

SE Module 1

The Product and the Process

- **Software** is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.
- Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

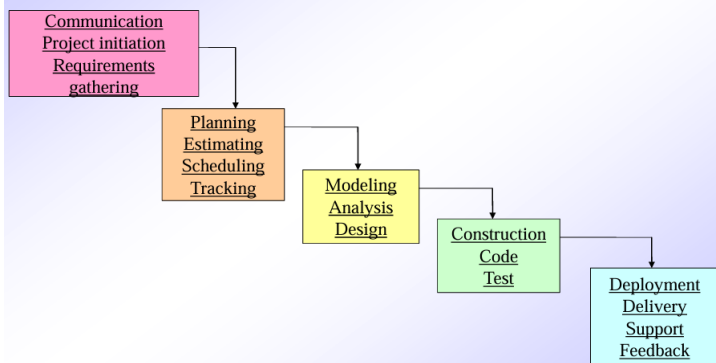
Software Engineering – Defined

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product

Waterfall Model

- Oldest software lifecycle model.
- Used when requirements are well understood and risk is low.
- Work flow is in a linear (i.e., sequential) fashion.
- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood .
- There are no ambiguous requirements.
- The project is short.

Waterfall Model (Diagram)



Remember it as **CPMCD**

C-Communication

P- Planning

M-Modelling

C-Construction

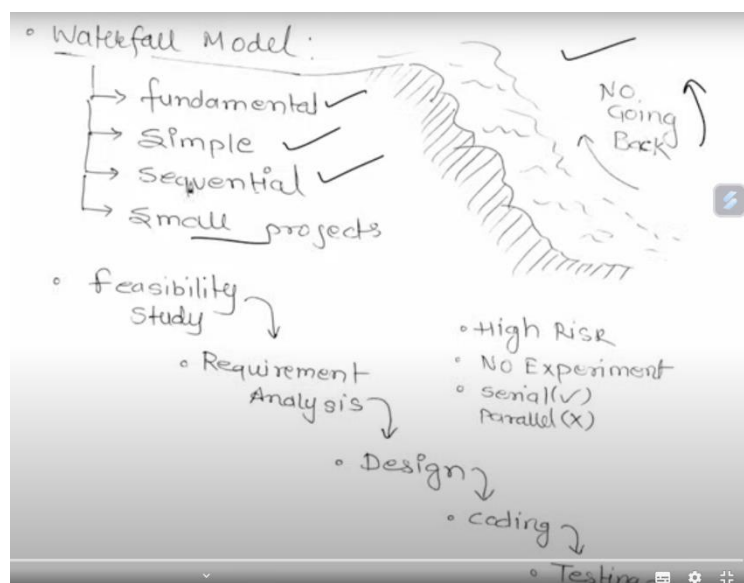
D-Deployment

Advantages

1. **Simple and Easy to Understand:** The model is straightforward, making it easy for everyone to follow.
2. **Easy to Manage:** Each phase has clear deliverables (outputs) and a review process, making management easier.
3. **One Phase at a Time:** Phases are completed one after the other, reducing complexity.
4. **Good for Small Projects:** Works well when you have a clear understanding of requirements, especially in smaller projects.
5. **Clearly Defined Stages:** Each stage of development is well defined, providing structure.
6. **Well Documented:** The process and outcomes are documented, making it easier to track progress.

Disadvantages

1. **Late Delivery of Working Software:** You won't see a working version of the software until much later in the project.
2. **No Iteration:** Changes are hard to incorporate once a phase is complete, which can lead to confusion if adjustments are needed.
3. **Requirement Challenges:** It can be tough for customers to clearly express all their requirements at the start.
4. **Customer Patience Required:** Customers have to wait until the final phase to see a functioning version, which can be frustrating.
5. **High Risk and Uncertainty:** If any assumptions turn out to be wrong, it can lead to significant problems later on.



RAD Rapid Application Development

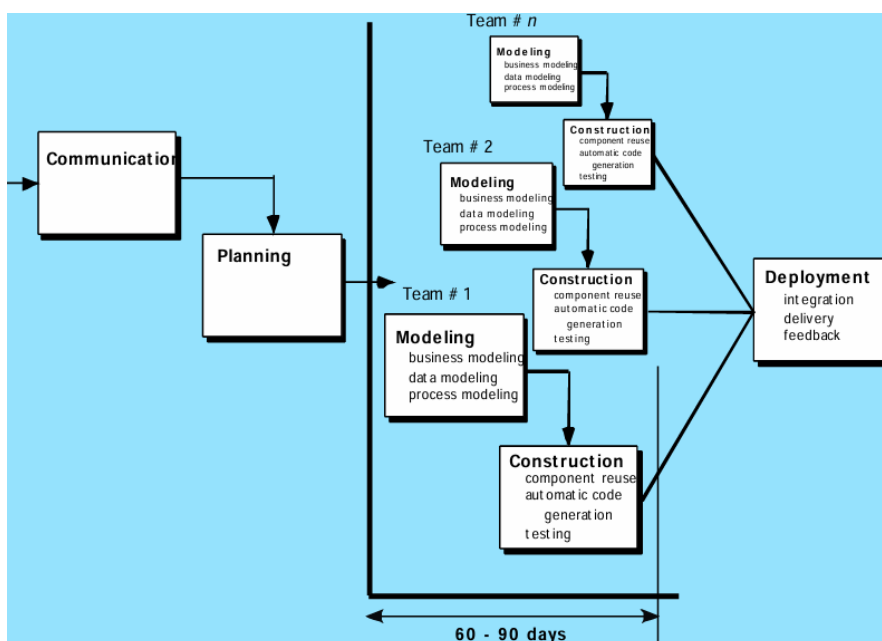
- **Faster Development:** RAD is a software development approach aimed at speeding up the process and improving quality compared to traditional methods.
- **Parallel Development:** Instead of building the entire system in a linear fashion, RAD focuses on creating functional parts (modules) as prototypes at the same time. These prototypes are then combined to form the complete product, allowing for quicker delivery.

Key Characteristics:

1. **Iterative and Incremental:** RAD follows an iterative approach, meaning it allows for regular feedback and adjustments, and it develops the software in small, manageable increments.
2. **Small Teams:** Projects typically involve small, collaborative teams that include developers, domain experts, customer representatives, and other IT professionals. This promotes better communication and quicker decision-making.
3. **Flexibility:** Since RAD doesn't rely on extensive preplanning, it can easily adapt to changes that arise during the development process.
4. **Reusability of Prototypes:** For RAD to succeed, it's crucial that the prototypes created can be reused in the final product. This maximizes efficiency and reduces redundant work.

Summary

RAD is great for projects that need to be completed quickly and where requirements might change. Its focus on prototypes and small teams allows for flexibility and faster delivery, but it relies on the reusability of components to be effective.



Advantages

1. **Accommodates Changes:** Easy to adapt to changing requirements during development.
2. **Measurable Progress:** You can track how much has been completed at any point.
3. **Short Iteration Times:** Powerful RAD tools can speed up development cycles.
4. **Increased Reusability:** Components and prototypes can often be reused, saving time and effort.
5. **Reduced Development Time:** Overall time to deliver the final product is shorter.
6. **Encourages Customer Feedback:** Regular interactions with users help ensure the final product meets their needs.
7. **Early Integration:** Starting integration early helps identify and resolve potential issues before the final product is completed.

Disadvantages

1. **Need for Strong Team Members:** Requires technically skilled team members to effectively identify and understand business requirements.
2. **Modular Requirement:** Only suitable for systems that can be broken down into modules; not all projects fit this model.
3. **High Costs for Smaller Projects:** The expense of modeling and automated code generation can make RAD unsuitable for lower-budget projects.
4. **Continuous User Involvement Needed:** Users must be engaged throughout the entire process, which may not always be feasible.
5. **Shorter Development Focus:** Best for projects that require quick turnaround times, which might limit scope.
6. **Increased Management Complexity:** Managing a project with multiple prototypes and constant iterations can be more challenging.

Spiral Model

The Spiral Model is ideal for projects where requirements are unclear and risks are significant. It's particularly useful in complex projects that may evolve over time.

Key Characteristics:

1. **Inner Spirals:**
 - Focus on identifying software requirements and assessing risks.

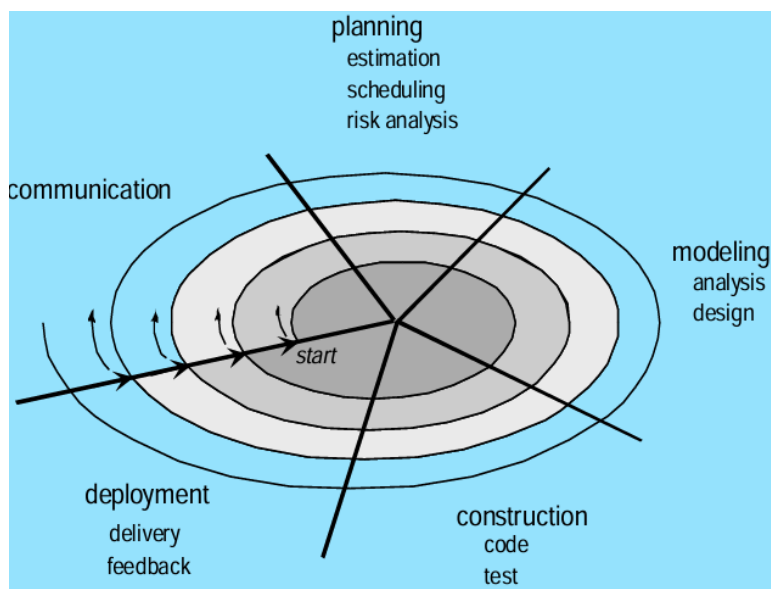
- May involve prototyping to help clarify requirements and gather feedback.

2. Outer Spirals:

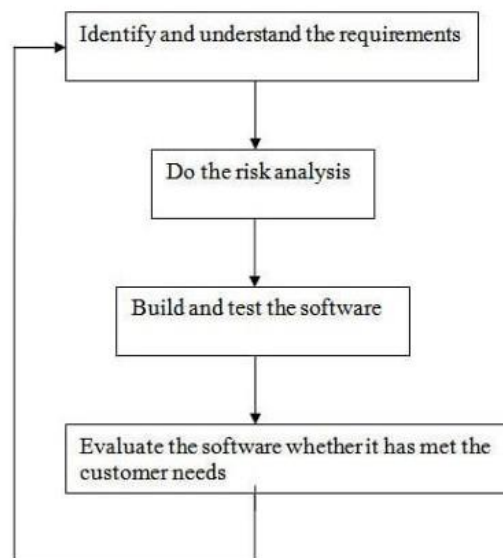
- Once requirements are defined, these spirals adopt a more traditional waterfall approach.
- Allow for iterative development, meaning the software can grow and improve with each cycle.

3. Risk Management:

- Emphasizes the importance of risk assessment at each iteration.
- Each cycle focuses on identifying and addressing the most critical risks first, making the process adaptive and responsive.



To explain in simpler terms, the steps involved in the spiral model are:

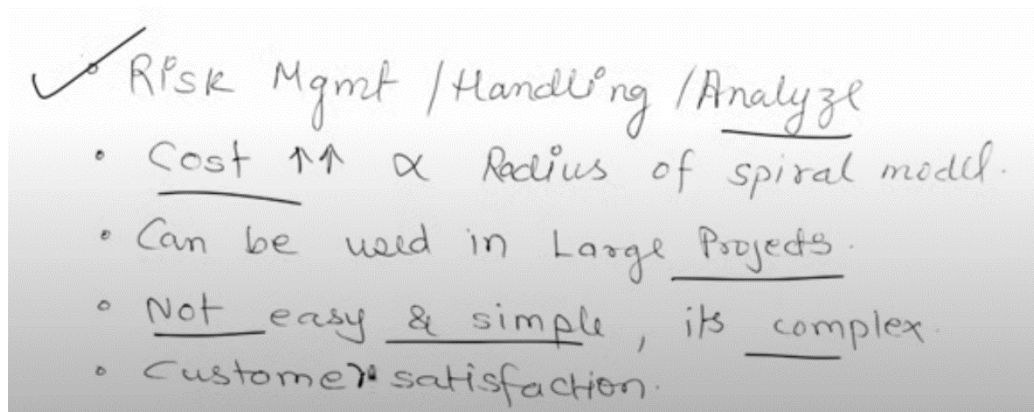


Advantages

1. **Realism:** Reflects the iterative nature of software development, especially for unclear requirements.
2. **Accurate Requirements:** Captures user needs more effectively through ongoing feedback.
3. **Early Visibility:** Users get to see and interact with the system early in development.
4. **Better Risk Management:** Breaks development into smaller parts, allowing risky components to be addressed sooner.
5. **Flexibility:** Combines benefits of both waterfall and evolutionary models.

Disadvantages

1. **Complex Management:** More challenging to manage due to its iterative nature.
2. **Uncertain End Dates:** Project completion timelines may be unclear.
3. **Not for Small Projects:** Less suitable for small or low-risk projects; can be costly for them.
4. **Potential for Indefiniteness:** The process can continue indefinitely if not properly managed.
5. **Excessive Documentation:** A large number of intermediate stages leads to more documentation needs.



Open Source Model

All successful open-source software projects go through two informal phases:-

📌 Initial Phase:

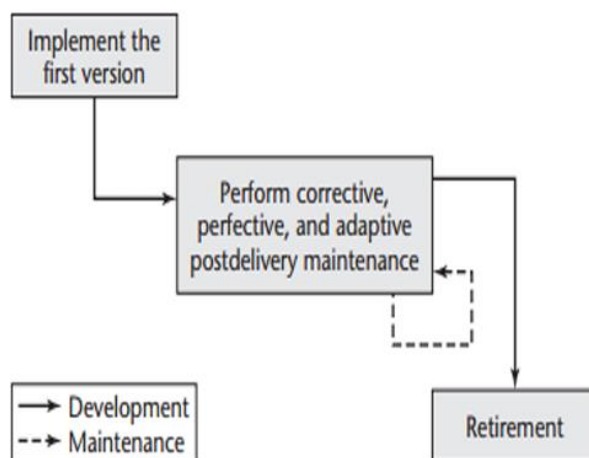
- An individual has an idea for software (like Linux, Firefox, or Apache).
- They build the first version of the program.

- The software is released for free online.

❓ User Participation Phase:

- Once users start using the software, they become co-developers.
- Users can:
 - Report bugs and suggest fixes.
 - Propose new features, with some users implementing those ideas.
 - Adapt the software to work on different operating systems or hardware.

Figure 2.11
The open-source life-cycle model.



Agile Process Model

Agile is a software development lifecycle (SDLC) model that combines iterative and incremental approaches. Its main goals are adaptability and customer satisfaction through rapid delivery of functional software.

Key Features

1. Incremental Builds:

- The product is divided into small parts (builds).
- Each build is delivered in short cycles called iterations, usually lasting 1 to 3 weeks.

2. Cross-Functional Teams:

- Teams work simultaneously on various tasks, including:
 - Planning
 - Requirements analysis

- Design
- Coding
- Unit testing
- Acceptance testing

3. Customer Involvement:

- At the end of each iteration, a working version of the product is shown to the customer and stakeholders for feedback.

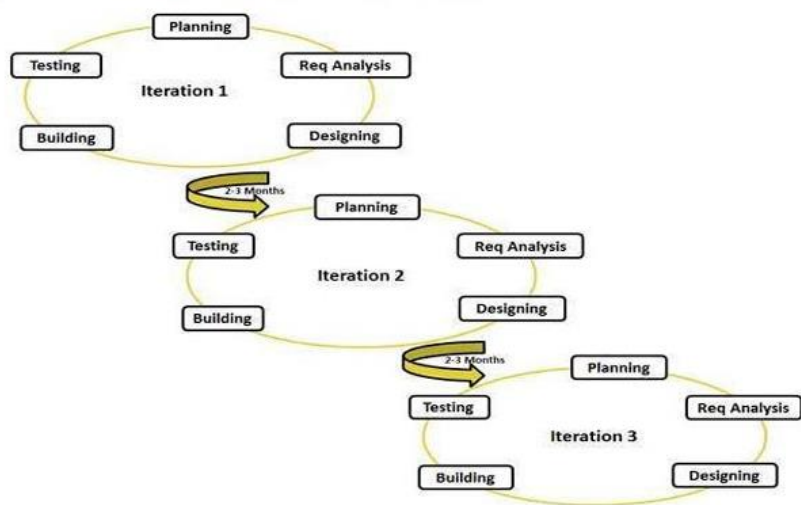
4. Iterative Development:

- Working software is delivered after each iteration.
- Each build adds more features, with the final product including all required functionalities.

5. Adaptability:

- Agile processes are designed to adapt incrementally, allowing teams to manage unpredictability effectively.

Here is a graphical illustration of the Agile Model:



Advantages

1. **Realistic Approach:** Reflects real-world software development challenges.
2. **Promotes Teamwork:** Encourages collaboration and skill-sharing among team members.
3. **Rapid Development:** Features can be developed quickly and shown to stakeholders.
4. **Flexible Requirements:** Works well with both fixed and changing requirements.
5. **Early Solutions:** Provides working versions of the software early in the process.

6. **Adaptability:** Ideal for environments where conditions change regularly.
7. **Easy Management:** Simpler to manage due to its iterative nature.
8. **Customer Focus:** Prioritizes satisfying the customer above all.

Disadvantages

1. **Effort Assessment Challenges:** Hard to estimate effort needed for large projects at the start.
2. **Less Focus on Design and Documentation:** May overlook detailed planning and documentation.
3. **Risk of Misalignment:** Can go off track if the customer isn't clear about their needs.
4. **Requires Experienced Resources:** Needs skilled team members to be effective.

Agile Process Models

- Extreme Programming (XP)
- Adaptive Software Development (ASD)
- Dynamic Systems Development Method (DSDM)
- Scrum
- Crystal
- Feature Driven Development (FDD)
- Agile Modeling (AM)

Extreme Programming (XP)

Based on object-oriented principles, XP relies on specific rules and practices to manage unpredictability in software development.

XP Overview

XP Planning

- **User Stories:** Write down what users want.
- **Cost Assessment:** Team assigns a cost to each story.
- **Group Stories:** Organize stories into deliverable chunks.
- **Delivery Commitment:** Set a delivery date for each chunk.
- **Project Velocity:** Measure speed of delivery to estimate future dates.

XP Design

- **KIS Principle:** Keep It Simple.
- **CRC Cards:** Use cards to clarify design responsibilities.

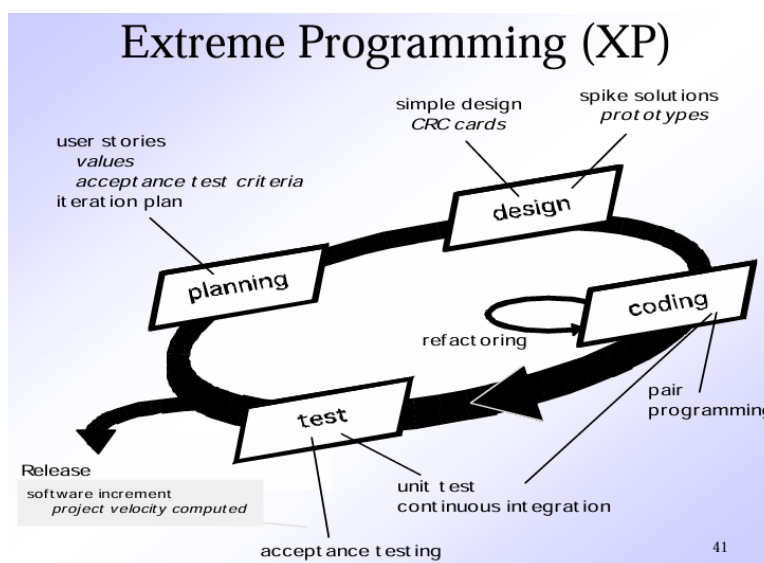
- **Spike Solutions:** Create quick prototypes for tough designs.
- **Refactoring:** Continuously improve code design.

XP Coding

- **Unit Tests First:** Write tests before coding.
- **Pair Programming:** Two developers work together on the same code.

XP Testing

- **Daily Testing:** Run all tests every day.
- **Acceptance Tests:** Customers define tests to ensure it meets their needs.



Adaptive Software Development (ASD)

Adaptive Software Development (ASD) is a software development methodology designed to address the challenges of complex projects in unpredictable environments. Here are the main aspects of ASD:

Key Characteristics

1. **Focus on Complexity:** ASD is particularly suited for building complex software systems where requirements may change frequently.
2. **Self-Organization:** Teams are empowered to organize themselves. This means that individuals collaborate and make decisions collectively, enabling them to tackle problems more effectively.
3. **Interpersonal Collaboration:** Strong emphasis on teamwork and communication, allowing members to share knowledge and ideas freely.

4. **Learning-Oriented:** Encourages both individual and team learning throughout the development process. Teams adapt based on what they learn from each iteration.
5. **Adaptability:** ASD is designed to be flexible, accommodating changes easily without extensive planning. This is crucial in dynamic environments.
6. **Result-Focused:** The lifecycle prioritizes achieving results, specifically application features, rather than just completing tasks. This helps ensure that the delivered product meets user needs.

Lifecycle Stages

ASD typically involves cycles of planning, executing, and assessing to iteratively improve the software, reflecting the adaptive nature of the methodology.

ASD — Distinguishing Features

1. **Mission Driven:**
 - Activities in each development cycle are aligned with the overall project mission.
2. **Component Based:**
 - Focuses on developing working software rather than being task-oriented, prioritizing results.
3. **Iterative:**
 - Involves redoing development as needed, emphasizing continuous improvement rather than getting it right the first time.
4. **Time Boxed:**
 - Sets fixed delivery timelines for projects to maintain focus and urgency.
5. **Change Tolerant:**
 - Views the ability to incorporate change as a competitive advantage rather than a problem.
6. **Risk Driven:**
 - High-risk items are prioritized and addressed early in the development process.

ASD Phases

1. **Speculation:**
 - **Project Initiation:** Start with a customer mission statement, delivery dates, and specified requirements.

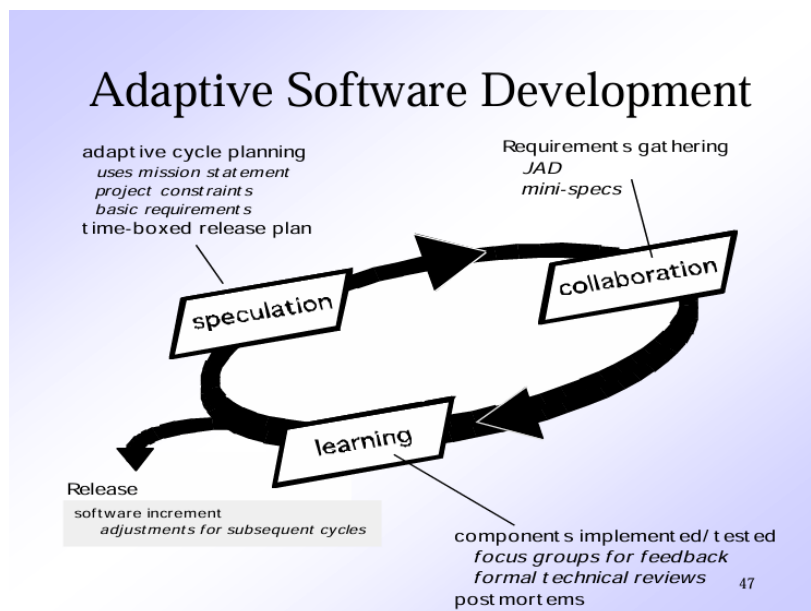
- **Adaptive Cycle Planning:** Acknowledge that requirements will change and plan for those changes.

2. Collaboration:

- Requires teamwork from a cohesive, well-functioning team.
- Joint Application Development (JAD) is preferred for gathering requirements collaboratively.

3. Learning:

- Implement and test components while gathering feedback.
- Use focus groups to provide insights.
- Conduct formal technical reviews and postmortems to evaluate and learn from the process.



Dynamic Systems Development Method (DSDM)

DSDM is an Agile project delivery framework specifically designed to build and maintain systems within tight time constraints. It emphasizes incremental development and prototyping in a controlled environment, allowing teams to deliver functionality quickly and adapt to changes.

Key Aspects:

- **Time-Boxed Delivery:** Projects are divided into increments, each with a fixed timeframe for delivery. This helps ensure that teams focus on delivering workable features within specific deadlines.

- **Incremental Prototyping:** Features are developed incrementally, allowing for early user feedback and adjustments based on real-world use.
- **Focus on Business Needs:** DSDM prioritizes delivering business value, ensuring that the final product meets user requirements effectively.
- **Collaborative Approach:** Encourages strong collaboration among stakeholders, including users, developers, and project managers, throughout the development process.

Overall Goal:

The main aim of DSDM is to enable organizations to respond quickly to changing requirements and deliver high-quality software on time and within budget.

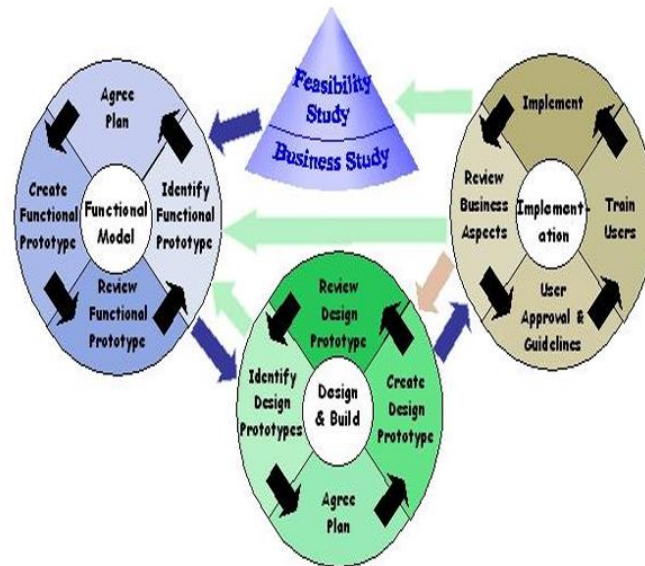
DSDM — Distinguishing Features

1. **Active User Involvement is Imperative:**
 - Users play a crucial role throughout the development process, ensuring their needs are met.
2. **Teams Must be Empowered to Make Decisions:**
 - Development teams are given the authority to make decisions, fostering autonomy and responsiveness.
3. **Focus on Frequent Delivery:**
 - Emphasizes delivering functional increments regularly to gather feedback and adapt quickly.
4. **Fitness for Business:**
 - Deliverables are accepted based on how well they meet business needs, ensuring relevance and value.
5. **Iterative and Incremental Development is Mandatory:**
 - Development occurs in cycles, allowing for continuous improvement and adaptation.
6. **Requirements Baselined at High-Level:**
 - Some high-level requirements are established to limit changes during development, providing a framework while allowing flexibility.
7. **Testing Integrated Throughout the Lifecycle:**
 - Testing is not a final step but is incorporated at every stage, ensuring quality is maintained.

8. Collaborative and Cooperative Approach:

- Strong collaboration between technical and business teams is essential for success, emphasizing teamwork and communication.

The DSDM Development Process



DSDM Life Cycle Activities

1. Feasibility Study:

- Establishes basic business requirements and constraints.
- Assesses whether the application is suitable for the DSDM process.

2. Business Study:

- Identifies functional and information requirements to ensure the application delivers business value.

3. Functional Model Iteration:

- Produces incremental prototypes demonstrating functionality for the customer.
- Allows for the incorporation of additional requirements based on user feedback.

4. Design and Build Iteration:

- Revisits prototypes created during the functional model iteration.
- Ensures each prototype is engineered to provide operational business value for end users.

5. Implementation:

- Places the latest software increment (an “operationalized” prototype) into the operational environment for use.

Scrum

Scrum is an Agile framework designed for managing and completing complex projects, particularly in software development. It emphasizes iterative progress, collaboration, and flexibility. Scrum facilitates teams in delivering high-quality products by breaking work into manageable units called sprints, fostering continuous improvement, and maintaining close communication with stakeholders.

[Sprint is an duration or a time period of a month or less]

Scrum Overview

1. Development Work Partitioned into “Packets”:

- Work is divided into smaller, manageable units to streamline the development process.

2. Ongoing Testing and Documentation:

- Testing and documentation occur continuously as the product is developed, ensuring quality and clarity.

3. Work in Sprints:

- Development occurs in time-boxed iterations known as sprints, which pull from a backlog of requirements.

Backlog:

[It is just like our Backlog]

It has a list of works to be done But the list of work to be done is in priority order]

- **Prioritized List:** A list of project requirements ranked by importance.
- **Item Management:** New items can be added, and existing priorities can be updated as necessary.

Sprints:

- **Work Units:** Each sprint consists of tasks required to meet specific backlog requirements.
- **Time-Boxed:** Sprints must be completed within a set timeframe (usually 1 to 4 weeks).

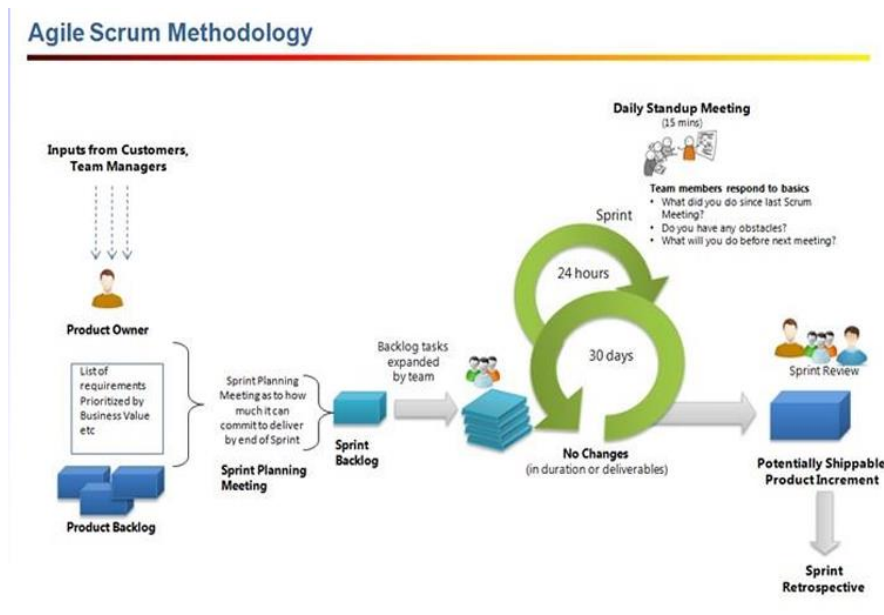
Demos:

- **Increment Delivery:** At the end of each sprint, a software increment is delivered to the customer.

- **Functionality Evaluation:** Customers can evaluate the features implemented, even if not all planned functions are included, as long as delivery occurs within the sprint timeframe.

Scrum Process Patterns:

- **Adaptability:** Scrum supports teams in managing uncertainty, promoting effective collaboration and responsiveness to change throughout the project



Crystal

Crystal is a family of Agile methodologies designed to accommodate various project sizes and criticalities. It emphasizes flexibility and adaptability in software development.

Key Features:

- **Lightweight Approach:** Crystal is known for being one of the most lightweight methodologies, making it easy to adapt to different project needs.
- **Family of Methodologies:** It consists of several Agile methods tailored to specific factors like team size, system criticality, and project priorities.

Core Characteristics:

1. **Teamwork:** Strong emphasis on collaboration among team members.
2. **Communication:** Encourages open communication to ensure everyone is aligned.
3. **Simplicity:** Focuses on keeping processes simple to enhance efficiency.
4. **Reflection and Improvement:** Regularly assesses and adjusts processes for continuous improvement.

Goals:

- **Primary Goal:** Deliver useful, working software efficiently.
- **Secondary Goal:** Prepare the team for future projects or iterations.

Feature Driven Development

Feature Driven Development (FDD) is an Agile methodology focused on delivering client-valued features through a structured process.

Distinguishing Features:

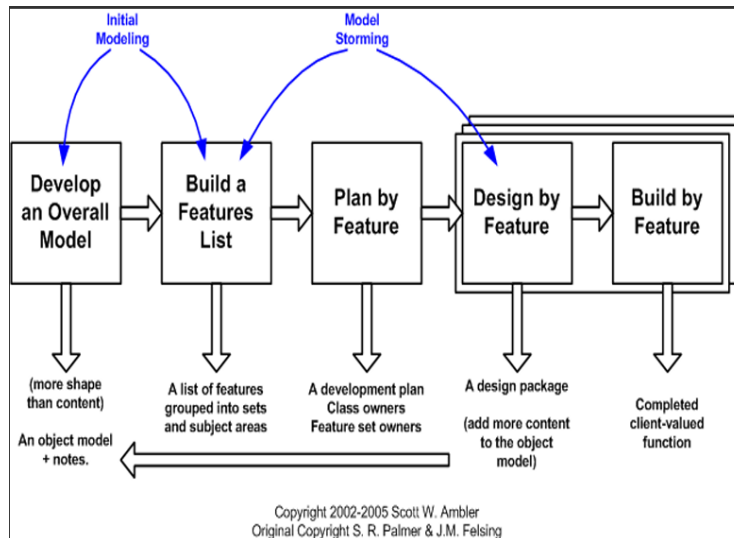
- **Feature Definition:** A feature is a small, client-valued function that can be implemented in two weeks or less, expressed as <action><result><object>.
- **Feature List Creation:** A list of features is generated, and planning is done by focusing on these features.
- **Design and Construction Merge:** Design and coding happen together, emphasizing collaboration among team members.

Key Components:

1. **Feature-Based Decomposition:** Manages complexity by breaking down the project into smaller, manageable features.
2. **Technical Communication:** Uses verbal, graphical, and textual methods to enhance understanding among team members.
3. **Quality Assurance:** Maintains software quality through incremental development, inspections, audits, metric collection, and design patterns.

Development Process Steps:

1. **Develop Overall Model:** Create a set of classes representing the business model of the application.
2. **Build Features List:** Extract features from the domain model, categorize, and prioritize them, breaking the work into two-week chunks.
3. **Plan by Feature:** Assess features based on priority, effort, technical issues, and dependencies.
4. **Design by Feature:** Select relevant classes, write method prologs, develop preliminary design details, and assign class owners responsible for documentation.
5. **Build by Feature:** Class owners translate designs into source code, conduct unit testing, and the chief programmer handles integration.



Agile Modelling (AM)

Agile Modelling (AM) is a practice-based methodology focused on effective modeling and documentation in software development.

Key Principles:

1. **Model with a Purpose:** Always have a specific goal for the model before creating it.
2. **Use Multiple Models:**
 - Utilize various models and notations to describe software.
 - Focus on a small subset that is essential for the project.
3. **Travel Light:**
 - Retain only those models that provide long-term value as development progresses.
4. **Content Over Representation:**
 - Prioritize conveying useful information to the audience rather than focusing on the model's appearance.
5. **Know Your Tools:**
 - Understand the strengths and weaknesses of the models and tools you use.
6. **Adapt Locally:**
 - Tailor the modeling approach to fit the specific needs of your agile team.

Process

- **Definition:** A process is a system of operations designed to produce something, consisting of a series of actions, changes, or functions that achieve a specific result.
- **Characteristics:**
 - **Sequence of Steps:** A process involves a set of steps performed to accomplish a particular goal.

Process Measurement

- **Purpose:** Measures the effectiveness of a software process indirectly.
- **Metrics Derived from Outcomes:**
 - **Errors:** Measures of errors identified before software release.
 - **Defects:** Defects reported by end-users after software delivery.
 - **Work Products:** Productivity metrics related to the deliverables.
 - **Human Effort:** The amount of effort expended by the team.
 - **Calendar Time:** Total time taken to complete the process.
 - **Schedule Conformance:** How well the process adheres to the planned schedule.

What is a Software Process?

- **Definition:** A software process is a set of activities, methods, practices, and transformations used to develop and maintain software, along with associated products like project plans, design documents, code, test cases, and user manuals.
- **Organizational Maturity:** As an organization matures, its software process becomes better defined and consistently implemented across all teams.
- **Process Maturity:** Refers to the extent to which a process is:
 - Explicitly defined
 - Managed
 - Measured
 - Controlled
 - Effective

- **Benefits of Metrics:** Software process metrics help organizations improve their process maturity by providing insights into effectiveness and areas for improvement.

Process Metrics

1. **Time Metrics:**
 - Measures the time taken to complete a process.
 - Includes total time, calendar time, and time spent by engineers.
2. **Resource Metrics:**
 - Evaluates resources needed for a process.
 - Can include person-days of effort, travel costs, and computer resource usage.
3. **Event Occurrence Metrics:**
 - Counts occurrences of specific events.
 - Examples include:
 - Number of defects found during code inspections.
 - Number of requirement changes requested.
 - Average lines of code modified due to requirement changes.
4. **Reuse Data:**
 - Assesses the number of components produced and their reusability.
5. **Quality Metrics:**
 - Focuses on the quality of work products and deliverables.

Immature Software Organizations

1. **Improvised Processes:** No structured methods; processes are made up as needed.
2. **Inconsistent Use:** Even if there are processes, they're not followed closely.
3. **Crisis Management:** Focus is on fixing immediate problems rather than planning ahead.
4. **Unrealistic Timelines:** Schedules and budgets are often not realistic, leading to overruns.
5. **Compromised Quality:** Rushing to meet deadlines often sacrifices product quality.
6. **No Quality Metrics:** There's no way to measure how good the processes are.

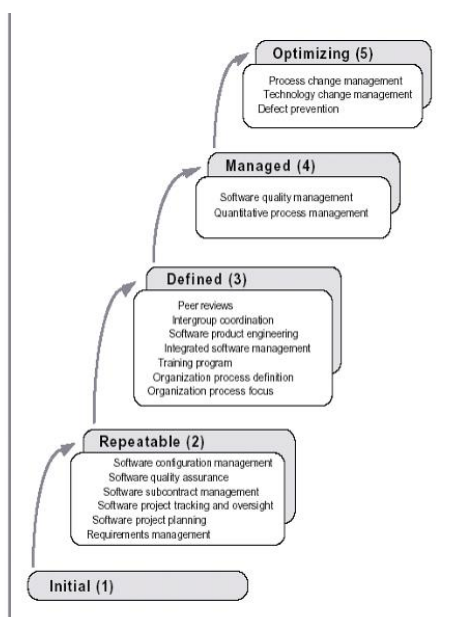
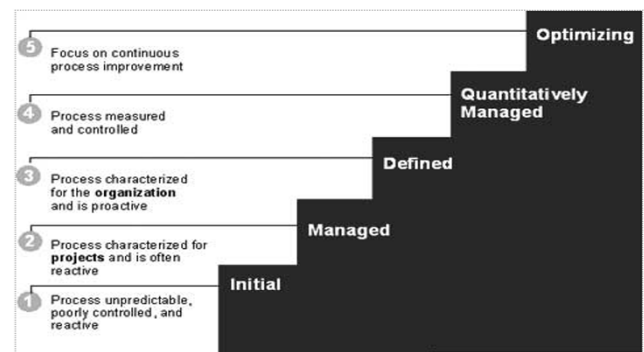
7. **Less Testing:** Reviews and testing get cut when projects fall behind schedule.

Capability Maturity Model (SW-CMM)

1. **CMMI Overview:** A framework for improving product quality and development efficiency in both hardware and software.
2. **Five Levels:** Describes organizational maturity across five levels, each with specific goals and process areas.
3. **Goal Measurement:** Progress is measured by achieving specific and generic goals tied to each process area.
4. **Maturity Levels:** Each level is an evolutionary step toward a mature software process, supporting ongoing improvement.
5. **Capability Levels:** Defines the organization's ability in a specific process area, with practices to enhance processes.

Five Levels of Software Process Maturity

CMMI Staged Representation- Maturity Levels



CMMI Levels

1. **Level 1: Initial**
 - **Chaos:** Processes are chaotic and rely on individuals.
 - **Outcome:** Projects often miss deadlines and budgets.
2. **Level 2: Managed**
 - **Basics:** Processes are managed and documented.
 - **Stability:** Practices are followed, even under stress.
3. **Level 3: Defined**
 - **Standardized:** Processes are documented and standardized across projects.
 - **Tailored:** Projects customize these standards.
4. **Level 4: Quantitatively Managed**
 - **Data-Driven:** Uses statistics to control processes.
 - **Predictable:** Performance is measurable and predictable.
5. **Level 5: Optimizing**
 - **Improvement:** Focus on continuous improvement and innovation.
 - **Root Causes:** Addresses underlying issues for better processes.

CMMI Capability Levels

1. **Level 0: Incomplete**
 - **Not Formed:** Processes are missing or not meeting goals.
2. **Level 1: Performed**
 - **Goals Met:** All specific CMMI goals are satisfied.
 - **Work Done:** Tasks are completed as defined.
3. **Level 2: Managed**
 - **Conformance:** Follows organizational policies.
 - **Resources:** Adequate resources are available.
 - **Stakeholder Involvement:** Active participation from stakeholders.
 - **Evaluation:** Tasks are assessed and meet CMMI standards.
4. **Level 3: Defined**

- **Tailored Processes:** Processes are adapted from standard guidelines.
- **All Level 2 Criteria:** Fully meets Level 2 requirements.

5. Level 4: Quantitatively Managed

- **Quantitative Goals:** Uses data to manage quality and performance.
- **All Level 3 Criteria:** Fully meets Level 3 requirements.

6. Level 5: Optimized

- **Continuous Improvement:** Processes adapt to changing needs.
- **Efficiency:** Focus on improving software product efficiency.
- **All Level 4 Criteria:** Fully meets Level 4 requirements.

The CMM Structure

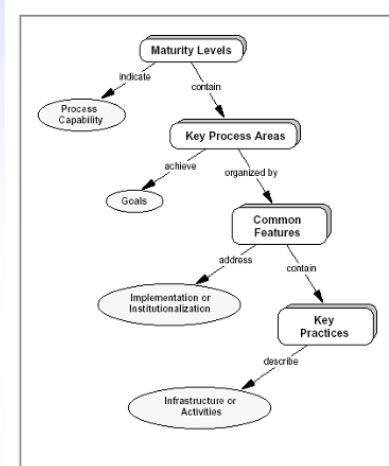


Figure 3.1 The CMM Structure

Function Point (FP)

Function Point (FP) is a metric used to measure the functionality of a system from the user's perspective. It evaluates what users request and what they receive.

Uses of FP Metric:

1. **Cost Estimation:** Estimate the cost or effort to design, code, and test software.
2. **Error Prediction:** Predict the number of errors during testing.
3. **Forecasting:** Estimate the number of components or lines of code in the implemented system.

Information Domain Values for FP Measurement:

1. **External Inputs (EIs)**

- **Definition:** Processes data/control information from outside the system.
- **Note:** Different from inquiries.

2. External Outputs (EOs)

- **Definition:** Generates data/control information sent outside the system.
- **Examples:** Reports, screens, error messages.

3. External Inquiries (EQs)

- **Definition:** Transactions with input and output that retrieve data.
- **Note:** Generates immediate responses.

4. Internal Logical Files (ILFs)

- **Definition:** Groups of related data within the application.
- **Note:** Maintained via external inputs.

5. External Interface Files (EIFs)

- **Definition:** Data maintained by another application.
- **Note:** Considered internal logical files for other systems.

Mnemonic to Remember FP Components:

E, E, E, I, E


- External Inputs (EIs)
- External Outputs (EOs)
- External Inquiries (EQs)
- Internal Logical Files (ILFs)
- External Interface Files (EIFs)

Function Point (FP)

- To compute function points (FP), the following relationship is used:

$$FP = \text{count total} \times [0.65 + 0.01 \times \sum (F_i)]$$

- where count total is the sum of all FP entries

Information Domain Value	Count		Weighting factor			
			Simple	Average	Complex	
External Inputs (EIs)	<input type="text"/>	X	3	4	6	= <input type="text"/>
External Outputs (EOs)	<input type="text"/>	X	4	5	7	= <input type="text"/>
External Inquiries (EQs)	<input type="text"/>	X	3	4	6	= <input type="text"/>
Internal Logical Files (ILFs)	<input type="text"/>	X	7	10	15	= <input type="text"/>
External Interface Files (EIFs)	<input type="text"/>	X	5	7	10	= <input type="text"/>
Count total						= <input type="text"/>

We will have to write on our own [Make assumptions].
Nothing will be given in question.

We will multiple Count with the given weighting factor.

Ex:- 5 simple **External Inputs (EIs)**

3 complex **External Outputs (EOs)**

4 average **External Inquiries (EQs)**

So we, for External Inputs (EIs) the count is 5 and we will multiple 5 with simple i.e. 3

$$5 \times 3 = 15$$

Total count is sum of all.

$$FP = \text{count total} \times [0.65 + 0.01 \times \sum (F_i)]$$

$\sum F_i$ is to be calculated by using 14 factors

The F_i ($i = 1$ to 14) are *value adjustment factors* (VAF) based on responses to the following questions [Lon02]:

1. Does the system require reliable backup and recovery?
2. Are specialized data communications required to transfer information to or from the application?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?
6. Does the system require online data entry?
7. Does the online data entry require the input transaction to be built over multiple screens or operations?
8. Are the ILFs updated online?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

We will have to write all 14 points using assumptions and calculate.

Each of these questions is answered using a scale that ranges from 0 (not important or applicable) to 5 (absolutely essential).

Take sum of all 14 points and that is our $\sum F_i$.

In Function point our main goal is to calculate estimate cost and effort

For cost :-

Estimated number of FP is

Count total * $(0.65 + 0.01 * \sum F_i) = 372$

Historical data indicates that the organizational average productivity for systems of this type is 6.5 FP/per-month and the cost per FP is \$1230.

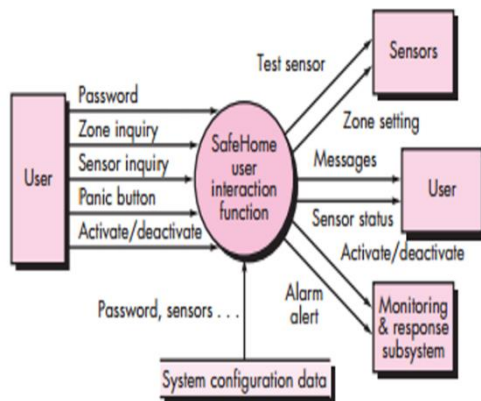
Thus, the FP estimate for this project is

- $372 / 6.5 = 58$ person months
- $372 * 1230 = \$45700$

We just assume anything.

For Calculating cost First Calculate Function point, then just do what the above pic is doing. For productivity they have taken 6.5, but we can take 7 or 8 or 9 also.

If question says also draw Diagram then we draw something like this



What is LOC?

- **Definition:** LOC is a metric used to quantify the size of a software program by counting the number of lines in its source code.
- **Purpose:** It helps in predicting development effort, estimating productivity, and evaluating maintainability.

Key Points:

- **Comment Lines and Headers:** When calculating LOC, lines that contain comments or are part of header files are typically excluded. This helps provide a clearer picture of the actual code that contributes to functionality.

Types of LOC:

1. Physical LOC:

- **Definition:** This counts every line of text in the source code, regardless of whether it's executable or not (including blank lines and comments).
- **Usage:** Useful for a general overview of code size but less effective for estimating functional complexity.

2. Logical LOC:

- **Definition:** This counts only the lines that contain executable statements (i.e., lines that contribute to the program's logic).
- **Usage:** Provides a better measure of the actual functionality and complexity of the code, which can be more useful for estimating development effort.

Why Use LOC?

- **Effort Prediction:** Helps in estimating how much time and resources will be needed for development based on the size of the code.

- **Productivity Measurement:** By comparing LOC to the time taken to write it, you can gauge programmer productivity.
- **Maintainability Assessment:** A higher LOC can sometimes indicate more complexity, which might affect how easy it is to maintain the software.

COCOMO(Constructive Cost Model)

It is used for predicting software development costs, effort.

Sizing Options

COCOMO II requires sizing information, which can be obtained through:

1. **Object Points:** Counts user interface objects (e.g., screens).
 2. **Function Points:** Measures functionality delivered based on user requirements.
 3. **Lines of Source Code (LOC):** Counts the number of lines in the code.
- Object Point is an indirect software measure that is computed using counts of the number
 - (1) screens (at the user interface)
 - (2) reports
 - (3) components likely to be required to build the application
 - Each object instance (e.g., a screen or report) is classified into one of three complexity levels (i.e. Simple, medium, or difficult) using criteria suggested

• New Object Point (NOP)

$$\text{NOP} = (\text{object points}) \times [(100 - \%reuse)/100]$$

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL component			10

- The **object point count** is then determined by multiplying the original number of object instances by the weighting factor and summing to obtain a total object point count.

estimate of effort is based on the computed NOP value, a “productivity rate” s derived

$$\text{PROD} = \frac{\text{NOP}}{\text{person-month}}$$

$$\text{Estimated effort} = \frac{\text{NOP}}{\text{PROD}}$$

The Software Equation

A dynamic multivariable model

$$E = [\text{LOC} \times B^{0.333}/P]^3 \times (1/t^4)$$

where

E = effort in person-months or person-years

t = project duration in months or years

B = “special skills factor”

P = “productivity parameter”

- P = 2,000 for development of real-time embedded software
- P = 10,000 for telecommunication and systems software.
- P = 28,000 for business systems applications
- B=0.16 for programs with LOC 5 to 15
- B=0.35 for programs with LOC greater than 70

Estimation

- To simplify the estimation process set of equations derived from the software equation are stated:
- Minimum development time

$$t_{\min} = 8.14 \frac{LOC}{P^{0.43}} \text{ in months for } t_{\min} > 6 \text{ months}$$

Estimation

- Estimated Effort

$$E = 180 Bt^3 \text{ in person-months for } E \geq 20 \text{ person-months}$$

$$t_{\min} = 8.14 (33200/12000)^{0.43}$$

$$t_{\min} = 12.6 \text{ calendar months}$$

$$E = 180 \times 0.28 \times (1.05)^3$$

$$E = 58 \text{ person-months}$$

Reasons for Project Delays

1. **Unrealistic Deadlines:** Deadlines set by external parties that don't consider the actual development process.
2. **Changing Customer Requirements:** Evolving requirements that aren't reflected in updated schedules.
3. **Underestimated Effort/Resources:** An honest miscalculation of the time and resources needed for the project.
4. **Unconsidered Risks:** Predictable or unpredictable risks not accounted for at the project's start.
5. **Technical Difficulties:** Unexpected technical challenges that arise during development.
6. **Human Difficulties:** Unforeseen human factors, such as team dynamics or turnover.
7. **Miscommunication:** Lack of clear communication among team members leading to delays.
8. **Poor Project Management:** Failure to recognize and address schedule slippage promptly.

Earned Value Analysis (EVA)

1. Earned Value (EV):

- A measure of project progress, indicating the percentage of completion.
- Provides quantitative insights rather than subjective judgments.
- Effective readings can be obtained from as early as 15% into the project.

2. Budget at Completion (BAC):

- Calculated by summing the Budgeted Cost of Work Scheduled (BCWS) for all tasks:

$$BAC = \sum (BCWS_k) \text{ for all tasks}$$

3. Budgeted Cost of Work Performed (BCWP):

- Represents the value of work actually completed:

$$BCWP = \sum (\text{BCWS values for completed tasks})$$

Key Distinction:

- **BCWS (Budgeted Cost of Work Scheduled):** Represents the planned budget for work that was scheduled to be completed.
- **BCWP (Budgeted Cost of Work Performed):** Represents the budgeted value of the work that has actually been completed.

- Given values for BCWS, BAC, and BCWP, important progress indicators can be computed:

$$\text{Schedule performance index, SPI} = \frac{BCWP}{BCWS}$$

$$\text{Schedule variance, SV} = BCWP - BCWS$$

..

$$\text{Percent scheduled for completion} = \frac{BCWS}{BAC}$$

Earned Value Analysis (EVA)

$$\text{Percent complete} = \frac{BCWP}{BAC}$$

- **Actual Cost of Work Performed (ACWP)**

$$\text{Cost performance index, CPI} = \frac{BCWP}{ACWP}$$

$$\text{Cost variance, CV} = BCWP - ACWP$$