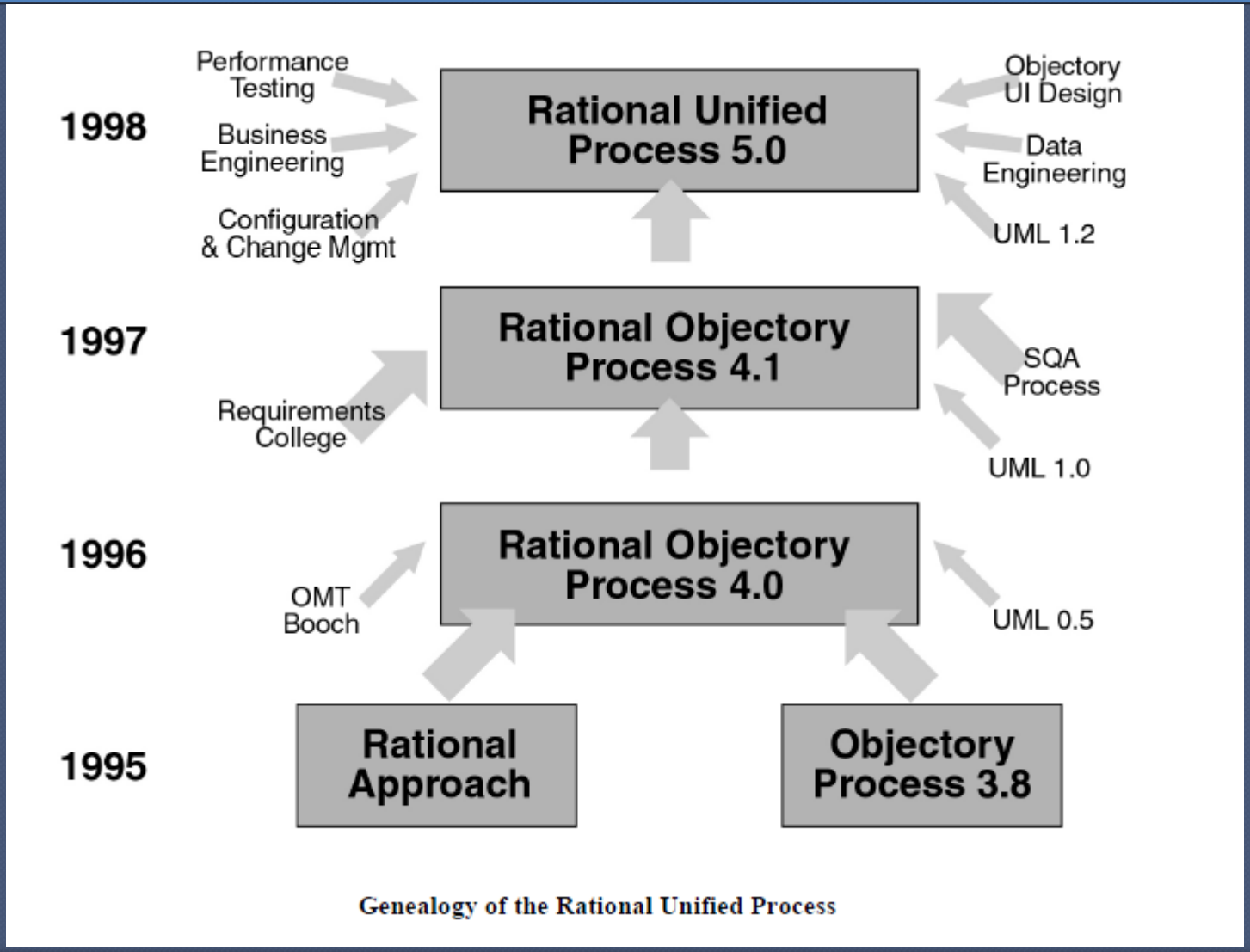
An activity of a specific worker is a unit of work that an individual in that role may be asked to perform.

Workflow

A workflow is a sequence of activities that produces a result of observable value.



Some history



Genealogy of the Rational Unified Process