Formal Methods Model

- A formal methods model in software engineering uses mathematical techniques to specify, design, and verify software systems, providing a rigorous and unambiguous approach to ensure correctness, reliability, and safety.
- Clean room Software Engineering is a variation of this approach.
- This approach uses formal specification languages to define system characteristics with mathematical precision, eliminating ambiguities and allowing for mathematical analysis and automated verification of the design before implementation.
- Z notation is a formal specification language used in software engineering to model and specify computing systems with mathematical precision.
- It is particularly crucial for safety-critical applications like those in aviation, security, and medicine.

Bottlenecks

- Development of formal model is time-consuming and expensive.
- Needs the S/W engineers to have adequate background and training.
- Also needs the customer to be technically sound to participate on the customer-communication / feedback mechanism.

Fourth Generation Techniques (4GT)

- Fourth Generation Techniques (4GT) in software engineering are a methodology that uses high-level, non-procedural languages and automated tools to generate source code from specifications, focusing on “what” the software should do rather than “how”.
- This approach aims to boost developer productivity and reduce development time by automating tasks such as database queries, report generation, and screen interaction.
- Examples of 4GT tools include SQL, Python, and report generators.

Key aspects of Fourth Generation Techniques

- **High-level specification**: Developers specify software characteristics at a high level, and the tools automatically generate the code.
- **Non-procedural languages**: These languages are closer to human language and focus on defining the desired outcome instead of the step-by-step procedure.
- **Automated code generation**: The process automates the creation of source code based on the provided specifications, which includes tasks like database queries, report generation, and data manipulation.
- **Focus on "what" over "how"**: 4GT shifts the development focus from writing detailed, step-by-step instructions to describing the desired functionality.

Examples of 4GT tools

- Database query languages (eg. SQL)
- Report generators (SAP Crystal Reports, Power BI Report Builder)
- Application generators (Microsoft Power Apps, OutSystems)
- Data manipulation tools (Microsoft Excel Power Query, Informatica PowerCenter)
- Screen interaction tools(Selenium, Cypress)
- Spreadsheet programs(Microsoft Excel, Google Sheets)
- Scripting and high-level languages (e.g., Python, PHP, Perl)

Advantages

- Reduced development time and increased productivity.
- Automation of repetitive tasks.

Disadvantages

- Requirements gathering can be challenging if the customer cannot clearly define them.
- For large projects, design is still a necessary step.

- Software development uses different **process models** to ensure projects are successful.
- Two major approaches are discussed:

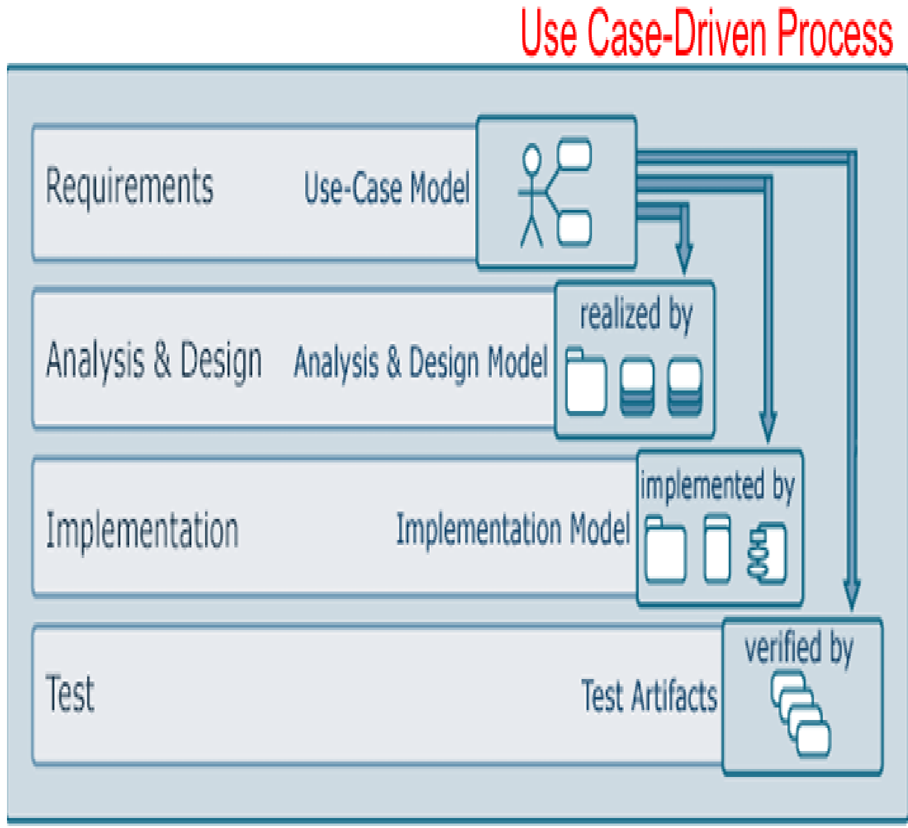
👉 **Rational Unified Process (RUP)** – Traditional, structured

👉 **Agile Methods (XP, Scrum, etc.)** – Flexible, fast

Rational Unified Process

Rational Unified Process (RUP)

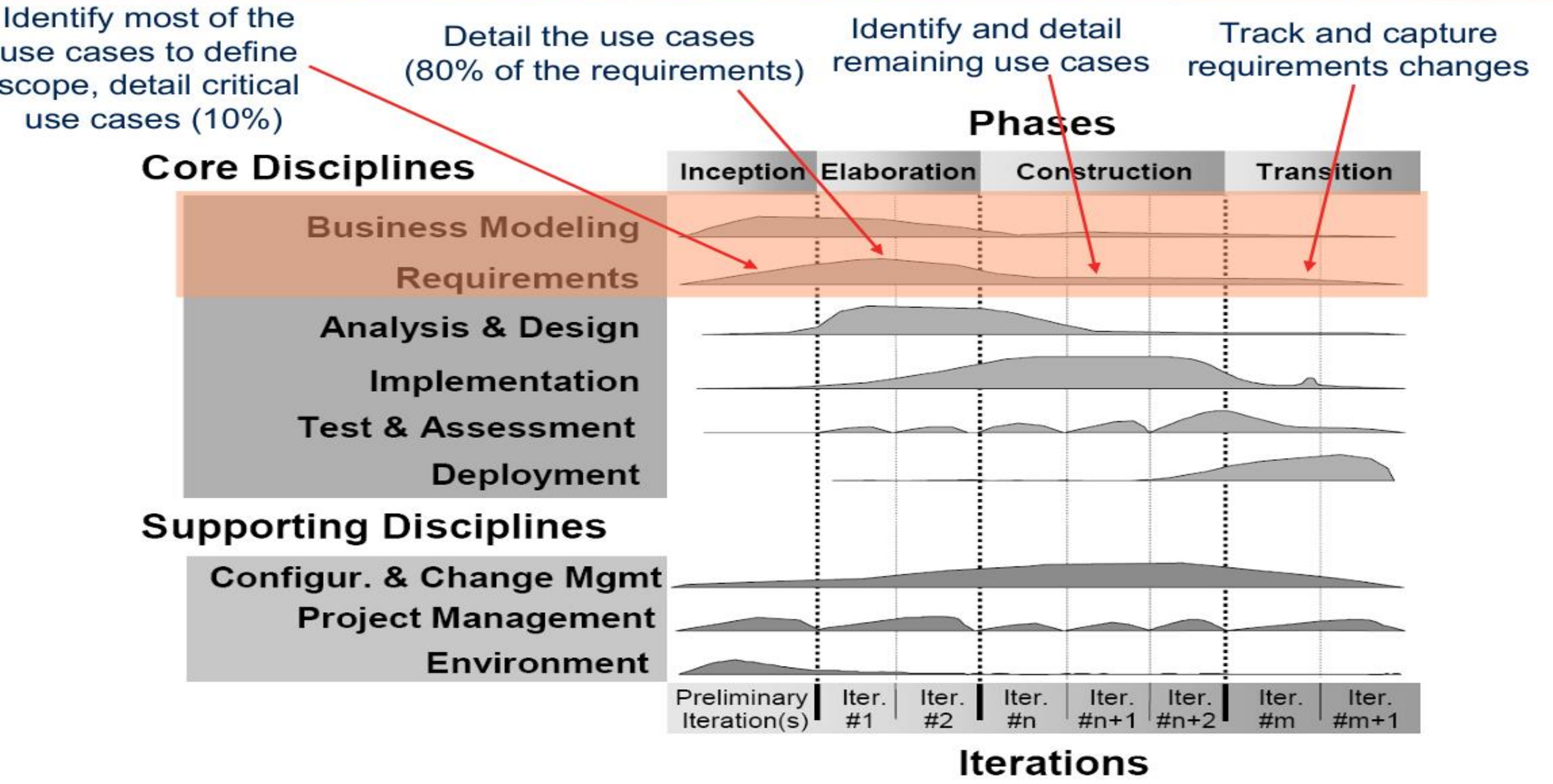
- Rational Unified Process (RUP) is a framework for software engineering processes.
- RUP is a software development **framework** (not a complete process) that helps teams organize work.
- It focuses on **use cases**, **architecture**, and **risk management**, using an **iterative and incremental approach**.
- RUP is proposed by Ivar Jacobson, Grady Bootch, and James Rambaugh *(original designers of the Unified Modeling Language (UML) and are key figures in object-oriented software development)*

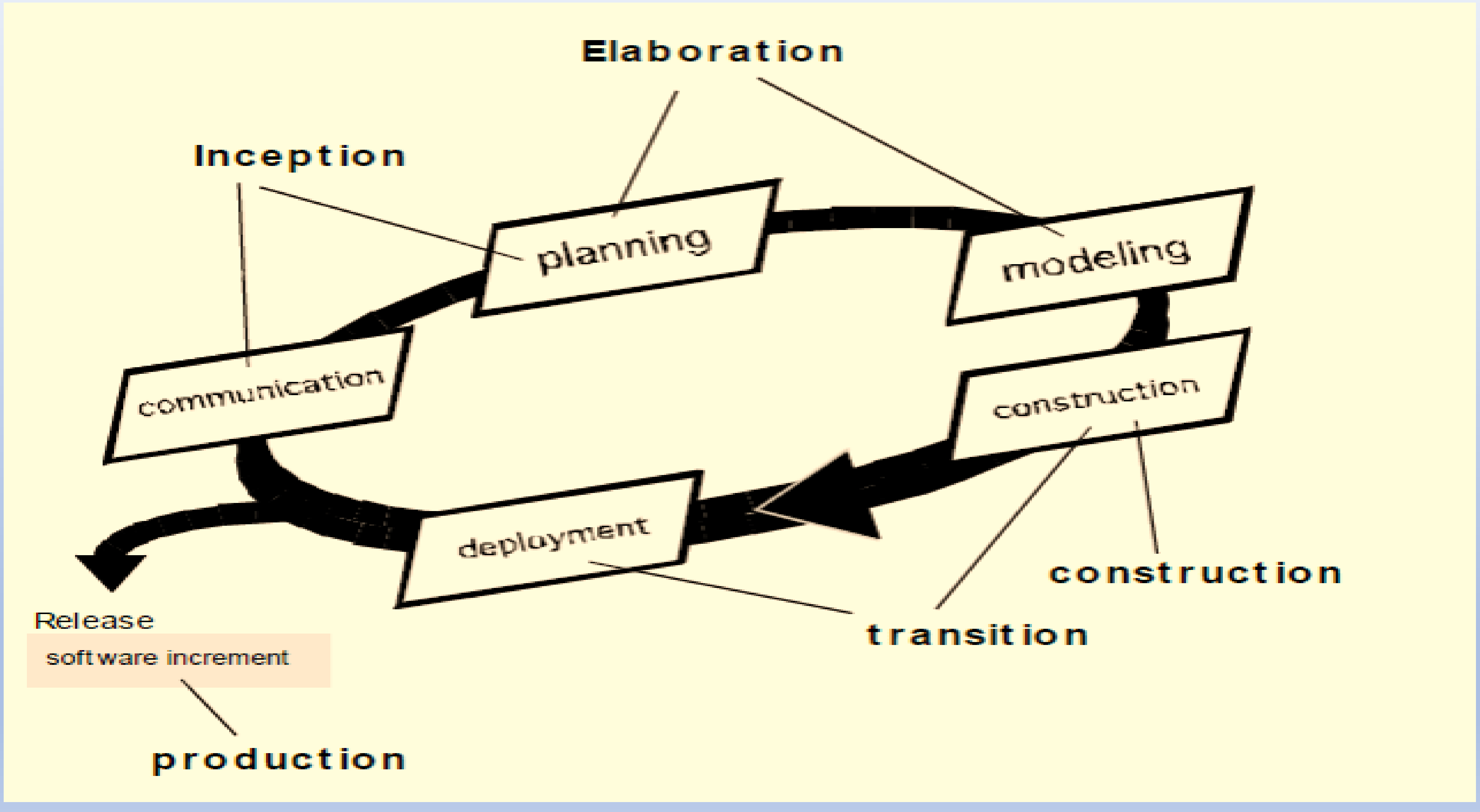


Key Terms and vocabulary

Artifact	Data element produced during the development (document, diagram, report, source code, model…)
Role	• 4 role categories: analyst, developer, tester, manager • Set of skills and responsibilities
Activity	Small, definable, reusable task that can be allocated to a single role
Discipline	• Set of activities resulting in a given set of artefacts • RUP includes 9 disciplines: engineering (business modeling, requirements, analysis and design, implementation, test, deployment) and support (configuration and change management, project management, environment) • Guidance for a discipline is provided as workflows: sequence of activities that produces a result of observable value

Disciplines, Phases, and Iterations




Elaboration
Inception
planning
modeling
communication
construction
deployment
construction
transition
Release
software increment
production

Inception

- Communication and planning are the main ones.
- Identifies the scope of the project using a use-case model allowing managers to estimate costs and time required.
- Customers' requirements are identified and then it becomes easy to make a plan for the project.
- The project plan, Project goal, risks, use-case model, and Project description, are made.
- The project is checked against the milestone criteria and if it couldn’t pass these criteria then the project can be either canceled or redesigned.

Elaboration

- Planning and modeling are the main ones.
- A detailed evaluation and development plan is carried out and diminishes the risks.
- Revise or redefine the use-case model (approx. 80%), business case, and risks.
- Again, checked against milestone criteria and if it couldn’t pass these criteria then again project can be canceled or redesigned.
- baseline for executable architecture.

Construction Phase

- The project is developed and completed.
- System or source code is created and then testing is done.
- Coding takes place.

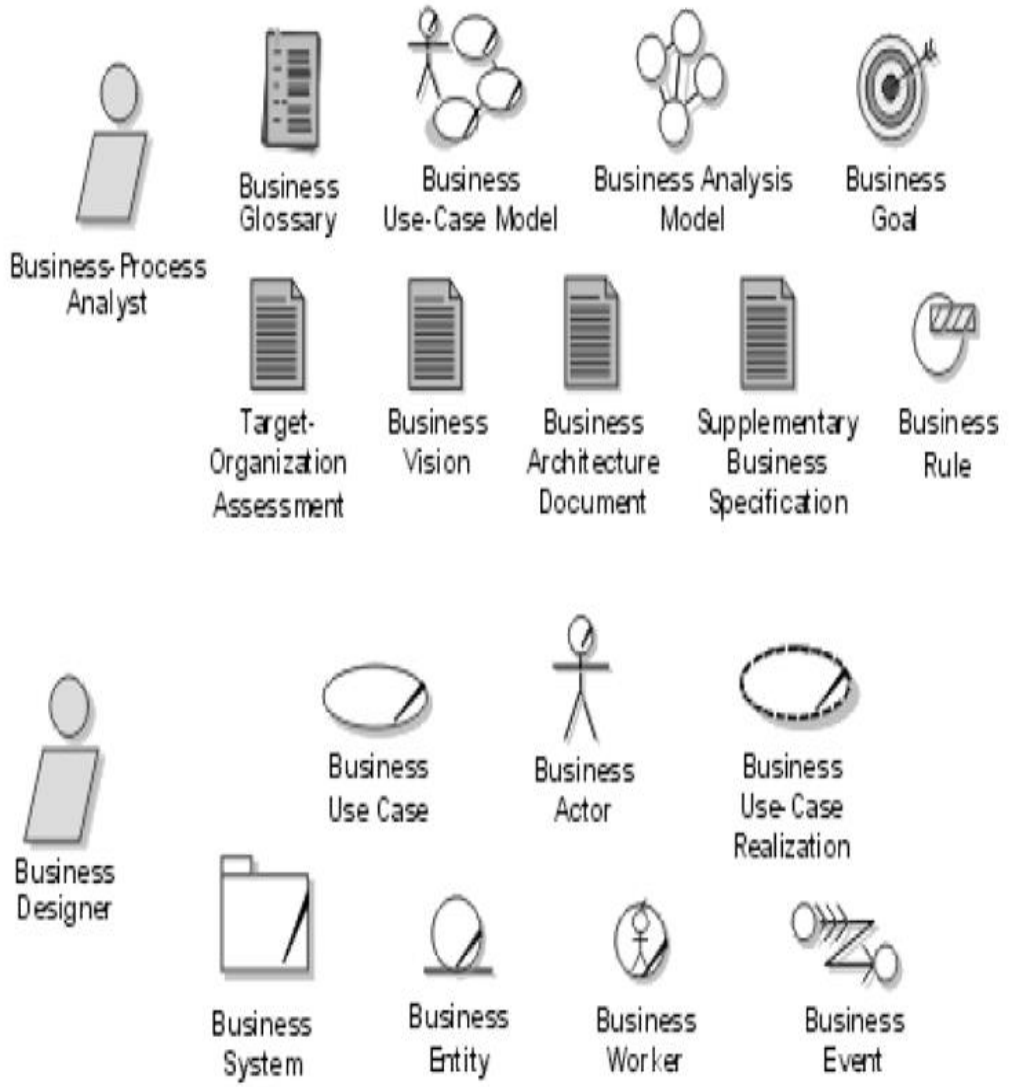
Transition Phase

- The transition phase consists of the transfer of the system to the user community
- Includes manufacturing, shipping, installation, training, technical support, and maintenance
- Development team begins to shrink
- Control is moved to maintenance team
- Alpha, Beta, and final releases
- Software updates
- Integration with existing systems (legacy, existing versions…)
- The project is maintained and updated accordingly.

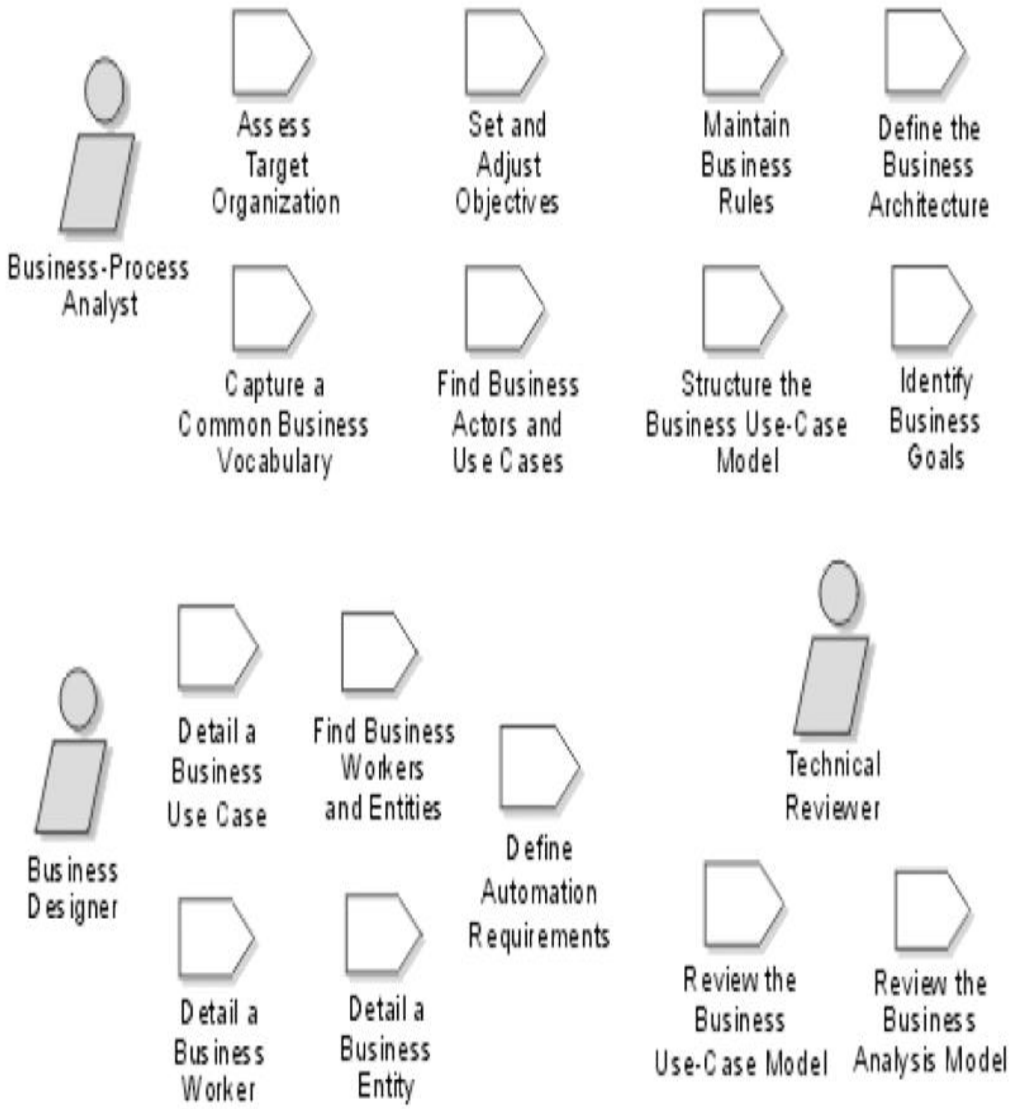
Business Modeling Discipline

- Objectives
  - Understand the structure and the dynamics of the organization in which a system is to be deployed (the target organization)
  - Understand current problems in the target organization and identify improvement potential
  - Ensure that customers, end users, and developers have a common understanding of the target organization
  - Derive the system requirements needed to support the target organization
- Explains how to describe a vision of the organization in which the system will be deployed and how to then use this vision as a basis to outline the process, roles, and responsibilities

Business Modeling Discipline – Artefacts



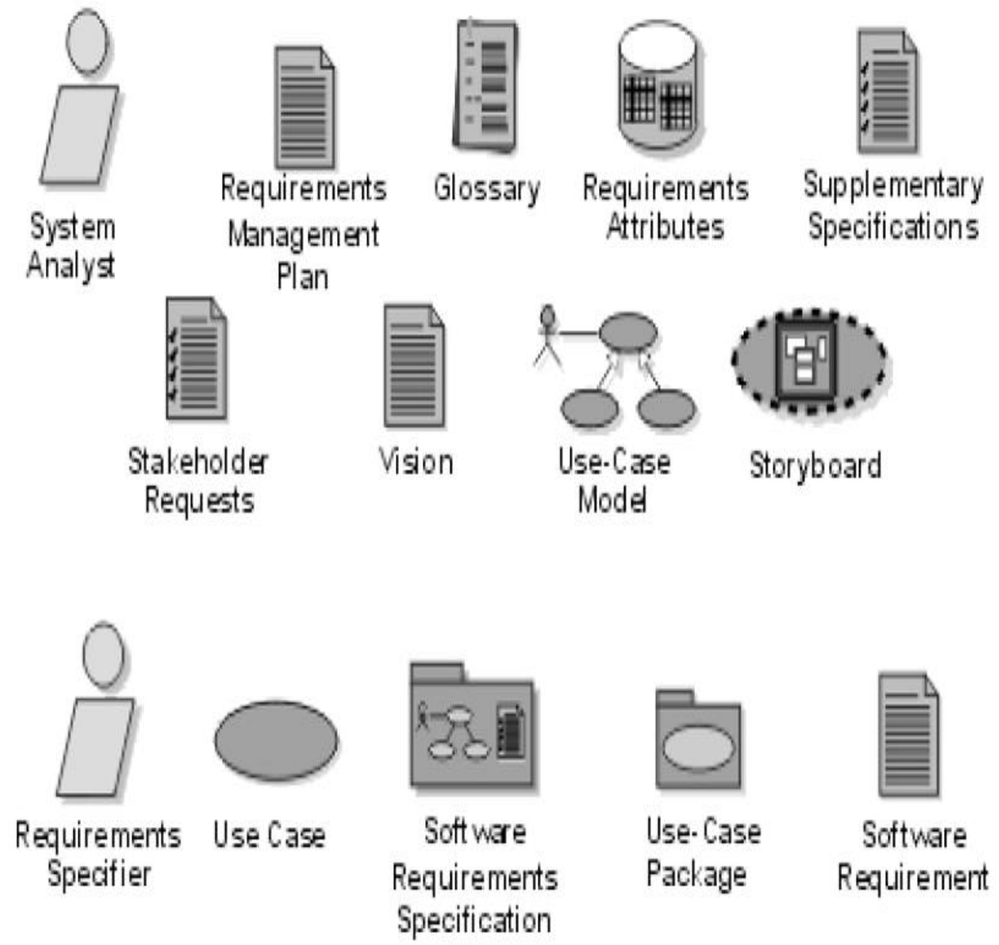
Business Modeling Discipline – Roles


Business-Process Analyst
Assess Target Organization
Set and Adjust Objectives
Maintain Business Rules
Define the Business Architecture
Capture a Common Business Vocabulary
Find Business Actors and Use Cases
Structure the Business Use-Case Model
Identify Business Goals
Business Designer
Detail a Business Use Case
Find Business Workers and Entities
Define Automation Requirements
Technical Reviewer
Detail a Business Worker
Detail a Business Entity
Review the Business Use-Case Model
Review the Business Analysis Model

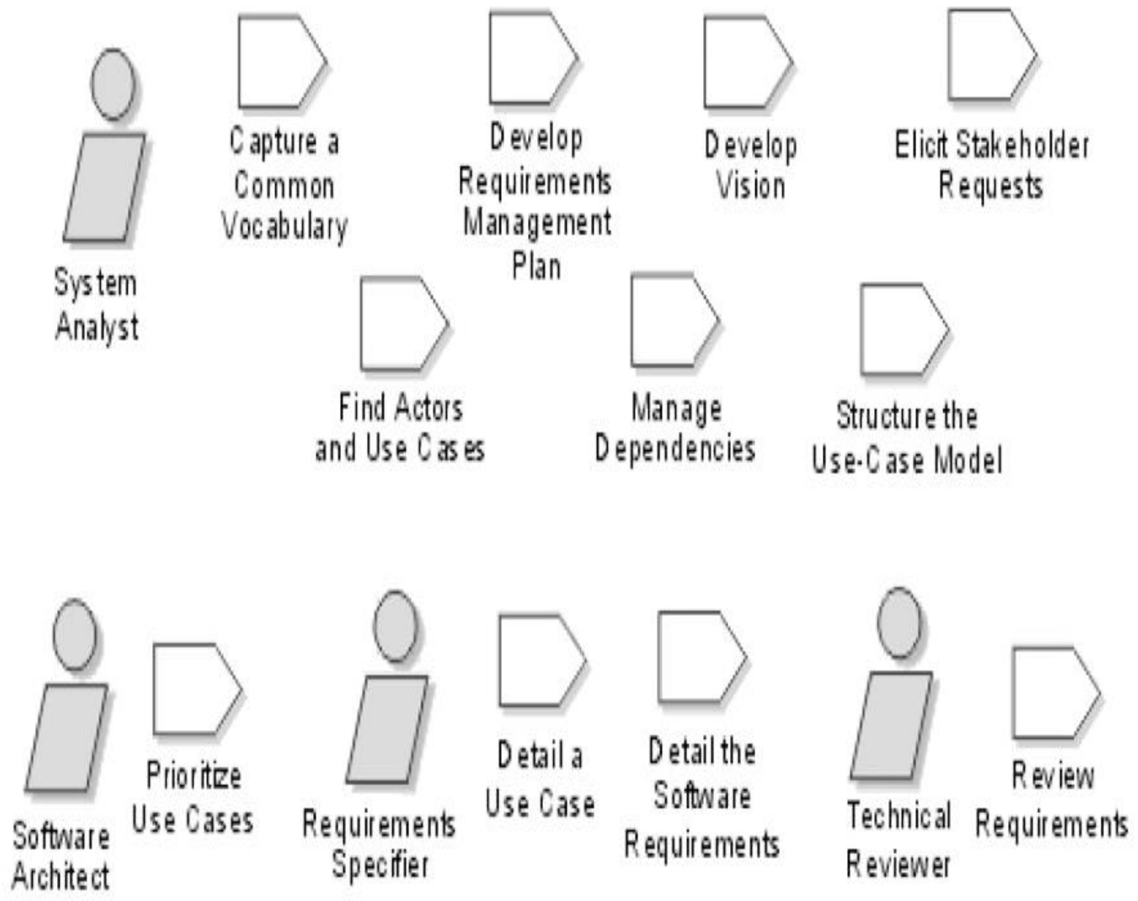
Requirements Discipline

- Establish and maintain agreement with the customers and other stakeholders on what the system should do
- Provide system developers with a better understanding of the system requirements
- Define the boundaries of the system
- Provide a basis for planning the technical contents of iterations
- Provide a basis for estimating the cost and time to develop the system

Requirements Discipline – Artefacts



Requirements Discipline – Roles


System Analyst
Capture a Common Vocabulary
Develop Requirements Management Plan
Develop Vision
Elicit Stakeholder Requests
Find Actors and Use Cases
Manage Dependencies
Structure the Use-Case Model
Software Architect
Prioritize Use Cases
Requirements Specifier
Detail a Use Case
Detail the Software Requirements
Technical Reviewer
Review Requirements

Requirements Discipline – Tasks

- Includes the following tasks
  - List candidate requirements
    - Candidate features that could become requirements
  - Understand system context
    - Based on business model, domain model or simple glossary
  - Capture functional requirements
    - Develop use cases and user interface support of use cases
  - Capture non-functional requirements
    - Tied to use cases or domain concepts
    - Defined as supplementary requirements
  - Validate requirements

Other Disciplines – Engineering (1)

- Analysis and Design Discipline
  - Transform the requirements into a design of the system-to-be
  - Evolve a robust architecture for the system
  - Adapt the design to match the implementation environment
- Implementation Discipline
  - Define the organization of the implementation
  - Implement the design elements
  - Unit test the implementation
  - Integrate the results produced by individual implementers (or teams), resulting in an executable system

Other Disciplines – Engineering (2)

- Test Discipline
  - Find and document defects in software quality
  - Provide general advice about perceived software quality
  - Prove the validity of the assumptions made in design and requirement specifications through concrete demonstration
  - Validate that the software product functions as designed
  - Validate that the software product functions as required (that is, the requirements have been implemented appropriately)
- Deployment Discipline
  - Ensure that the software product is available for its end users

Supporting Disciplines

- Configuration & Change Management Discipline
  - Identify configuration items
  - Restrict changes to those items
  - Audit changes made to those items
  - Define and manage configurations of those items
- Project Management Discipline
  - Manage a software-intensive project; manage risk
  - Plan, staff, execute, and monitor a project
- Environment Discipline
  - Provide the software development organization with the software development environment – both processes and tools – that will support the development team
  - This includes configuring the process for a particular project, as well as developing guidelines in support of the project

RUP Best Practices

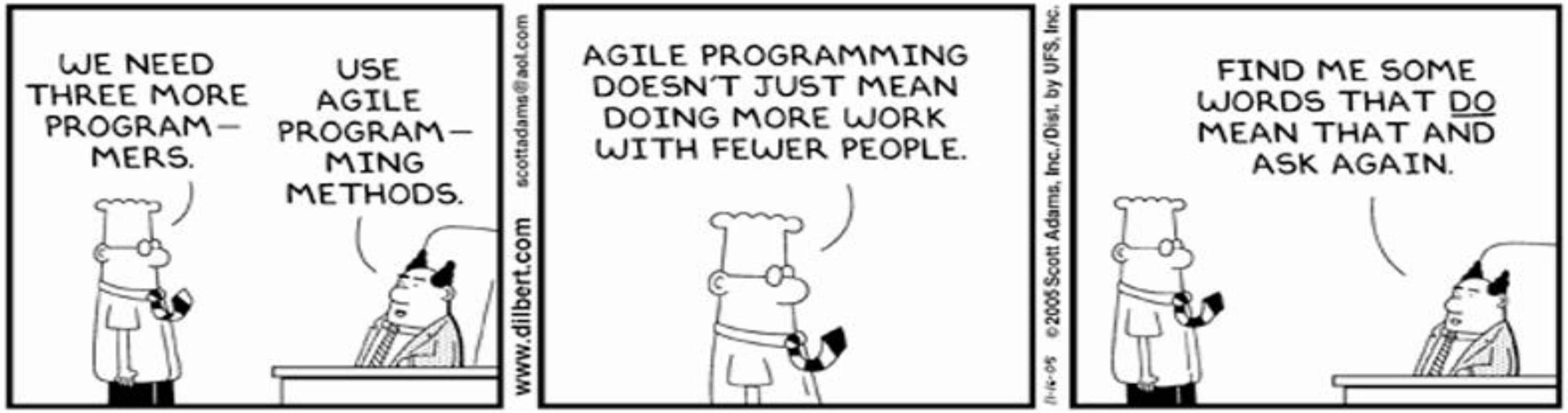
The best practices for the Rational Unified Process (RUP) are:

- RUP gives guidelines for making good documents and using the right process.
- Development is done in steps (iterations), improving the software each time.
- Requirements are managed using tools and use cases to clearly understand customer needs.
- The software is built from reusable parts (components) to support easier testing and reuse.
- UML diagrams are used for visual modeling.
- Quality is checked all the time by reviews and measuring results.
- Changes are managed properly to keep track of baselines and requests.

These practices help make software projects well-organized and high-quality

Agile Methods

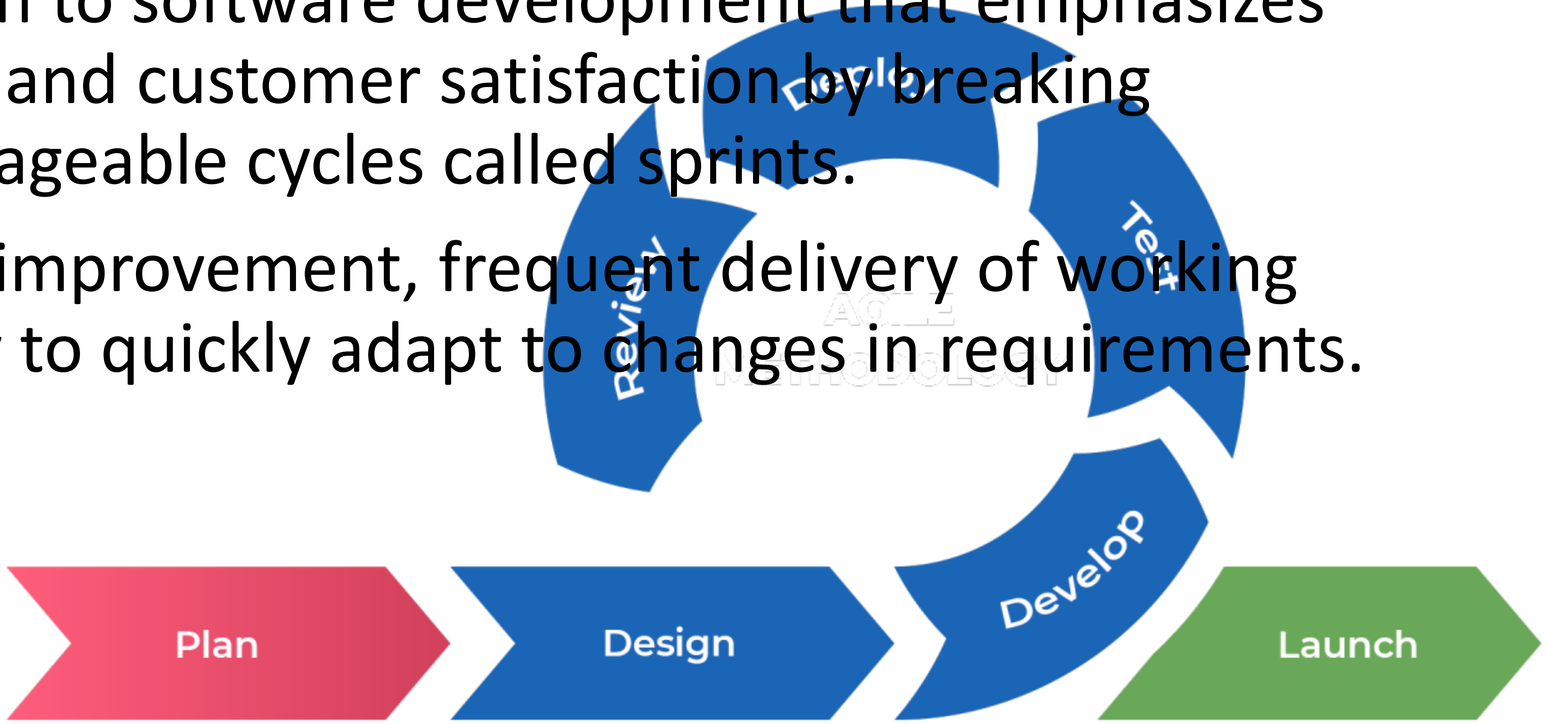
Agile Methods


WE NEED THREE MORE PROGRAM- MERS.
USE AGILE PROGRAM- MING METHODS.
AGILE PROGRAMMING DOESN'T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.
FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.
scottadams@aol.com
www.dilbert.com
© 2005 Scott Adams, Inc./Dist. by UFS, Inc.

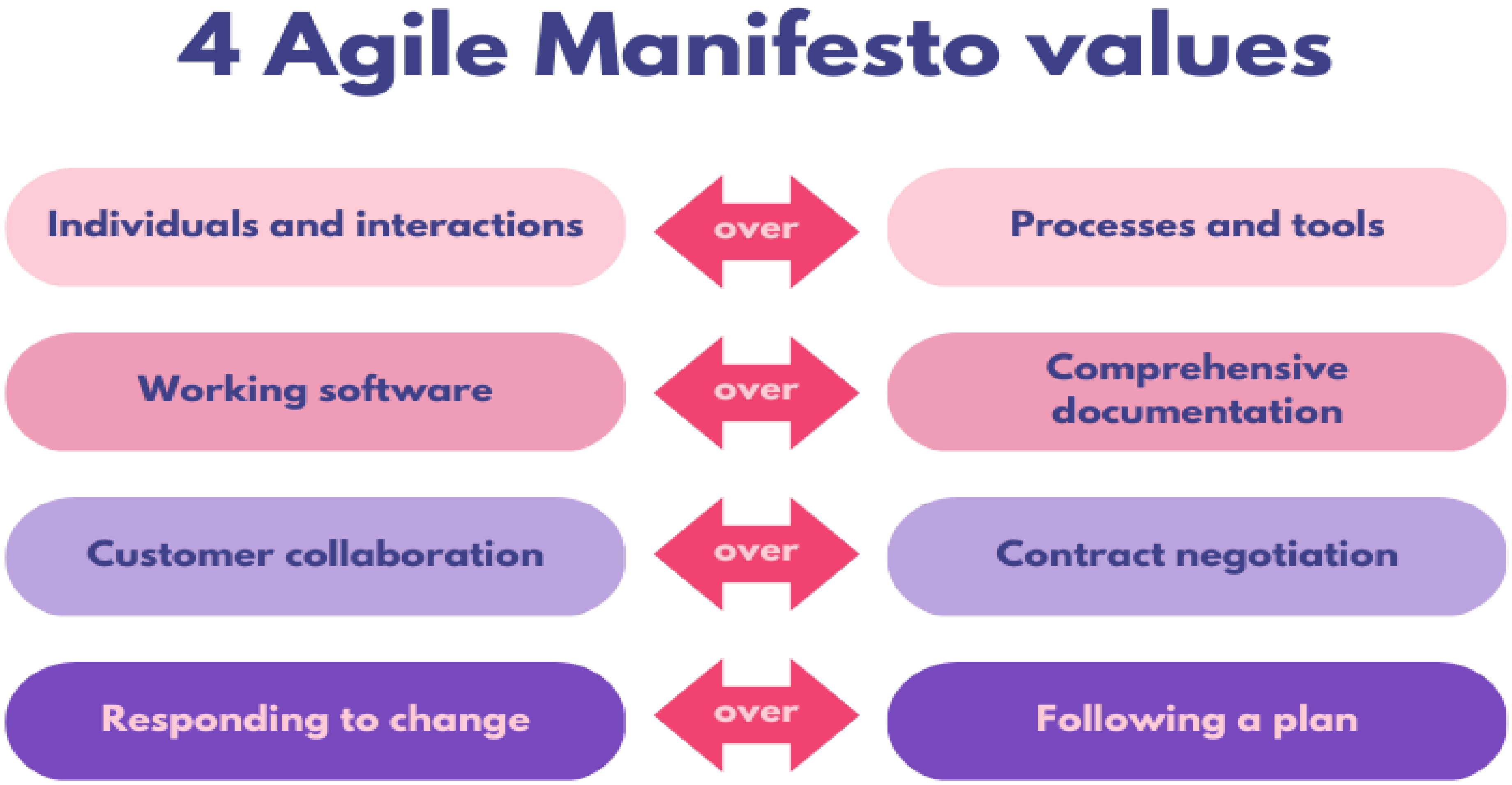
WE'RE GOING TO TRY SOMETHING CALLED AGILE PROGRAMMING.
THAT MEANS NO MORE PLANNING AND NO MORE DOCUMENTATION. JUST START WRITING CODE AND COMPLAINING.
I'M GLAD IT HAS A NAME.
THAT WAS YOUR TRAINING.
scottadams@aol.com
www.dilbert.com
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.

Agile Methods

- In software development, the term “agile” means “the ability to respond to changes”(move quickly)- changes from requirements, Technology and people.
- It is an iterative approach to software development that emphasizes flexibility, collaboration, and customer satisfaction by breaking projects into small, manageable cycles called sprints.
- It allows for continuous improvement, frequent delivery of working software, and the ability to quickly adapt to changes in requirements.



Values of Agile Methods



4 Agile Manifesto values

Individuals and interactions	over	Processes and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Values of Agile Methods

- **Individuals and interactions over processes and tools:**

➢This value prioritizes the people doing the work and how they communicate with each other. While processes and tools are useful, effective teamwork and communication are considered more important for success.

- **Working software over comprehensive documentation:**

➢Agile methodologies focus on delivering functional software as the primary measure of progress. While documentation is still necessary, it should not take priority over the creation of a working product.

Values of Agile Methods

- **Customer collaboration over contract negotiation:**

➢Agile emphasizes continuous, collaborative engagement with the customer throughout the project. This contrasts with traditional methods that rely heavily on detailed contracts and aims to ensure the final product meets the customer's evolving needs.

- **Responding to change over following a plan:**

➢Agile teams are built to be flexible and adaptable. They are encouraged to embrace and respond to change rather than rigidly sticking to a predefined plan.

Principles of the Agile Manifesto (1)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development – agile processes harness change for the customer's competitive advantage
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter period
- Business people and developers must work together daily throughout the project
- Build projects around motivated individuals – give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Principles of the Agile Manifesto (2)

- Working software is the primary measure of project progress
- Agile processes promote a sustainable pace of development – the sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity – the art of maximizing the amount of work not done – is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Examples of Agile Approaches

- Extreme Programming (XP)
- Scrum
- Open Unified Process (OpenUP)
- Adaptive Software Development (ASD)
- Crystal Clear
- DSDM
- Feature Driven Development
- Lean software development
- Agile documentation
- Agile ICONIX
- Microsoft Solutions Framework (MSF)
- Agile Data
- Agile Modeling
- Agile Unified Process (AUP)
- Essential Unified Process (EssUP)
- Getting Real
- …

see http://en.wikipedia.org/wiki/Agile_software_development

When to use in general?

- **Requirements are uncertain or expected to change**:

Agile embraces changing requirements, even late in development, to ensure the final product remains relevant and valuable in a dynamic market.

- **Rapid delivery is a priority**:

The iterative nature of Agile allows for the frequent delivery of working software (every few weeks) so that value reaches the customer quickly.

- **Customer collaboration is essential**:

Agile requires close, continuous collaboration between the development team and business representatives/customers to ensure the product meets their evolving needs.

- **Projects involve a high level of uncertainty or risk**:

By breaking down complex projects into smaller, manageable chunks, risks can be identified and addressed early on, preventing small issues from snowballing into larger problems.

- **Cross-functional and self-organizing teams are available**:

Agile relies on motivated, cross-functional teams that can work collaboratively and make decisions without rigid hierarchies.

- **Continuous improvement is valued**:

Regular reflection (retrospectives) on the process allows teams to learn from their experiences and adjust their behavior accordingly, fostering an environment of ongoing improvement.

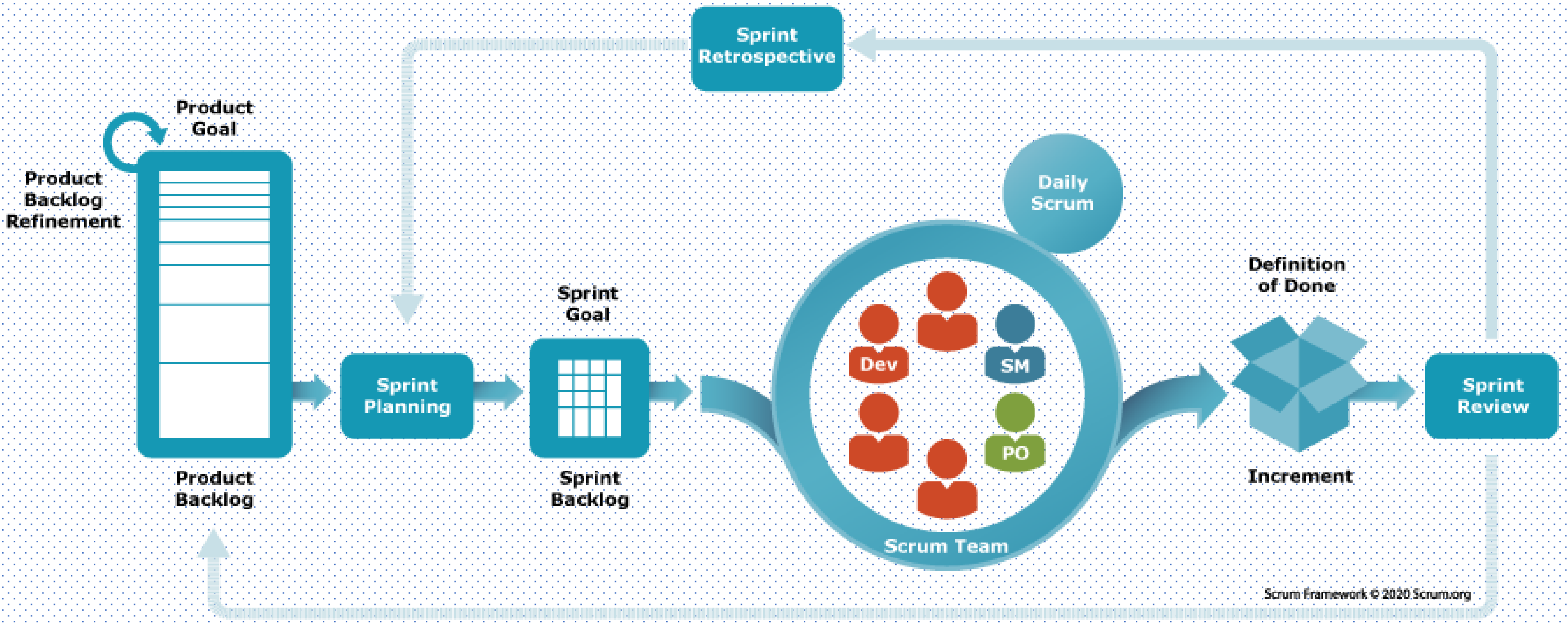
When not to use?

- **Requirements are fixed and well-defined**: For projects with stable, clearly documented requirements (like certain construction or government software projects), a traditional approach like Waterfall may be more efficient.
- **Extensive upfront documentation is required**: Agile prioritizes working software over comprehensive documentation, which might be a problem in highly regulated industries that mandate detailed paperwork.
- **Customers cannot participate actively**: The success of Agile heavily depends on regular customer feedback and involvement; if customers are unavailable or unwilling to collaborate, the process can struggle.

Scrum

- Scrum is an agile framework used for managing complex projects, particularly in software development.
- It involves a team working in short, time-boxed iterations called "sprints" to deliver a functional piece of a product.
- The framework uses specific roles, events, and artifacts to promote collaboration, transparency, and continuous improvement through inspection and adaptation.

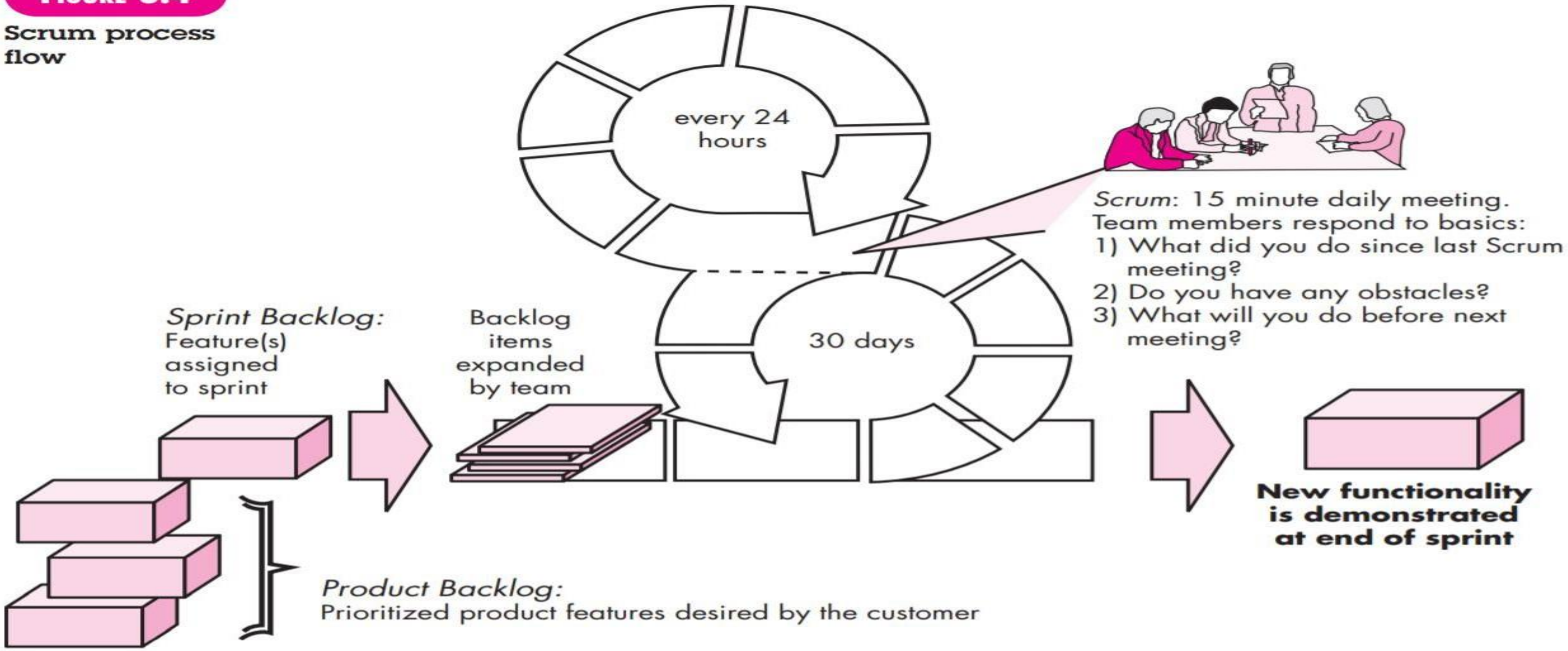
Scrum


Sprint Retrospective
Product Goal
Product Backlog Refinement
Daily Scrum
Definition of Done
Sprint Goal
Sprint Planning
Dev
SM
PO
Sprint Review
Product Backlog
Sprint Backlog
Increment
Scrum Team
Scrum Framework © 2020 Scrum.org

scrum

FIGURE 3.4

Scrum process flow



every 24 hours
Scrum: 15 minute daily meeting.
Team members respond to basics:
1) What did you do since last Scrum meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Key components of Scrum

- **Artifacts:**

➢ *Product Backlog*: A prioritized list of all desired features for the product.

➢ *Sprint Backlog*: The set of work the development team has committed to completing during a sprint.

- **Sprints:** Time-boxed, iterative cycles (typically one to four weeks) during which a team creates a potentially releasable product increment.
- **Roles:**

➢ *Scrum Master*: A team leader, Scrum master, leads the meeting and assesses the responses from each person. Facilitates the process, removes obstacles, and ensures the team follows Scrum principles.

➢ *Product Owner*: Represents the customer and is responsible for the product backlog and its priorities.

➢ *Development Team*: A cross-functional group that builds the product.

Key components of Scrum

- **Events**:

➢ *Daily Scrum*: A short daily meeting for the development team to synchronize activities.

➢ *Sprint Planning*: The team plans the work for the upcoming sprint.

➢ *Sprint Review*: The team demonstrates the increment and discusses progress with stakeholders.

➢ *Sprint Retrospective*: a meeting in the Agile methodology held at the end of a sprint for the entire team to reflect on their work and identify ways to improve.

Key components of Scrum

- **Demos**:

➢Deliver the software increment to the customer so that functionality that has been implemented can be demonstrated and evaluated by the customer.

➢Demo may not contain all planned functionality, but rather those functions that can be delivered within the time-box that was established

Advantages of Scrum

- Freedom and adaption(able to adopt with the new changes)
- High quality and low risk product
- Reduce the development time up to 40%
- Reviewing the current sprint before moving to new phases

Disadvantages of Scrum

- Not efficient for large team
- No changes in the sprint (change is made in another sprint)

Extreme Programming (XP)

- One of the widely used approach of agile software development.
- Extreme Programming (XP) is an Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation.
- XP uses a set of five values that establish a foundation for all work per formed:

– Communication

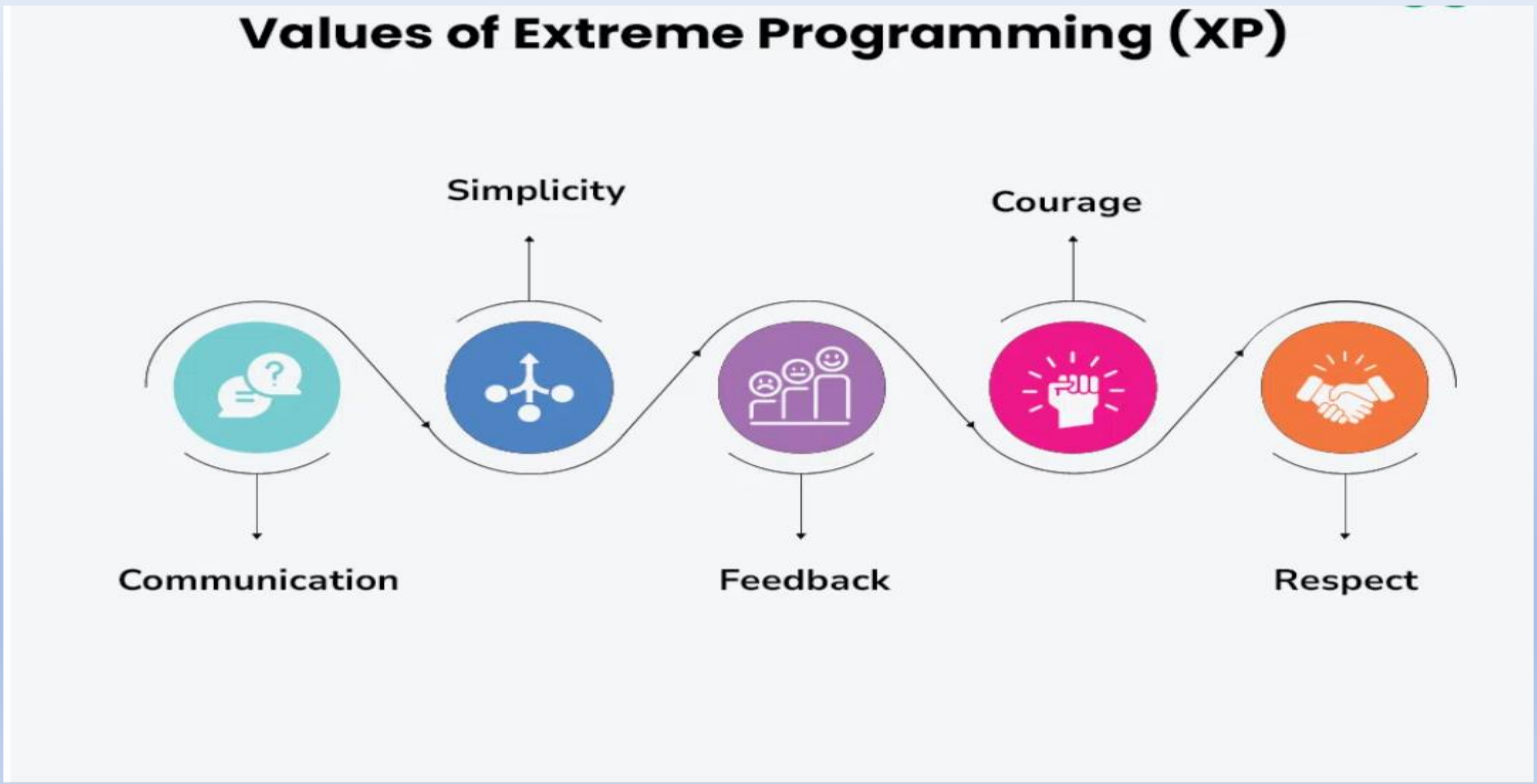
– Simplicity

– Feedback

– Courage

– Respect

Extreme Programming (XP)



Communication

- **Communication:** Promotes open, frequent, and face-to-face communication among team members and with customers to ensure a shared understanding of the project's goals and requirements.
- For effective communication between software engineers and other stakeholders XP emphasizes close, yet informal (verbal) collaboration between customers and developers.
- The establishment of effective metaphors for communicating important concepts, continuous feedback, and the avoidance of voluminous documentation as a communication medium
- a metaphor is “a story that everyone— customers, programmers, and managers— can tell about how the system works”

Simplicity

- To achieve simplicity, XP restricts developers to design only for immediate needs rather than consider future needs.
- The intent is to create a simple design that can be easily implemented in code
- If the design must be improved, it can be refactored at a later time

Feedback

- Feedback is derived from three sources: the implemented software itself, the customer, and other software team members.
- Team members deliver software frequently, get feedback about it, and improve a product according to the new requirements.
- By designing and implementing an effective testing strategy the software (via test results) provides the agile team with feedback.
- XP makes use of the unit test as its primary testing tactic

Courage

- Encourages the team to be courageous enough to make bold decisions, speak up about problems, tell the truth about progress, and adapt to changes without fear.
- For example, there is often significant pressure to design for future requirements.
- Most software teams succumb, arguing that “designing for tomorrow” will save time and effort in the long run.
- An agile XP team must have the discipline (courage) to design for today, recognizing that future requirements may change dramatically, thereby demanding substantial rework of the design and implemented code.

Respect

- Fosters a culture where every team member's input is valued and respected, creating a supportive and collaborative environment where everyone can contribute their best.

The XP Process

- Encompasses a set of rules and practices that occur within the context of four framework activities:

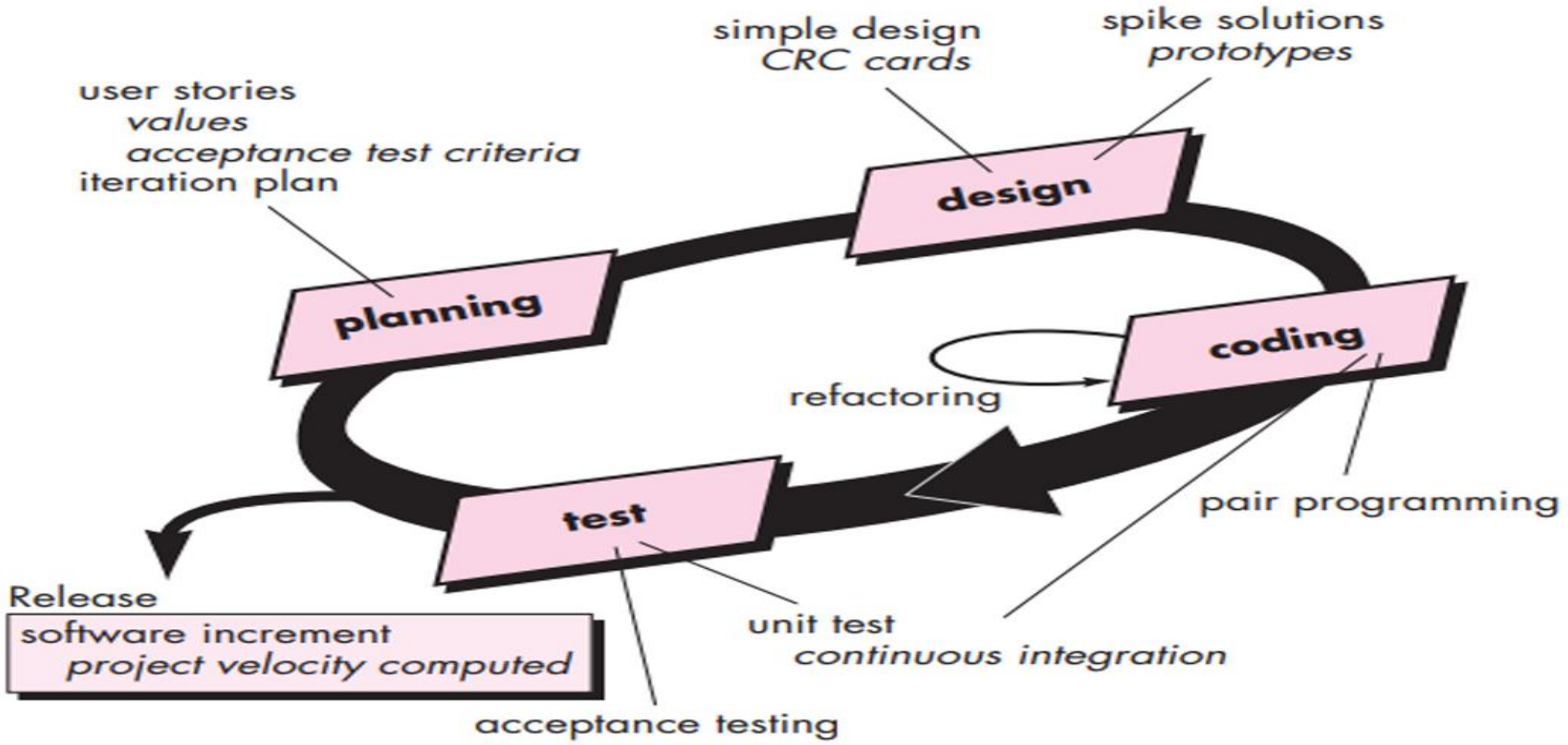
– planning

– design

– coding

– testing.

The XP Process



Planning

- The planning activity also called the **planning game** begins with **listening**—a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to get a broad feel for required output and major features and functionality.
- Listening leads to the creation of a set of "**stories**" also called **user stories** that describe required output, features, and functionality for software to be built
- Each story similar to use is written by the customer and is placed on an index card.
- The customer assigns a value i.e a priority to the story based on the overall business value of the feature or function.

Planning

- Members of the XP team then assess each story and assign a cost—measured in development weeks—to it.
- If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.
- It is important to note that new stories can be written at any time.
- Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team.

Planning

- Once a basic commitment (agreement on stories to be included, delivery date, and other project matters) is made for a release, the XP team orders the stories that will be developed in one of three ways:

1.all stories will be implemented immediately (within a few weeks),

2. the stories with highest value will be moved up in the schedule and implemented first

3. the riskiest stories will be moved up in the schedule and implemented first.

Design

- XP design rigorously follows the KIS (keep it simple) principle.
- A simple design is always preferred over a more complex representation.
- The design of extra functionality (because the developer assumes it will be required later) is discouraged.
- XP encourages the use of CRC cards as an effective mechanism for thinking about the software in an object-oriented context.
- CRC (class-responsibility collaborator) cards identify and organize the object-oriented classes that are relevant to the current software increment.

Design

- If a difficult design problem is encountered as part of the design of a story, XP recommends the immediate creation of an operational prototype of that portion of the design Called a spike solution
- XP encourages refactoring—a construction technique that is also a method for design optimization
- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves the internal structure

Coding

- After stories are developed and preliminary design work is done, the team does not move to code, but rather develops a series of unit tests
- Once the unit test has been created, the developer is better able to focus on what must be implemented to pass the test.
- Nothing extraneous is added (KIS).
- Once the code is complete, it can be unit-tested immediately, thereby providing instantaneous feedback to the developers

Coding

- A key concept during the coding activity is pair programming.
- XP recommends that two people work together at one computer workstation to create code for a story.
- This provides a mechanism for real time problem solving (two heads are often better than one) and real-time quality assurance (the code is reviewed as it is created
- It also keeps the developers focused on the problem at hand. In practice, each person takes on a slightly different role.
- One person – coding details of particular design
- Another person – Coding standards

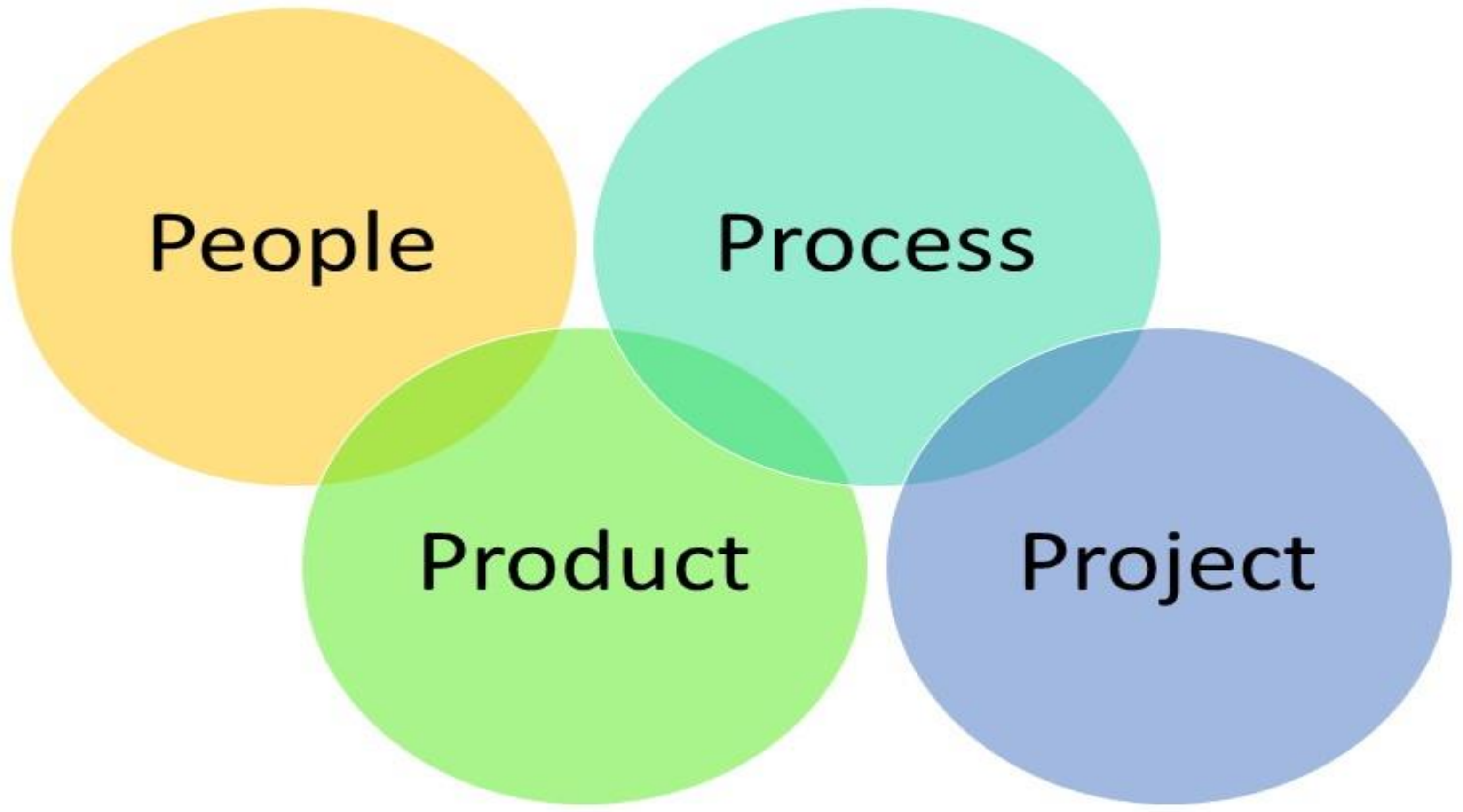
Testing

- There creation of **unit tests** before coding commences is a key element of the XP approach.
- The unit tests that are created should be implemented using a **framework** that enables them to be **automated** (hence, they can be executed easily and repeatedly).
- This encourages a **regression testing** strategy whenever code is modified (which is often, given the XP refactoring philosophy).
- As the individual unit tests are organized into a “**universal testing suite**”, integration and validation testing of the system can occur on a daily basis.

Testing

- This provides the XP team with a continual indication of progress and also can raise warning flags early if things go away.
- “Fixing small problems every few hours takes less time than fixing huge problems just before the deadline.”
- **XP acceptance tests**, also called **customer tests**, are specified by the customer and focus on overall system features and functionality that are visible and reviewable by the customer.
- Acceptance tests are derived from user stories that have been implemented as part of a software release.

4 P's



4 P's of Management Spectrum

4 P's

- The "4 P's" in software engineering are **People**, **Product**, **Process**, and **Project**, which together form the management spectrum for software projects.
- They represent the key components of software project planning, requiring the project manager to balance each element to achieve project goals effectively.

4 P's

- **People:** The most important component of a product and its successful implementation is human resources.
- In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product.
- We need to have a good team in order to save our time, cost, and effort.
- Some assigned roles in software project planning are **project manager, team leaders, stakeholders, analysts,** and other **IT professionals**.
- Managing people successfully is a tricky process which a good project manager can do.
- This involves the project team, stakeholders, and anyone else involved in the project, including their roles, responsibilities, and communication.

4 P's

- **Product:** This refers to the actual software being developed, encompassing its features, functionality, and what it will deliver to users and stakeholders.
- The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product.
- **Process:** In every planning, a clearly defined process is the key to the success of any product.
- This covers the methodologies, workflows, and procedures used to manage and control the development and delivery of the software.

4 P's

- **Project:** The last and final P in software project planning is Project.
- It can also be considered as a blueprint of process.
- In this phase, the project manager plays a critical role.
- They are responsible to guide the team members to achieve the project's target and objectives, helping & assisting them with issues, checking on cost and budget, and making sure that the project stays on track with the given deadlines.
- This refers to the overall management of the project, including its planning, budget, timeline, risks, and the integration of the other three P's.