Chapter 2.

TECHNOLOGIES FOR SW PRODUCTS DEVELOPMENT

3. A Rational Development Process

Exercise #2

1 Microsoft TECHNOLOGY FOR SW PRODUCT DEVELOPMENT

1.1 GENERAL TEMPLATE

1.1.1 Microsoft Presentation:

- World's largest producer of PC software
- Has probably tackled more PC software projects than any other company in the industry.
- Complexity of some of its products, rivals that of many systems for mainframe computers and telecommunication systems.

1.1.2 Microsoft's Philosophy:

- (1) To cultivate its roots as a highly flexible, entrepreneurial company
- (2) To **not** adopt too many of the *structured software-engineering practices* commonly promoted by such organizations as:
  - **Software Engineering Institute (SEI)**
  - **International Standards Organization (ISO)**
- (3) To "scale up" a loosely structured small-team (some might say hacker) style of product development.

1.1.3 MS Objectives:

- (1) To get many small parallel teams (three to eight developers each) or individual programmers to work together as a single relatively large team in order to build large products relatively quickly
- (2) Allowing individual programmers and teams freedom to evolve their designs and operate nearly autonomously.
- (3) The small parallel teams evolve features and whole products incrementally while occasionally introducing new concepts and technologies.
- (4) However, because developers are free to innovate as they go along, they must synchronize their changes frequently so product components all work together.
- **Microsoft:** uses various techniques and melds them into an **Overall Approach** that balances flexibility and structure in software product development.
- The **Overall Approach** is based on a **fundamental paradigm.**

1.2 MS PARADIGM: FREQUENT SYNCHRONIZATIONS AND PERIODIC STABILIZATIONS

1.2.1 MS Approach Basic Paradigm: *"synch-and-stabilize"*

The idea:

- *Continually* synchronize what people are doing as individuals and as members of parallel teams
- *Periodically* stabilize the product in increments as a project proceeds, rather than once at the end of a project.

Terms:

- Microsoft people refer to their techniques variously as the *"milestone," "daily build"*, *"nightly build,"* or *"zero-defect"* process paradigm
- The term *build* refers to the act of putting together partially completed or finished pieces of a software product during the development process, to see what functions work and what problems exist, usually by completely recompiling the source code and executing automated regression tests.

Methodology requirements:

- (1) To develop new, highly complex products
- (2) To use much smaller teams
- (3) To stimulate the inventiveness, creativity and innovation during the development of the product.
- (4) The need to create components that are interdependent. Such components are difficult to define accurately in the early stages of the development cycle
- (5) Encouraging ways that structure and coordinate what the individual members develop
- (6) Encouraging the incremental development
- (7) Allowing developers enough flexibility to be creative and evolve the product's details in stages.
- (8) The development approach must also have a mechanism that allows developers to test the product with customers and refine their designs during the development process.

1.2.2 SW Industry Trends

- Many companies now use new approaches:
  - *Prototyping*
  - *Multiple cycles of concurrent design, build, and test activities to control iterations*
  - *Incremental changes in product development*
- In the software community, researchers and managers talk about:
  - *Iterative enhancement*

    - *Spiral model* for iteration among the phases in project development
    - *Concurrent development* of multiple phases and activities
- However, many companies have been **slow** to formally adopt these recommendations.
- The **basic idea** shared by these approaches is that:
    - Users' *needs* for many types of software are difficult to understand
    - The *changes* in hardware and software technologies are so continuous and rapid
    - It is unwise to attempt to design a software system completely in *advance*
- **Instead**, projects may need to **iterate** while **concurrently managing** many design, build, and testing **cycles** to move forward toward completing a product.

1.2.3 The Waterfall (Sequential) Classical Development Approach

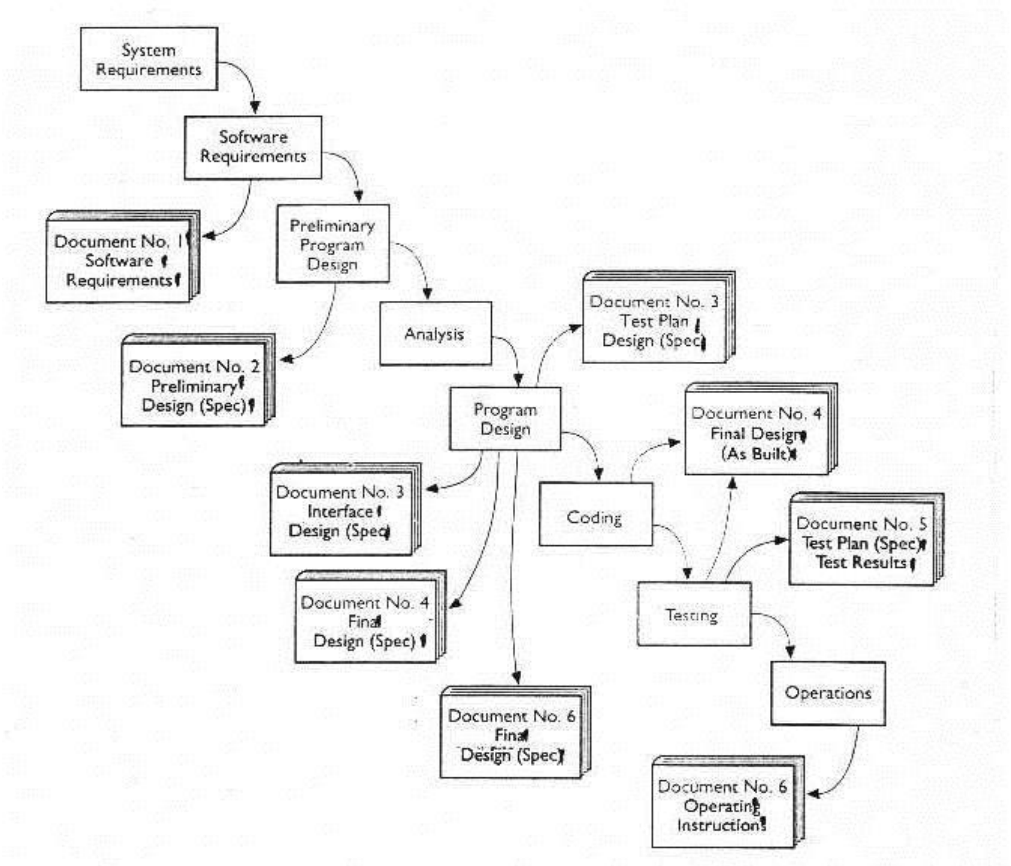

System Requirements
Software Requirements
Document No. 1 Software Requirements
Preliminary Program Design
Analysis
Document No. 2 Preliminary Design (Spec)
Document No. 3 Test Plan Design (Spec)
Program Design
Document No. 4 Final Design (As Built)
Document No. 3 Interface Design (Spec)
Coding
Document No. 5 Test Plan (Spec) Test Results
Document No. 4 Final Design (Spec)
Testing
Operations
Document No. 6 Final Design (Spec)
Document No. 6 Operating Instructions

Fig.1.2.3.a. Waterfall development process model

- **The philosophy**:
    - In the **waterfall** approach:
        - (1) Projects seek to "freeze" a product specification
        - (2) Create a design
        - (3) Build components

        - (4) Merge the components at the end of the project in one large integration and testing phase
    - This approach to software development was common in the 1970s and 1980s
    - It also remains a basic model for project planning in many industries
- **Disadvantages:**
    - The *specifications* and designs can’t be changed
    - There is **not** possible to get *feedback* from customers
    - Is **not** possible to *continually test* components as the products are evolving

1.2.4 Desired Facilities for Development Process

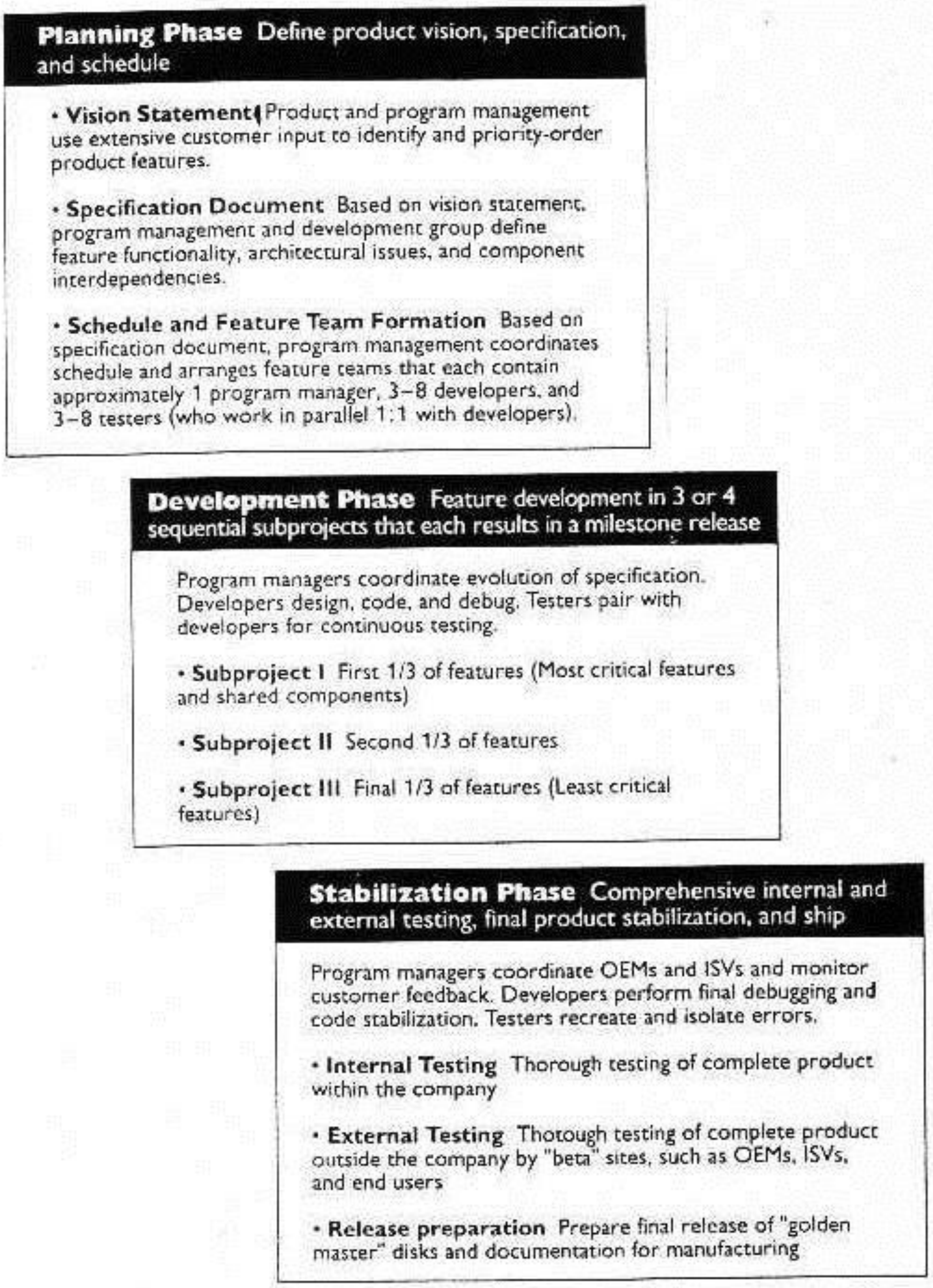
- (1) *Iteration* among design, building components and testing.
- (2) *Overlap* of the development phases.
- (3) *Multiple interactions* with the customers during development.
- (4) *Facilities to ship* preliminary versions of the products.
- (5) *Incrementally adding features or functionalities* over time in various product releases.
- (6) *Frequent integration* of the pieces of the product in order to determine what does and does not work without waiting until the end of the project.

1.3 STRATEGIES AND PRINCIPLES

- To implement its **fundamental paradigm** *“synch-and- stabilize”,* MS has developed:
    - (1) A specific development approach - **Sync-and-Stabilize Development Approach**
    - (2) Two strategies for defining products as well as development processes
        - Each strategy consists in sets of principles that seem critical to making the *synch-and-stabilize style* of product development.
        - These **strategies** are:
            - (1) Defining Product and Organize the Development Process
            - (2) Developing and Shipping Products

1.3.1 Overview of the Synch-and-Stabilize Development Approach

- Product development process is based on **Phase Model** and consists in 3 phases



Planning Phase Define product vision, specification, and schedule

- **Vision Statement** Product and program management use extensive customer input to identify and priority-order product features.
- **Specification Document** Based on vision statement, program management and development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation** Based on specification document, program management coordinates schedule and arranges feature teams that each contain approximately 1 program manager, 3–8 developers, and 3–8 testers (who work in parallel 1:1 with developers).

Development Phase Feature development in 3 or 4 sequential subprojects that each results in a milestone release

Program managers coordinate evolution of specification. Developers design, code, and debug. Testers pair with developers for continuous testing.

- **Subproject I** First 1/3 of features (Most critical features and shared components)
- **Subproject II** Second 1/3 of features
- **Subproject III** Final 1/3 of features (Least critical features)

Stabilization Phase Comprehensive internal and external testing, final product stabilization, and ship

Program managers coordinate OEMs and ISVs and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.

- **Internal Testing** Thorough testing of complete product within the company
- **External Testing** Thorough testing of complete product outside the company by "beta" sites, such as OEMs, ISVs, and end users
- **Release preparation** Prepare final release of "golden master" disks and documentation for manufacturing

Fig.1.3.1.a. Overview of the Sync-and Stabilize Development Approach

- (1) **Planning Phase**
  - (1.1)The process of product development begins by creating a ***"Vision Statement"*** document
    - The ***Vision Statement***
      - Defines the *goals* for the new product

      - Orders the *user activities* that need to be supported by the product features
    - Product managers (marketing specialists) take charge of this task while consulting program managers who specialize in writing up functional specifications of the product
    - Product and program management use extensive customer input to **identify** and **priority-order** product **features**
  - (1.2) Based on ***Vision Statement***, program management and development group elaborate the ***Specification Document***
    - The ***Specification Document*** defines for **each functionality**:
      - (1) *Feature functionality*
      - (2) *Architectural issues*
      - (3) *Component interdependencies*
    - Out-lines the product features in sufficient depth to organize schedules and staffing allocations
    - The initial specification document does **not** try to cover all the details of each feature or lock the project into the original set of features
    - During product development, the team members revise the feature set and feature details as they learn more about what should be in the product.
    - Experience at Microsoft suggests that the **feature set** in a ***Specification Document*** may change by *30%* or more.
  - (1.3) **Schedule** and **Feature Team** Formation
    - Based on specification document, program management coordinates schedule and arranges **feature teams**
    - Each **team** contain approximately:
      - 1 program manager
      - 3—8 developers
      - 3—8 testers who work in parallel 1:1 with developers
- (2) **Development Phase**
  - The project managers then divide the product and the project into **parts** assignable to *features and small features teams*
  - The project schedule is divided into three or four **milestone** junctures (sequential subprojects) representing **completion points** for major portions of the product
    - **Subproject I** *(Milestone 1)* - First 1/3 of features (Most critical features and shared components)
    - **Subproject II** *(Milestone 2)* - Second 1/3 of features
    - **Subproject III** *(Milestone 3)*- Final 1/3 of features (Least critical)
  - Program managers coordinate evolution of specification
  - Developers design, code, and debug
  - Testers pair with developers for continuous testing.

- All the feature teams go through a **complete cycle of development**, feature integration, testing, and fixing problems in **each milestone subproject**
- Throughout the entire project, the feature teams **synchronize** their work by building the product and by **finding** and **fixing** errors on a *daily and weekly basis*.
- At the **end** of a milestone sub-project, the developers **fix** almost all the errors detected in the evolving product.
    - These **error corrections**:
        - **Stabilize** the product
        - **Enable** the team to have **a clear understanding** of which portions of the product have been completed.
    - The development team may then proceed to the next milestone and, eventually, to the ship date.

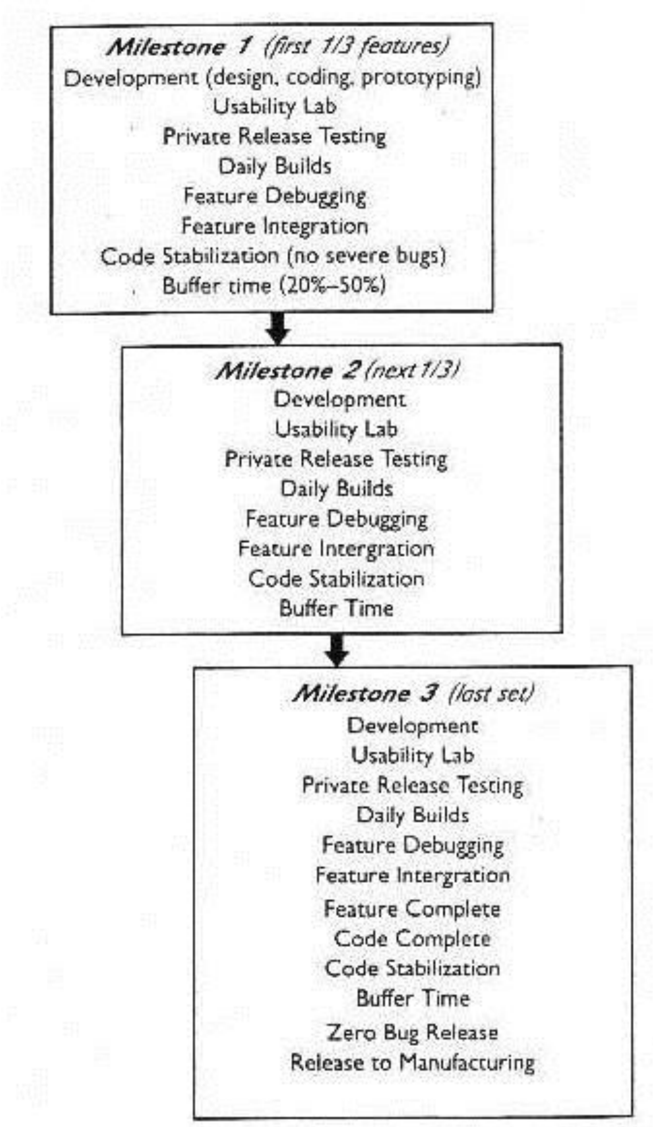


Fig. 1.3.1.b. Milestones in the sync-and-stabilize approach.

- (3) **Stabilization Phase**
    - Comprehensive **internal testing** – consists in testing of complete product within the company
    - **External testing** – consists in testing of complete product outside the company by “beta’ sites, such as OEMs, ISVs and end users
    - Program managers coordinate OEMs and ISVs and monitor customer feedback
    - Developers perform **final debugging** and **code stabilization**
    - Testers recreate and isolate errors
    - **Final** product stabilization
    - Release preparation - Prepare **final release** of “golden master” disks and documentation for manufacturing
    - Product shipment

1.3.2 The first MS Strategy: Defining Product and Organize the Development Process.

- To define products and organize the development process, leading Microsoft product groups follow the ***first strategy*** which can be described as *“focus creativity by evolving features and ‘fixing’ resources.”*
- Teams implement this strategy through ***five** specific **principles**:*
    - (1) Divide large projects into multiple milestone cycles with buffer time (20%—50% of total project time) and no separate product maintenance group. Microsoft gets around these problems by :
        - Structuring projects into *sequential subprojects* containing **priority-ordered features**
        - Establishing *buffers time* within each subproject which gives people time to respond to changes and to unexpected difficulties or delays.
    - (2) Use a “vision statement” and outline feature specifications to guide projects rather than detailed designs and complete product specifications before coding
        - Teams realize they cannot determine in advance everything developers need to do to build a good product.
        - This approach leaves developers and program managers room to **innovate** or **adapt** to changed or unforeseen competitive opportunities and threats.
        - Particularly for applications products:
            - Development teams try to **come up** with features that map directly to activities typical customers perform
            - The teams need to carry out **continual observation** and **testing** with users during development.
    - (3) Base feature selection and priority order on user activities and data.
    - (4) Evolve a modular and horizontal design architecture, mirroring the product structure in the project structure.

        - Most product designs have **modular architectures** allowing teams to **incrementally add or combine features** in a straightforward, predictable manner.
        - Managers allow team members to set their **own schedules**, but only after the developers have analyzed tasks in detail (for example, half-day to three-day chunks) and have been asked to **personally commit to the schedules they set.**
        - Managers then “**fix**” **project resources** by **limiting** the **number** of **people** they allocate to each project.
        - Managers also try to **limit the time** spent on projects, (especially for applications like Office and multimedia products), so teams can **delete** features if they fall too far behind.
        - However, cutting features to save schedule time is **not** always possible with operating systems projects in which:
            - **Reliability** is more important than features
            - Many features are closely coupled and cannot easily be deleted individually.
    - (5) Control by individual commitments to small tasks and “fixed” project resources.

1.3.3 The Second MS Strategy: Developing and Shipping Products.

- To manage the process of developing and shipping products, Microsoft follows the ***second strategy*** which can be described as *“do everything in parallel with frequent synchronization”.*
- Teams implement this strategy by following another *set of five principles*:
    - (1) Work in parallel teams but “synch up” and debug daily.
    - (2) Always have a product you can ship with versions for every major platform and market.
    - (3) Speak a “common language” on a single development site.
    - (4) Continuously test the product as you build it.
    - (5) Use metric data to determine milestone completion and product release.
- **Obs:** These principles bring considerable discipline to the development process **without** the need to control every moment of every developer’s day.

1.4 MS TECHNOLOGY

1.4.1 Objective

- Microsoft tries to allow many **small teams** and **individuals** enough freedom to **work in parallel** yet still function as **a single large team**, so they can build **large-scale products** relatively **quickly** and **cheaply**.

1.4.2 Rules

- The teams adhere to a few rigid rules that enforce a high degree of **coordination** and **communication**.
    - (1) Whatever day the developer decide to **“check in”**, or enter their pieces of code into the product database, they must do so by a **particular time** (say, 2 p.m. or 5 p.m.) This rule **allows**:
        - The project team to assemble available components
        - Then to completely recompile the product source code, and create a **new “build”** of the evolving product by the end of the day or by the next morning
        - Then start testing and debugging immediately.

      This rule is analogous to telling children that they can do whatever they want all day, but they must be in bed at 9 p.m.

    - (2) If developers check in code that **“breaks”** the build by preventing it from completing the recompilation, they **must fix the defect immediately**.

      This rule resembles Toyota’s famous production system, in which factory workers are encouraged to stop the manufacturing lines whenever they notice a defect in a car they are assembling.

1.4.3 Working Manner

- Microsoft’s **daily build process** has several **steps**.
    - (1) First, in order to develop a feature for a product, developers can **check-out** private copies of source code files from a **centralized master version** of the source code
    - (2) The developers **implement their features** by making changes to their **private copies** of the source code files
    - (3) The developers then create a **private build** of the product containing the new feature and test it
    - (4) Then the developers **check-in** the changes from their **private copies** of the source code files to the **master version** of the source code (**branch**)
    - (5) The check-in process includes an **automated regression test** to help ensure that their changes to the source code files do not cause errors elsewhere in the product.
    - (6) Developers usually **check in** their code back to the master copy at least twice a week but may **check it in** daily.
    - (7) Regardless of how often individual developers check in their changes to the source code, a designated developer, called the ***project build master***, generates a ***complete build of the product*** on **a daily basis** using the master version of the source code
    - (8) Generating a **build** for a product consists of executing an **automated sequence of commands** called a ***“build script.”***
        - This daily build creates a new ***internal release*** of the product and includes many steps that compile source code.
        - The build process also automatically translates the source code for a product into one or more executable files and may create

        various library files, allowing end users to customize the product.

        - The **new internal release** of the product built each day is the ***daily build***.

    - (9) **Daily builds** are generated for each **platform**, such as Windows and Macintosh, and for each market, such as the U.S., and for the major international versions.
    - (10) Product teams also **test features** as they build them from multiple perspectives, including bringing in customers “off the street” to try prototypes in a Microsoft usability lab.

1.4.4 MS Teams

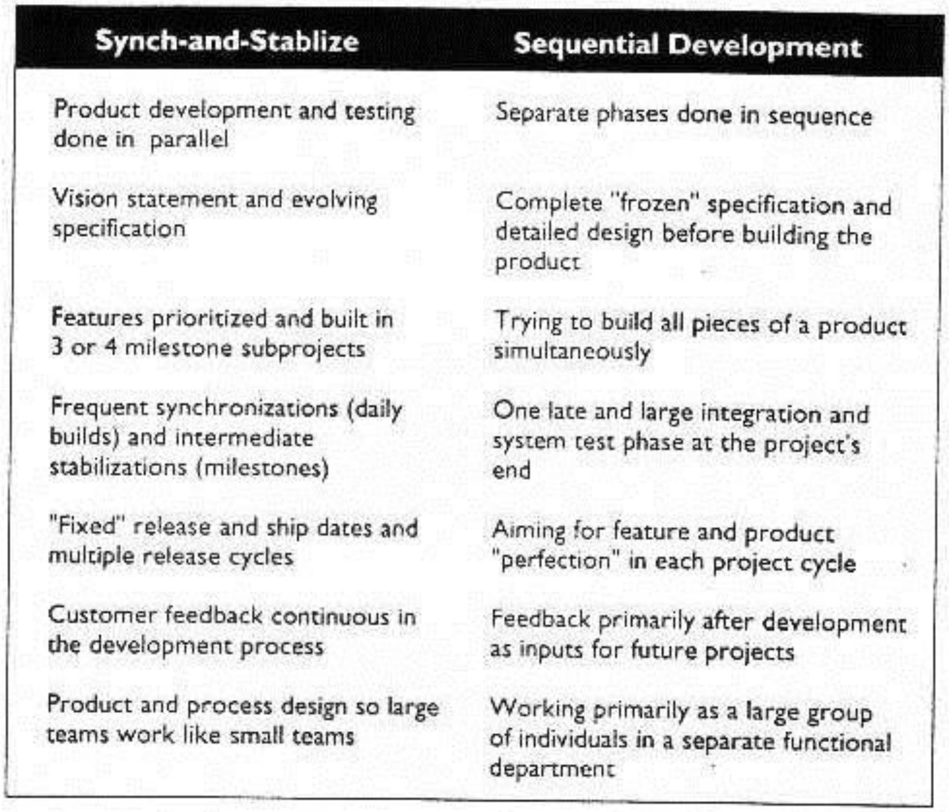
- Nearly all **Microsoft teams** work at a single physical site
- Utilize a common development language (primarily C and C++),
- Utilize a common coding style
- Utilize standardized development tools.
    - A common site and common language and tools help teams communicate, debate design ideas, and resolve problems face to face.
- Project teams also use a small set of quantitative metrics to guide decisions, such as when to move forward in a project and when to ship a product to market.
    - For **example**, *managers rigorously track progress of the daily builds by monitoring how many bugs are newly opened, resolved (such as by eliminating duplicates or deferring fixes), fixed, and active.*
- **Obs:** Some people may argue that Microsoft’s key practices in product development—daily synchronization through product builds, periodic milestone stabilizations, and continual testing—are no more than process and technical fixes for a **hacker software organization** now building huge software systems.

1.5 Conclusions

1.5.1 MS Innovations

- MS encourages some teams to experiment and make lots of changes without much up-front planning.
- Projects generally remain under control because teams of programmers and testers frequently synchronize and periodically stabilize their changes.
- Microsoft resembles companies from many industries that do incremental or iterative product development as well as concurrent engineering.
- MS has also **adapted** software-engineering practices introduced earlier by other companies (such as various testing techniques)
- MS has **reinvented** the wheel on many occasions (such as concluding that accumulating historical metric data is useful for analyzing bug trends and establishing realistic project schedules)

- MS has introduced a structured hacker-like approach to software product development that works reasonably well for both small- and large-scale products (Excel, Office, Publisher, Windows 95,98,2000, Windows NT, Windows XP, Windows Vista, Word, Works, and other)
- Microsoft is a fascinating example of how culture and competitive strategy can drive product development and the innovation process.
- The Microsoft culture centers around fervently antibureaucratic PC programmers who do not like a lot of rules, structure, or planning.
- The principles behind the **synch-and-stabilize philosophy** add a semblance of order to the fast-moving, often chaotic world of PC software development.
- There are no silver bullets here that solve major problems with a single simple solution. Rather, there are:
  - Specific approaches, tools, and techniques;
  - A few rigid rules;
  - Highly skilled people whose culture aligns with this approach.
- Several elements distinguish synch-and-stabilize from older, more traditional sequential and rigid styles of product development



Synch-and-Stablize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at the project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department

Fig. 1.5.1.a. Sync-and-stabilize vs. sequential development

1.5.2 MS Competitive Strategies

- Identifying mass markets quickly
- Introducing products that are “good enough” (rather than waiting until something is “perfect”)
- Improving these products by incrementally evolving their features
- Selling multiple product versions and upgrades to customers around the world.

1.5.3 Microsoft Development Approach Weaknesses

- MS now needs to pay more attention to product architectures, to defect prevention mechanisms.
- Insufficient involvement in conventional engineering practices, such as more formal design and code reviews.
- New product areas also pose new challenges for its development methods not always successfully solved.
  - For **example**, some new areas, such as video on demand, have many tightly linked components with real-time constraints that require precise mathematical models of when video/audio/user data can be delivered reliably and on time.
- Many existing and new products have an extremely large or even infinite number of potential user conditions or scenarios to test based on what hardware and applications each customer is using.
  - These new products can benefit from some incremental changes in the development process.
  - They also require more advance planning and product architectural design than Microsoft usually does to minimize problems in development, testing, and operation.

1.5.4 MS Technology Advantages

- It breaks down **large products** into manageable pieces (a priority-ordered set of product features that small feature teams can create in a few months).
- It enables projects to proceed systematically even when they cannot determine a complete and stable product design at the project’s beginning.
- It allows large teams to work like small teams by:
  - Dividing work into pieces
  - Proceeding in parallel but synchronizing continuously
  - Stabilizing in increments
  - Continuously finding and fixing problems.
- It facilitates competition on customer input, product features, and short development times by:

  - Providing a mechanism for incorporating customer inputs,
  - Setting priorities,
  - Completing the most important parts first,
  - Changing or cutting less important features.
- It allows a product team to be very responsive to events in the marketplace by:
  - "Always" having a product ready to ship,
  - Having an accurate assessment of which features are completed,
  - Preserving process-product flexibility and opportunism throughout the Development process.
- The Microsoft ideas and examples provide useful lessons for organizations and managers in many industries**.**
- The synch-and-stabilize approach used at Microsoft is especially suited to fast-paced markets with:
  - Complex systems products
  - Short lifecycles
  - Competition based around evolving product features and de facto technical standards.
- Coordinating the work of a large team building, many interdependent components that are continually changing requires a constant and high level of communication and coordination.
- It is **difficult** to ensure that such communication and coordination take place while still allowing designers, engineers, and marketing people the freedom to be **creative**.
  - Achieving this balance is perhaps the **central dilemma** that managers of product development face in **PC software** as well as in many **other industries**.

2 ORACLE TEHNOLOGY FOR SW PRODUCTS DEVELOPMENT

2.1 PREZENT PROBLEMS OF THE IT INDUSTRY

2.1.1 IT Industry problems

- IT industry hasn't a very good reputation. In SW production (2014):
  - Estimated costs were exceeded on average with 189%
  - Assumed deadlines were exceeded on average with 222%
  - In average are delivered only 61% from the initial agreed features
  - Only 12% of SW projects are delivered in time and don't exceed the established budget
  - 25% of projects are canceled before to be finalized
  - 75% of the big projects were considered by their users as operational failures

2.1.2 Causes emphasized by ORACLE experience

- Lack of a real implication of the company upper management
- Delayed decisions, and even lack of decision
- Unrealistic plans
- Non-experienced project manager or not committed 100% from time and effort point of view
- Non-identification of risks and lack of appropriate action plans
- Weakly financial monitoring
- Lack of properly qualified people
- Lack of a good quality plan

2.1.3 CONCLUSION

- In the most cases, the causes of failures in SW industry are the **management deficiencies**

2.2 ORACLE SET OF DEVELOPING METHODOLOGIES

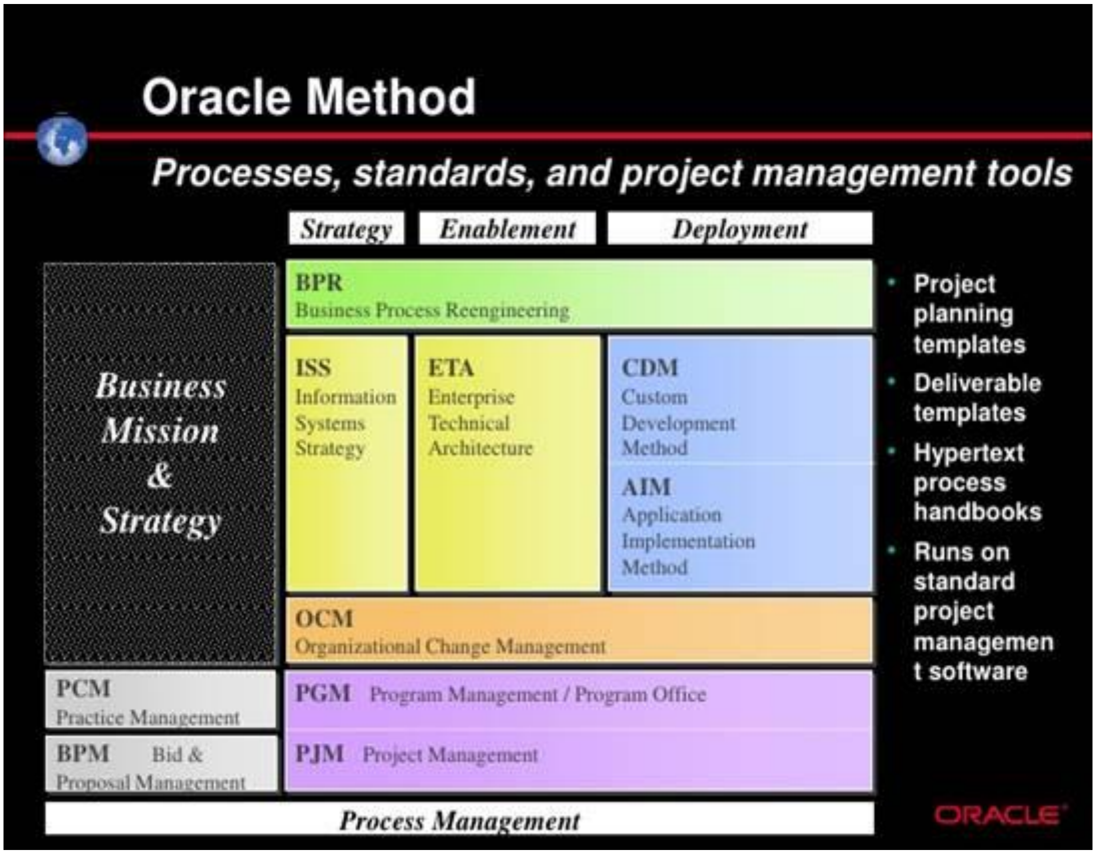
- **ORACLE tradition and experience** accumulated in the process of development of large size projects imposed the necessity of elaboration of a ***development methodology***

2.2.1 Fundamental idea

- **Idea:** *construction of a partnership environment between client and developer*

- **Result:** a set of developing methodologies oriented to several functional or business areas dedicated to project leaders

2.2.2 ORACLE Set of developing methodologies



Oracle Method

Processes, standards, and project management tools

Strategy *Enablement* *Deployment*

Business Mission & Strategy

BPR Business Process Reengineering

ISS Information Systems Strategy

ETA Enterprise Technical Architecture

CDM Custom Development Method

AIM Application Implementation Method

OCM Organizational Change Management

PCM Practice Management

BPM Bid & Proposal Management

PGM Program Management / Program Office

PJM Project Management

Process Management

- Project planning templates
- Deliverable templates
- Hypertext process handbooks
- Runs on standard project managemen t software

ORACLE

Fig. 2.2.2.a. The set of ORACLE Methodologies

- **Business Processes Re-engineering** is the main method which aligns an organization with its strategy in order to put into practice its vision and to obtain benefits
- **Technological Solutions** cover domains as *technological strategy, technical architecture, SW maintenance, and data administration.*
    - Among Technological Solutions play an important role:
        - **ISS – Information Systems Strategy**
        - **ETA – Enterprise Technical Architecture**
- For **Applications Development** two distinct methodologies are used in dependence of the client nature:
    - **AIM – Applications Implementation Method** centered on carrying out projects starting from **ORACLE Applications**
    - **CDM – Custom Development Method** which structures projects development based on applications built by means of **ORACLE tools** *(Oracle Designer, Oracle Programmer, Oracle Developer)*
- **Preparation of organization** refers to *projects management* and to *organization change management*
    - **OCM – Organizational Change Management** prepares organization to accept the change associated with carrying out SW projects.
        - The success of carrying out valuable SW projects is in **direct connection** with organization capacity to adapt oneself to new situations and to accept the change

- **Management** - there are different types of specific management
    - **PCM – Practice Management**
    - **BPM – Bid & Proposal Management**
    - **PGM – Program Management**
    - **PJM – Project Management**

2.3 PROJECT MANAGER RESPONSIBILITIES

2.3.1 Project manager responsibilities

- (1) Represents in the same time the client and the consultant management
- (2) Is considered responsible for satisfying the requirements of the both involved parts with respect to project expectances
- (3) Has to understand business objectives and to conceive a clear vision to rich them
- (4) Has to be early implied in relation with the client, preferably before the signing of contract
- (5) Is responsible in front of resource suppliers from consulting organization, for efficient utilization of resources
- (6) Is responsible in front of client for reaching the assumed objectives in project framework

2.3.2 Major tasks of the Project manager:

- (1) To make correct decisions under pressure of risks, incertitude, and a large amount of information potential relevant
- (2) Carrying out project tasks usually under unfavorable circumstances:
    - Different and heterogeneous staff
    - Usually, low authority over the people
    - Limited control over staff

2.4 ORACLE PROJECT PHASES

- ORACLE uses three **phases** in development process
    - (1) **Planning**
    - (2) **Execution**
    - (3) **Finalizing**

2.4.1 Planning

- Is the first development phase. Consists in:
- (1) Up-dating *amplitude of the project* in order to reflect the changes imposed by negotiation and signing of the contract
- (2) Developing a *detailed plan for execution phase*
- (3) Obtaining the *client commitment* concerning his responsibilities and resources that he will supply
- (4) Obtaining *the approvals* to pass to the execution phase from client management and from consulting management
- (5) Identifying the eventually *infrastructure changes* necessary to support the execution phase
- (6) *Allocation and preparation of human resources* for project development

2.4.2 Execution

- (1) *Targeting and tailoring the effort* in accordance with project magnitude, imposed quality and provided costs
- (2) *Anticipating the possible risks* and preparing the measures to prevent or to correct them
- (3) *Solve with efficiency the current problems* which occur
- (4) *Assuring the integrity and consistency of the deliverables* in accordance with project objectives

2.4.3 Finalizing

- (1) Obtaining the *client acceptance* for the delivered product
- (2) Successfully finalizing the *contractual relationship*
- (3) *Transferring the documentation and the production environment* to the client
- (4) *De-allocation of human and physical used resources*
- (5) *Results documenting and archiving*

2.5 ORACLE PROJECT PROCESSES

2.5.1 The PROCESS definition. TYPE OF PROCESSES

- **The Process:** - is a set of related activities, with a high degrees of interdependence, having as result one or more critical deliverables for the project
- **Types of processes:**
    - (1) **Control and Report**

  - (2) **Labor Management**
  - (3) **Resources Management**
  - (4) **Quality Management**
  - (5) **Configuration Management**

2.5.2 The Processes in ORACLE technology

- (1) **Control and Report Process** – handles *the amplitude of the project* as well as *the relations with the client*. For this purpose utilizes:
    - Procedures for *control of modifications (changes)*
    - Procedures for *management of risks*
    - Procedures for *management of occurred problems*
    - Procedures for *report* concerning *project evolution*
- (2) **Labor Management** – establishes a consistent ***Work Plan*** (approved by the customer) which details:
    - The amplitude of the project
    - The objectives
    - The project approach manner
  - ***The Work Plan*** is used by the manager to:
    - *Monitor*
    - *Control*
    - *Report*
    - *Re-plan the project execution*
- (3) **Resources Management** – is responsible for supplying *physical and human resources* for the project.
    - Is tightly interconnected with Labor Management Process
- (4) **Quality Management** – establishes a ***Quality Plan*** which defines standards and procedures for ***assurance*** *of consistency and coherency* of the *quality audits*, for *deliverables* and for *project team's members*

  Obs. *Testing activity is* ***not*** *part of the Quality Management Process, though they are interrelated, but is a separate process included in* **CDM** *or* **AIM**

- (5) **Configuration Management** – refers to *identification, control* and *monitoring* of any item produced during project development.
    - In the frame of this process, *the deliverables* are assembled step by step in a *unitary product.*
    - This process must assure the fact that only *the changes approved* by customer and by consulting management will affect the final product.

2.5.3 General remarks

- The processes acts in *different phases of the project*
- *The project manager* is the person who establish the *level of responsibility and authority* for each member of developing team
- The *project manager* establishes *the set of rules and standards* to be followed during the process of development

2.6 CONCLUSIONS

ORACLE

- Uses *practical and clear methods*
- Stress on the construction of a *partnership environment between client and developer* as fundament of the success
- Uses *experimented teams* which develop projects having all the time in mind, *the project objectives* and *the benefits of the developing organization*

3 A RATIONAL Development process

3.1 The Rational Way

- The "Rational Way" is:
  - (1) Iterative and incremental
  - (2) Object-oriented
  - (3) Managed and controlled
  - (4) Highly automated
- It is generic enough to be tailorable to a wide variety of software products and projects, both in size and application domain.
- It is centered around three **poles**:
  - (1) People
  - (2) Process
  - (3) Tools and methods

3.2 The Overall Software Lifecycle

3.2.1 Two Perspectives

- The Rational process may be approached from two different and integrated **perspectives** (Fig. 3.2.1.a):
  - (1) A *management perspective*, dealing with the *financial*, *strategic*, *commercial*, and *human aspects*
  - (2) A *technical perspective*, dealing with *quality*, *engineering* and *design method aspects*

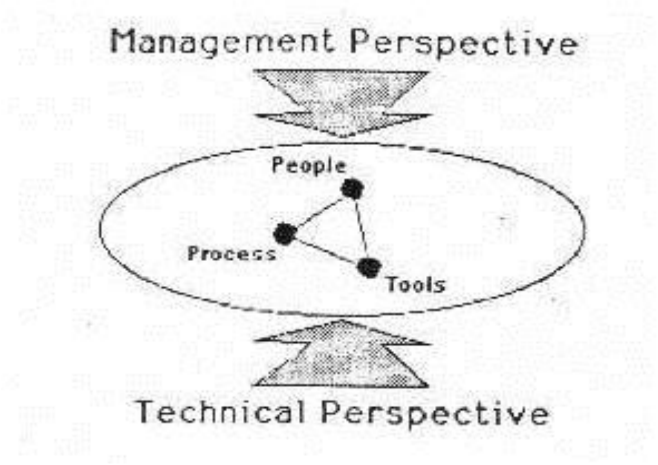


Fig. 3.2.1.a. Management and technical perspective in Rational Process

3.2.2 Management Perspective: Cycles and Phases

- From a **management perspective**, the ***software lifecycle*** is organized along **4 main phases** (Fig. 3.2.2.a):
  - (1) **Inception**:
    - Starts with a good idea
    - Specifying the end-product vision
    - Specifying its business case
    - Defining the scope of the project
  - (2) **Elaboration**:
    - Planning the necessary activities
    - Planning the required resources
    - Specifying the features
    - Designing the architecture
  - (3) **Construction**:
    - Building the product
    - Evolving the vision, the architecture, and the plans until the product *(the completed vision)* is ready for transfer to its users' community
  - (4) **Transition**:
    - Making the transition from the product to its user's community
    - Manufacturing, delivering, training, supporting, maintaining the product until the users are satisfied

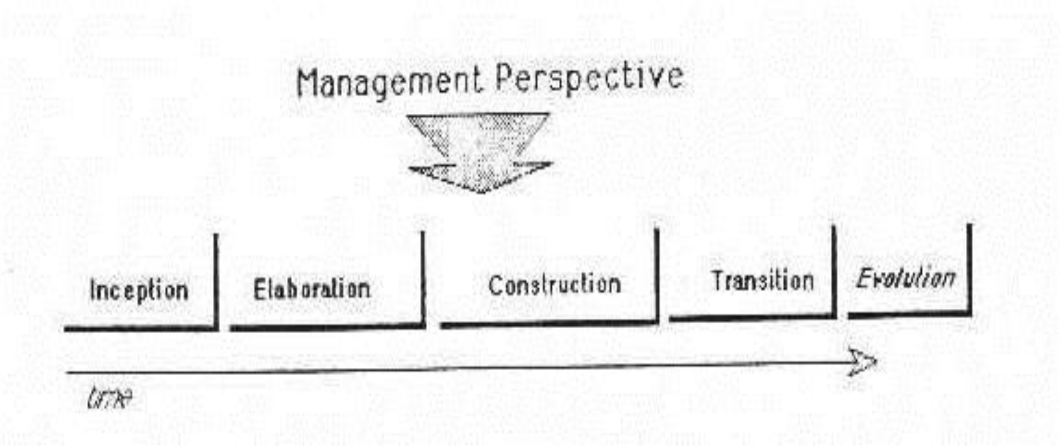


Fig.3.2.2.a. Management perspective

- Going through the 4 phases is called a *development cycle*, and it produces a *software generation* (Fig. 3.2.2.b)
- An existing product can evolve into its *next generation* by repeating the same sequence of phases, with a different emphasis however on the various phases.
    - This period is called *evolution cycle.*
- As the product eventually goes through several cycles, *new generations* are being produced.
- *Evolution cycles* may be triggered by:
    - User suggested *enhancements*
    - *Changes* in the *users' context*
    - *Changes* in the underlying *technology*
    - *Reaction* to the *competition*, etc.
- In practice, cycles may slightly overlap:
    - The inception and elaboration phase may start during the transition phase of the previous cycle.

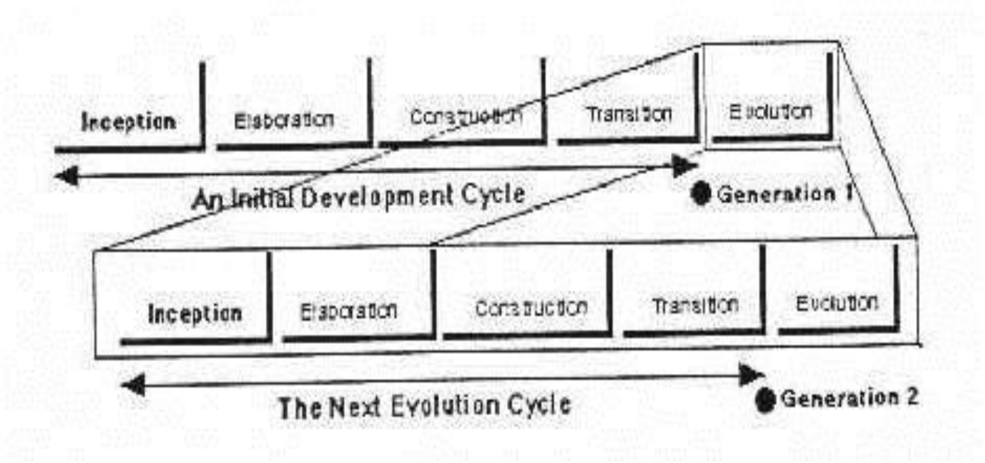


Fig. 3.2.2.b. Generations and evolution cycles

3.2.3 Technical Perspective: Iterations

- From a **technical perspective** the ***software development*** is seen as a succession of *iterations*, through which the software under development evolves *incrementally*. (Fig. 3.2.3.a).
- Each **iteration** is concluded by the *release* of an executable product (which may be a *subset* of the complete vision, but useful from some engineering or user perspective)
- Each **release** is accompanied by *supporting artifacts*: *plans*, *release description, user's documentation, etc*.
- An *iteration* consists of the activities of planning, analysis, design, implementation, and testing in various proportions depending on where the iteration is located in the development cycle.

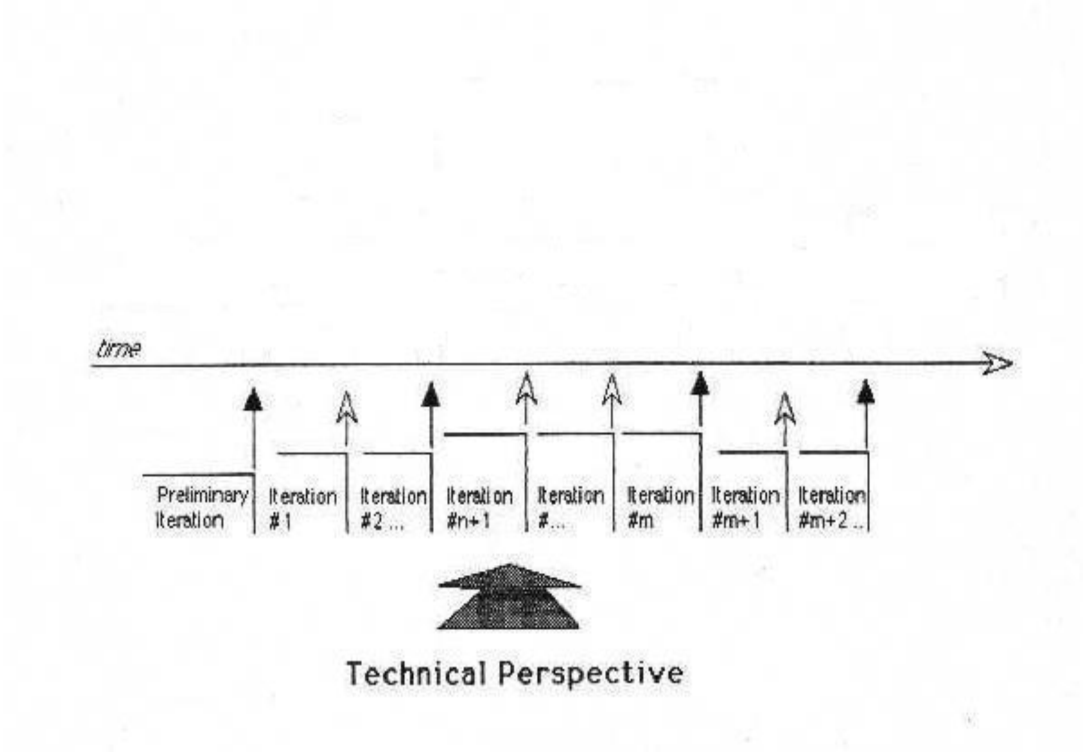


Fig.3.2.3.a. Technical Perspective

3.2.4. Reconciliation of the two perspectives

- The **management perspective** and the **technical perspective** are reconciled.
    - In particular *the end of the **phases*** are synchronized with *the end of **iterations**.*
    - In other words, each *phase* is broken down into one or more *iterations*. (Fig. 3.2.4.a).
- The two perspectives--management and technical--synchronize in fact on a few well identified ***milestones****.*

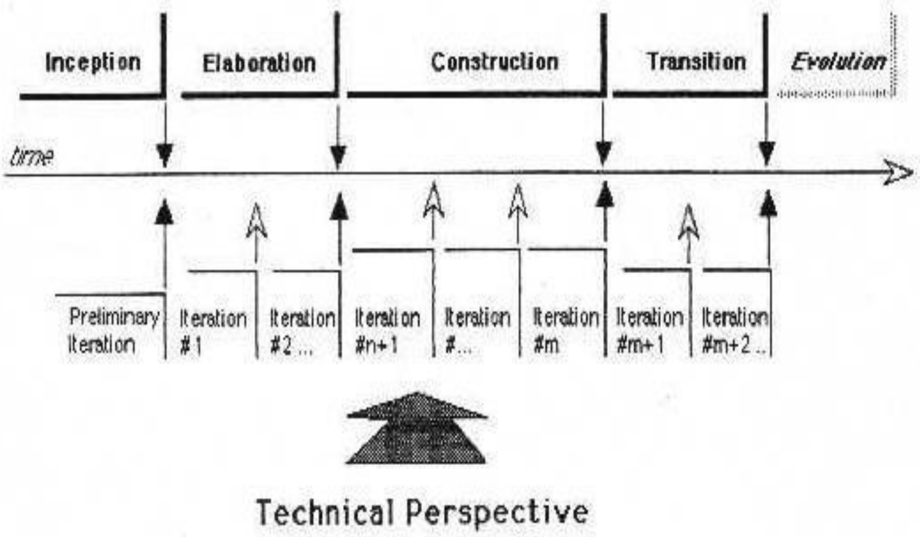


Fig.3.2.4.a. Reconciliation of the two perspectives

- The both perspectives contribute to a common *set of products* and *artifacts* that evolve over time.
    - Some artifacts are more under the control of the **technical side**, some more under control of the **management side** (Fig. 3.2.4.b.)

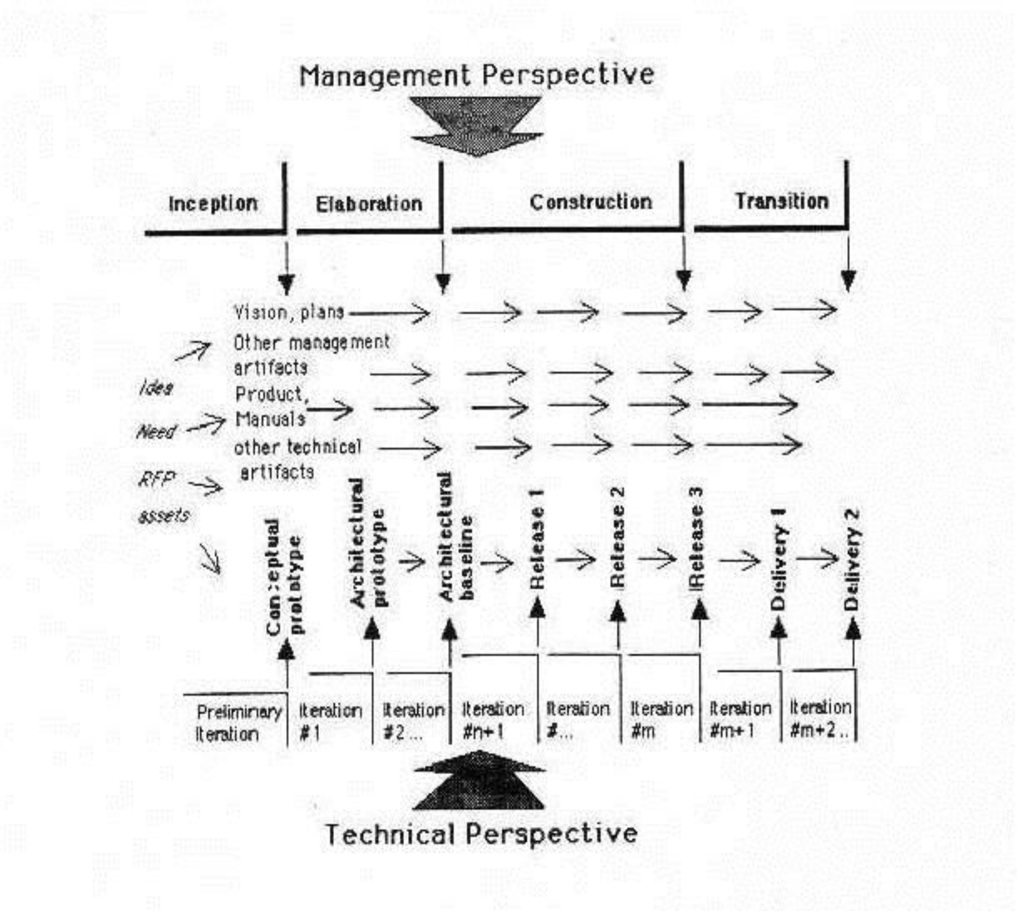


Fig.3.2.4.b. Artifacts in the project life cycle

- The **tangible elements** that constitute the ***milestones***, much more than mere dates on a calendar are
  - (1)*The availability of the **artifacts*** and
  - (2) The *satisfaction of the established evaluation criteria* for the ***product*** and the ***artifacts***
- Like cycles, iterations may slightly *overlap*,
  - E.g., the planning or architecture activities of iteration N may be started toward the end of iteration N-1.
  - In some cases, some iterations may proceed *in parallel.*

3.2.5. Discriminants

- The development process is influenced by the following factors:
  - (1) The emphasis and importance of the various phases
  - (2) The entry and exit criteria
  - (3) The artifacts involved along a development cycle
  - (4) The number and length of the iterations
- All this **factors** may vary depending on ***four** major project characteristics* that are named ***process discriminants***.
  - In order of decreasing impact, the most important process discriminants are:

- (1) The **business context.** There are three types business context:
  - (a) Contract work, where the developer produces the software on a *given customer specification* and for this customer only
  - (b) Speculative or commercial development, where the developer produces software to be put on the *market*
  - (c) Internal project, where the customer and developer are in the same organization
- (2) The **size** of the software development efforts defined by some metrics:
  - Delivered Source Instructions
  - Functions points, etc., or
  - Number of person-months
  - Cost
- (3) The **degree of novelty**--how "precedented" this software effort is *relative to the development organization*, and in particular whether the development is in a *second or subsequent* cycle.
  - This discriminant includes:
    - The maturity of the organization and the process
    - Its assets (active)
    - Its current skill set
    - The issues such as assembling and training a team, acquiring tools and other resources.
- (4) The **type of application**, presume the target ***domain***:
    - MIS – (Management Information System),
    - Command and control,
    - Embedded real-time,
    - Software development environment tools, etc.
  - Especially with respect to the *specific constraints* the domain may impose on the development:
    - *Safety,*
    - *Performance,*
    - *Internationalization,*
    - *Memory constraint, etc.*
- This chapter:
    - (1) First describes the *generic process,* i.e., the part of the process that applies to all kinds of software developments, across a broad and general range of these **discriminants**.

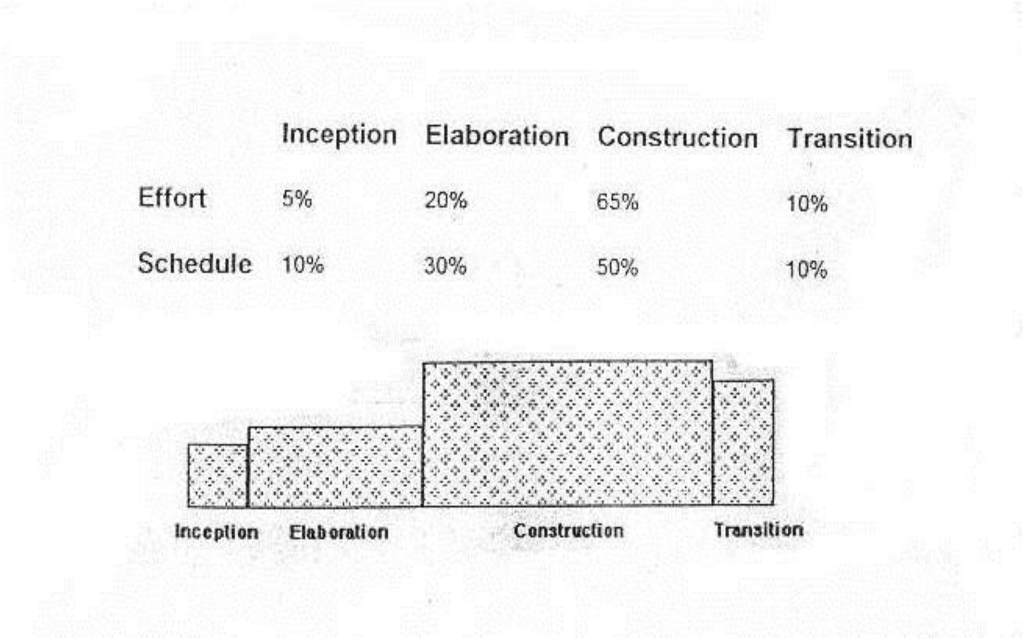
    - (2) It then describes some *specific instances* of the process for some values of the discriminants, as examples.

3.2.6 Effort and Schedule

- The SW project development phases are *not identical* in terms of schedule and effort.
    - Although this will vary considerably depending on the project discriminants,
- ***A typical initial development cycle*** for a *medium size project* should anticipate the following ratios:

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

- This can be depicted graphically as in Fig. 3.2.6.a.



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

Fig. 3.2.6.a. Effort and schedule in a typical initial development cycle

- For an **evolution cycle**, the inception and elaboration phases can be considerably reduced.
- Also, using certain **tools and techniques**, such as applications builders, the construction phase can be much smaller than the inception and elaboration phase together.

3.3 The Phases of the Rational Process

3.3.1 Inception Phase

- ***Inception Phase Objectives:***
  - (1) To light *an original vision* of a potential product, and transforms it into *an actual project.*
  - (2) To establish the *business case* for a new product or a major update
  - (3) To specify the *project scope.*

- ***Inception Phase Outcomes:***
  - (1) For the development of **a new product**, the main outcome of this phase is a *"go-no go" decision* to move into the next phase and to invest time and money to analyze in detail what is to be built, can it be built, and how to build it.
  - (2) For the evolution of **an existing product**, this may be *a simple and short phase*, based on users' or customers' requests, on problem reports, and on new technological advances.
  - (3) For **a contractual development**, the decision to proceed is based on experience of the specific domain and on the competitiveness of the development organization in this domain or market.
    - In this case the inception phase may be concluded by a *decision to bid,* or by the *bid itself*.
    - The idea may be based on an *existing research prototype*, whose architecture may or may not be suitable for the final software.

- ***Inception Phase Entry Criteria:***
  - (1) An *original vision*
  - (2) A legacy system
  - (3) An RFP (Request For Proposal)
  - (4) The *previous generation* and a *list of enhancements*
  - (5) Some *assets* (software, know-how, financial assets)
  - (6) A *conceptual prototype*, or a *mock-up*

- ***Inception Phase Exit Criteria:***
  - (1) An *initial **business case*** containing at least:
    - A *clear formulation of the product vision*--the core requirements-- in terms of functionality, scope, performance, capacity, technology base
    - *Success criteria* (for instance revenue projection)
    - An *initial risk assessment*

        - An *estimate of the resources* required to complete the elaboration phase

- ***Optionally*** at the end of the *inception phase*, we may have:
    - (2) An *initial* ***domain analysis model*** **(**~10%-20% complete), identifying the top key use cases, and sufficient to drive the architecture effort
    - (3) An *initial* ***architectural prototype*****,** which at this stage may be a throw-away prototype

3.3.2 Elaboration Phase

- ***The Elaboration Phase Main Goal:***
    - (1) To more thoroughly *analyze the problem domain*
    - (2) To *define and stabilize the architecture*
    - (3) To address the *highest risk elements* of the project
    - (4) To produce a *comprehensive plan* showing how the *2 next phases* will be done. The plan contains:
        - A *baseline product vision* (i.e., an initial set of requirements) based on an analysis model
        - *Evaluation criteria* for at least the first construction iteration
        - A *baseline software architecture*
        - The *resources* necessary to develop and deploy the product, especially in terms of people and tools
        - A *schedule*
        - A *resolution of the risks* sufficient to make a "high fidelity" cost, schedule and quality estimate of the construction phase

- ***The Elaboration Phase Outcomes:***
    - (1) An executable ***architectural prototype*** which is built in one or several *iterations* depending on the scope, size, risk, novelty of the project, which addresses:
        - At least *the top key use cases* identified in the inception phase
        - *The top technical risks* of the project.
    - (2) An *evolutionary prototype* of production quality code, which becomes the ***architectural baseline***.
        - It does not exclude the development of one or more exploratory, *throw-away prototypes* to mitigate specific risks:
            - *Refinement of the requirements*
            - *Feasibility*
            - *Human-interface studies*
            - *Demonstrations to investors*
            - *Etc*.

        - It becomes the ***architectural baseline***.
    - (3) At the end of this phase, there is again a *"go-no go" decision* point to actually invest and build the product, or *bid* for the complete development of the contract.
    - (4) The *plans produced* must be **detailed** enough, and the *risks* sufficiently mitigated to be able to determine with accuracy the *cost and schedule* for the completion of the development.

- ***The Elaboration Phase Entry criteria:***
    - (1) The ***products*** *and* ***artifacts*** described in the exit criteria of the inception phase
    - (2) *The* ***project*** *was approved* by the project management and funding authority
    - (3) *The* ***resources*** required for the elaboration phase have been allocated

- ***The Elaboration Phase Exit criteria:***
    - (1) ***A detailed software development plan***, containing:
        - *An updated risk assessment*
        - *A management plan*
        - *A staffing plan*
        - *A phase plan* showing the number and contents of the iteration for the next phase
        - *An iteration plan*, detailing the next iteration
        - *The development environment* and other tools required
        - *A test plan*
    - (2) *A* ***baseline vision***, in the form of *a set of evaluation criteria* for the final product.
    - (3) Objective, measurable ***evaluation criteria*** for assessing the results of the initial iterations(s) of the construction phase.
    - (4) *A* ***domain analysis model*** (80% complete), sufficient to be able to call the corresponding architecture 'complete'.
    - (5) *A* ***software architecture description*** (stating constraints and limitations).
    - (6) *An* ***executable architecture baseline*****.**

3.3.3 Construction Phase

- ***Construction Phase Objectives:***
    - (1) The first release of the *final product*.
    - (2) The Construction Phase is broken down into several *iterations*--fleshing out the *architecture baseline* and evolving it in steps or increments toward the *final product*.

- (3) At each iteration, *the various artifacts* prepared during the elaboration phase (see above) *are expanded* and *revised*, but they *ultimately* ***stabilize*** as the system evolves in correctness and completeness.
- (4) *New artifacts* are produced during this phase beside the *software itself*.
  - *Documentation*, both internal and for the end-users
  - *Test beds* and *test suites*
  - *Deployment collateral* to support the next phase (marketing collateral, for example).

- **For each iteration of *Construction Phase:***

- ***Iteration Entry criteria:***
  - (1) *The product and artifacts* of the previous iteration.
  - (2) The *iteration plan* must state the iteration specific goals:
    - *Additional capabilities* being developed, which *use cases* or *scenarios* will be covered
    - *Risks* being mitigated during this iteration
    - *Defects* being fixed during the iteration

- ***Iteration Exit criteria:***
  - (1) The *same **products** and **artifacts** updated***,** plus:
  - (2) *A **release description document***, which captures the results of an iteration
  - (3) ***Test cases*** and ***results of the tests*** conducted on the products
  - (4) *An **iteration plan***, detailing the **next iteration**
  - (5) Objective measurable ***evaluation criteria*** for assessing the results of the next iteration(s)

- ***Construction Phase Exit Criteria:***
  - Toward *the end of the construction phase*, the following artifacts must be produced and *additional exit criteria* for the ***last iteration*** of the phase must be established:
    - (1) *A **deployment plan***, specifying as necessary:
      - Packaging
      - Pricing
      - Roll out
      - Support
      - Training
      - Transition strategy (e.g., an upgrade plan from an existing system)

        - Production (e.g., making floppies and manuals)
    - (2) ***User documentation*.**

3.3.4 Transition Phase

- ***Transition Phase Objectives:***
    - (1) To put the product in the hands of its end users.
        - It involves *issues of marketing, packaging, installing, configuring, supporting the user-community, making corrections, etc.*
    - (2) From a **technical** perspective the *iterations* continue with one or more *releases* (or deliveries):
        - *`Beta' releases,*
        - *General availability releases,*
        - *Bug fixes,*
        - *Enhancement releases*.
    - (3) The phase is completed when the user community is satisfied with the product: ***formal acceptance*** for example in a contractual setting, or when all *activities on this product are terminated.*
    - (4) It is the point where some of the accumulated assets can be made *reusable* by the next cycle or by some other projects.

- ***Transition Phase Entry criteria:***
    - (1) *The **product and artifacts*** of the previous iteration
    - (2) A ***software product*** sufficiently matured to be put into the hands of its users

- ***Transition Phase Exit criteria:***
    - (1) *An update of **some of the previous documents*** (as necessary)
    - (2) The **plan** being replaced by *a "**post-mortem" analysis*** of the performance of the project relative to its original and revised success criteria
    - (3) *A **brief inventory*** of the organization's new assets as a result of this cycle

3.3.5 Evolution Cycles

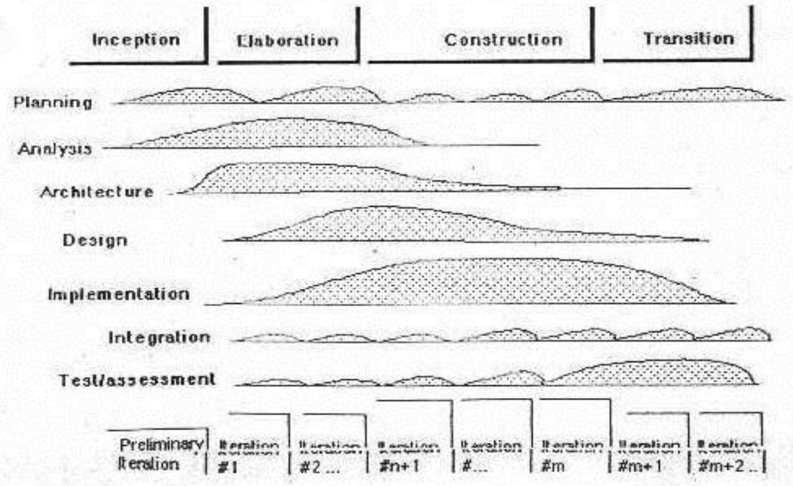
- For a substantial **evolution**, *the whole process can be applied* ***recursively***, starting again at the inception phase, for a ***new cycle***.
    - Since a product exists already, *this **inception phase** may be considerably reduced*, compared to an initial development cycle.
- The ***elaboration phase** also may be limited* and focused more on the *planning aspects* than evolving the analysis or the architecture.

- Said otherwise: *cycles can slightly **overlap***.

- Minor evolutions are done in *extending the **transition phase***, adding one or more iterations.
    - Alternatively, the transition phase may be concluded by an *end-of-life process*, i.e., the product does not evolve any more, but some specific actions must be taken in order to terminate it, or retire it.

3.4 Activities in the Rational Process

- The *names of the **phases*** of the Rational Process **stay away** from the terms describing an intellectual *activity*: *analysis, design, test*, etc.
    - A such *activity* in **not** confined to a **certain phase**
- The **activities** remain *independent* of terms employed by other authors, standards, and domain-specific jargon.
- These *activities* do take place, but *in varying degree in **each** phase and iteration* (Fig. 3.4.a.).



▨	Planning and management	15%
▨	Analysis/requirements	10%
▨	Design/integration	15%
▨	Implementation/functional tests	30%
▨	Measurement/assessment/acceptance test	15%
▨	Tools/environment/change management	10%
▨	Maintenance (fixes during development)	5%

Fig. 3.4.a. Activities in the Rational Process

- The *exact nature* and *contents of the iterations* **evolves** over time, although all are structured in the same way.
- *The beginning* of an activity is **not** bound to *the end of another*, **e.g**., *design does not start when analysis completes*
- The various *artifacts* associated with the activities *are revised* as the problem or the requirements are better understood.
- In an iterative process, *the activities* of *planning*, *test* and *integration are spread incrementally* throughout the cycle, *in each iteration*, and **not** massively lumped at the beginning and at the end, respectively.
- The *activities* do not appear as separate steps or phases in the process.
- Although this will vary considerably depending on the project discriminants, a **typical initial development cycle** for a **medium size project** should anticipate the following ratios for various activities:

Planning and management	15%
Analysis/requirements	10%
Design/integration	15%
Implementation/functional tests	30%
Measurement/assessment/acceptance test	15%
Tools/environment/change management	10%
Maintenance (fixes during development)	5%

3.5 Lifecycle Artifacts

- The RUP development process is ***not*** *document-driven*
  - Its main artifact must remain, at all times, the *software product itself*.
  - The *documentation* should remain lean and limited to the few documents that bring real value to the project from a **management** or **technical** point of view.
  - **Rational** suggests the following typical set of documents.

3.5.1 Management Artifacts

- The *management artifacts*:
  - Are not the products
  - They are used:
    - (a) To drive or monitor the progress of the project
    - (b) To estimate the risks
    - (c) To adjust the resources
    - (d) To give visibility to the customer (in a contractual setting) or the investors.
- **Rational** recommends the following **8** ***management artifacts****:*
  - (1) An ***Organizational Policy*** *document*, which is *the codification of the organization's process*
    - It contains an instance of this *generic process*.
  - (2) A ***Vision*** *document*, which describes:
    - (1) *The system level requirement*
    - (2) *Qualities*
    - (3) *Priorities*.
  - (3) A ***Business Case*** *document*, describing:
    - (1) *The financial context*
    - (2) *The contract*
    - (3) *Projected return on investment*, etc.
  - (4) A ***Development Plan*** *document*, which contains in particular:
    - (1) *The overall iteration plan*
    - (2) *The plan for the current iteration*
    - (3) *The plan for the upcoming iteration*.
  - (5) An ***Evaluation Criteria*** *document*, containing:
    - (1) *The requirements*
    - (2) *Acceptance criteria*
    - (3) Other *specific technical objectives*, which evolves from major milestone to major milestone.
    - It contains also:
      - (4) *The iteration goals*
      - (5) *The acceptance levels*.
  - (6) A ***Release Description*** *documents* for *each release*.
  - (7) A ***Deployment*** *document*, gathering *additional information* useful for:
    - (1) *Transition*

        - (2) *Training*
        - (3) *Installation*
        - (4) *Sales*
        - (5) *Manufacturing*
        - (6) *Cut-over*.
    - (8) A ***Status Assessment*** *documents*:
        - (1) *Periodic snapshots* of project status with *metrics of progress*
        - (2) *Staffing*
        - (3) *Expenditure*
        - (4) *Results*
        - (5) *Critical risks*
        - (6) *Actions items*
        - (7) *Post-mortem analysis*

3.5.2 Technical Artifacts

- The *technical artifacts* are:
    - (1) The *delivered goods* (*executable software* and *manuals*)
    - (2) The *blueprints* that were used to manufacture the *delivered goods*, (*software models, source code,* and *other engineering information* useful to understand and evolve the product).
- **Rational** recommends the following **3** ***technical artifacts****:*
    - (1) ***User's Manual***, developed early in the lifecycle.
    - (2) ***Software documentation***, preferably in the form of
        - *Self-documenting source code*, supported by appropriate tools to maintain it
        - *Models* *(uses cases, class diagrams, process diagrams, etc.)* captured and maintained with appropriate CASE tools.
    - (3) A ***Software Architecture*** *document*, describing:
        - *The overall structure of the software*,
        - *Its decomposition in major elements*: *class categories, classes, processes, subsystems, the definition of critical interfaces, and rationale for the key design decisions.*
- The artifacts enumerated in the *entry* and *exit criteria* in section 3 *can all be mapped onto one of these 11 documents*.
- Depending on the type of project, *this typical document set* can be *extended* or *contracted*, some documents can be *merged*.
- The documents do not have to be *paper* documents
    - They can also be *spreadsheet, text-files, database, annotations in source code, hypertext documents*, etc.

- The corresponding information source must be *clearly identified, easily accessible*, and *some of its history preserved*.

3.5.3 Requirements

- The **Rational Process** is ***not*** *requirement-driven* either.
- The *requirements* for the product *evolve* during a cycle, and take ***different forms***:
    - (1) The *business case* gives *the main constraints*, mostly in terms of resources that can be expended.
    - (2) The *vision* document describes only the *key requirements* of the system from a user's perspective,
        - It evolves only slowly during the development cycle.
    - (3) The *more detailed requirements* are elaborated during the elaboration phase, in the form of *use cases* and *scenarios*
- The *requirements* are *refined incrementally* throughout the construction phase, as the product and the users needs become better understood.
    - These *more detailed requirements* are in the *evaluation criteria* document;
    - They drive the definition of the contents of the construction and transition iterations and are referenced in the *iteration plan*.

3.6 Examples of Rational Processes

- The Rational process takes different aspects depending on the *discriminants* (Fig. 3.6.a.)

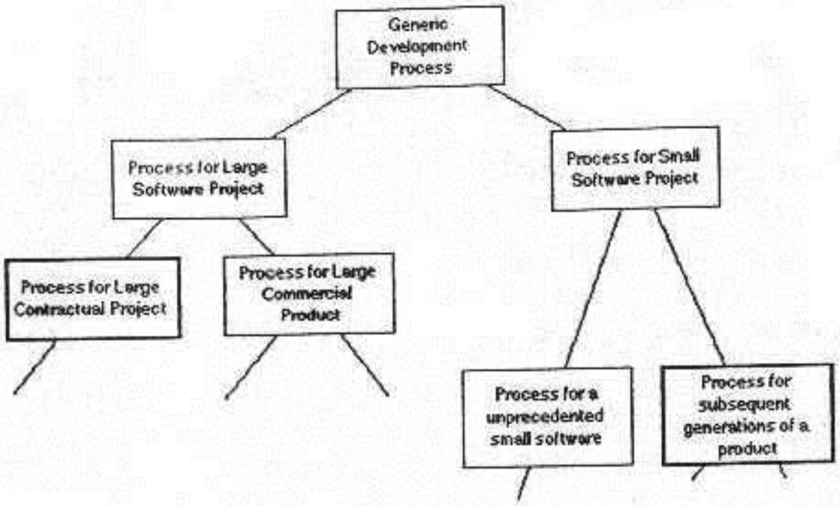

Generic Development Process
Process for Large Software Project
Process for Small Software Project
Process for Large Contractual Project
Process for Large Commercial Product
Process for a unprecedented small software
Process for subsequent generations of a product

Fig. 3.6.a. Taxonomy of Project Development Process in RP

3.6.1 Rational Process for Large Contractual Software Development

- Rational proposes to set up the procurement of **large software** in 3 stages, associated with 3 different kinds of contract. (Fig. 3.6.1.a)
  - (1) An **R&D** stage, comprising the inception and elaboration phase, typically bid in a risk sharing manner, e.g., as a *Cost Plus Award Fee contract* (CPAF).
  - (2) A **production** stage, comprising the construction and transition phases, typically bid as a *Firm Fixed Price contract* (FFP).
  - (3) A **maintenance** stage if any, corresponding to the evolution phase, typically bid as a *Level of Effort contract* (LOE).

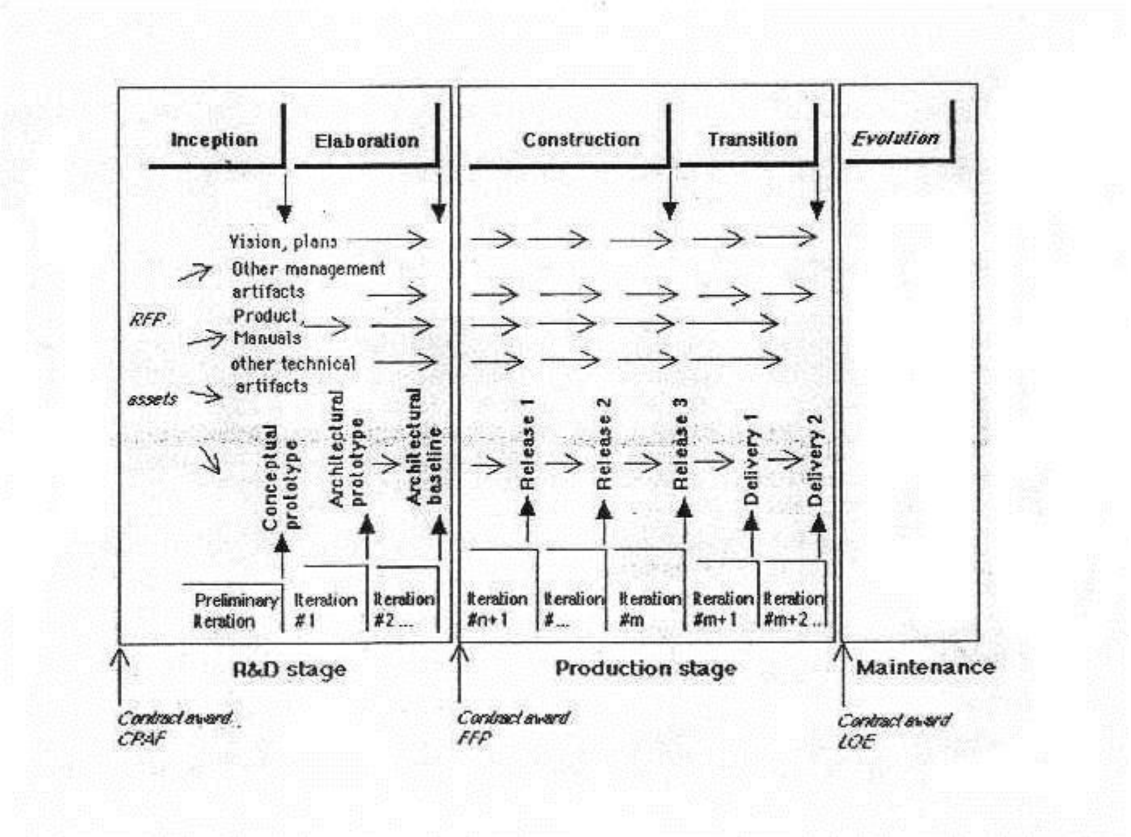


Fig. 3.6.1.a. Rational Process for Large SW Project Development

- ***Large SW Project Characteristics:***
  - A *higher level of visibility* is usually required from the customer on the evolution of the project
  - A *large number of people and organizations* are involved
  - *More formalism* is required in the process
  - There may be *more emphasis on written artifacts*, than would be in the case of a small, internal project.
  - *All 11 described documents* are present in some form or name.

3.6.2 Rational Process for a Small Commercial Software Product

- A **small commercial development** would see a *more fluid process*, with only *limited amount of formalism* at the major milestones and a *more limited set of documents*:
  - (1) A product ***vision***
  - (2) A ***development plan***, showing schedule and resources
  - (3) ***Release description*** documents, specifying the goal of an iteration at the beginning of the iteration, and updated to serve as release notes at the end
  - (4) ***User documentation***, as necessary
  - (5) ***Software architecture, software design***, and ***development process and procedures*** can be documented by the code itself or the software development environment.

3.7. Conclusion

- The **Rational Process**
  - (1) Puts an emphasis on addressing very early *high risk areas*, by developing rapidly an *initial version* of the system, which defines its *architecture*.
  - (2) It does not assume *a fixed set of firm requirements* at the inception of the project, but *allows to refine the requirements* as the project evolves.
  - (3) It expects and accommodates *changes*.
  - (4) The process does not put either a strong focus on *documents or `ceremonies'*, and it lends itself to *the automation* of many of the tedious tasks associated with software development.
  - (5) The *main focus* remains *the software product itself*, and its quality, as measured by the degree to which it satisfies its end-users, and meets its return on investment objective altogether.
  - (6) A process derived from the generic process described here would fully conform to the requirements of a standard such as ISO 9000.

3.8 Glossary

Artifact	Any document or software other than the software product itself.
Baseline	A release that is subject to change management and configuration control.
Construction	The 3rd phase of the process, where the software is brought from an executable architectural baseline to the point where it is ready to make the transition to its user's community.
Cycle	One complete pass through the 4 phases: inception, elaboration, construction, and transition. The span of time between the beginning of the inception phase and the end of the transition phase.
Elaboration	The 2nd phase of the process where the product vision and its architecture are defined.
Evolution	The life of the software after its initial development cycle; any subsequent cycle, where the product evolve.

Generation	The result of one software development cycle.
High fidelity	Sufficient accuracy (in a cost estimate, quality estimate, or schedule estimate) that a development contractor would commit to a firm, fixed price achievement.
Inception	The first phase of the process, where the seed--idea, RFP, previous generation--is brought up to the point of being (at least internally) founded to enter into the elaboration phase.
Iteration	A distinct sequence of activities with a baselined plan and an evaluation criteria.
Milestone	An event held to formally initiate and/or conclude an iteration.
Phase	The span of time between 2 major milestones of the process where a well defined set of objectives are met, artifacts are completed, and decisions are made to move or not into the next phase.
Product	The software that is the result of the development, and some of the associated artifacts (documentation, release medium, training).
Prototype	A release which is not necessarily subjected to change management and configuration control.
Release	A subset of the end-product which is the object of evaluation at a major milestone (see: prototype, baseline).
Risk	An ongoing or upcoming concern which has a significant probability of adversely affecting the success of major milestones.
Transition	The 4th phase of the process where the software is turned into the hands of the user's community.
Vision	The user's view of the product to be developed.

3.9 Acronyms

CPAF	Cost Plus Award Fee
FFP	Firm Fixed Price
LOE	Level Of Effort
OOT	Object-Oriented Technology
RFP	Request for Proposal
ROI	Return On Investment

Exercise #2

1) What is the *Sync-and-Stabilize Development Approach*? Describe the *phases* of this approach

2) What are the MS Development *strategies*? Describe some *principles* of each strategy

3) Describe the *rules*, the *working manner* and the *teams* in MS Development Approach

4) Describe the Oracle *philosophy* of developing projects

5) Which are the *manager responsibilities* in Oracle Development technology?

6) Describe the Oracle *project phases* and *project processes*

7) Which are the *perspectives* of the Rational Process? How the perspectives are *reconciled*?

8) Which are the *phases* of the RP? Describe their contents (*objectives, outcomes, entry and exit criteria*)

9) Which are the *activities* in the RP. Describe their relations with the RP phases.

10) Describe RP *lifecycle artifacts*

11) Give some examples of Rational Process (e.g. large contractual SW, small commercial SW development