CMP 2101: Software Engineering

Lecture 02: Software Processes

1.1. Software Process

A software process is a structured set of activities required to develop a software system. It is needed to transform a user's *requirements* into a *software system*.

1.1.1. Process Attributes

- **Understandability** - To what extent is the process explicitly defined and how easy is it to understand?
- **Visibility** - Do the process activities culminate in clear results so that the progress of the process is externally visible?
- **Supportability** - To what extent can the process activities be supported by CASE tools?
- **Acceptability** - acceptable and usable by engineers?
- **Reliability** - Is the process designed in such a way that process errors are avoided or trapped before they result in product errors?
- **Robustness** - Can the process continue in spite of unexpected problems?
- **Maintainability** - Can the process evolve to reflect changing organizational requirements or identified process improvements?
- **Rapidity** - How fast can the process of delivering a system from given specification be completed?

1.1.2. Software Process Activities

- **Requirements Engineering**: This phase involves finding out, understanding and documenting what the client wants the software to do; Specifically the activities include: Feasibility study, Requirements Elicitation and Analysis, Requirements Validation, Requirements Specification
- **Design** – This stage involves planning the software solution; modelling data structures, software architecture, interface representations, algorithmic details. This stage translates requirements into a representation of the software.
- **Implementation** – Mainly involves coding and user documentation.
- **Testing -**This phase is usually coupled with implementation and it involves activities for checking for bugs, verification and validation, integration, and quality assurance.
- **Software Evolution/Maintenance** – this involves adjusting of changing the existing software product after delivery.

1.2. Process Models

Process models are general approaches for organizing a project into activities. They are a strategy employed in software engineering that encompasses or specifies the process, tools and methods that support the engineering process. Process models for software engineering are chosen depending on:

- **nature of the project or application** at hand; large and complex, small, enterprise, custom e.t.c;
- **methods** ; technical how-to's for building software; a broad array of tasks that include requirements analysis, design, program construction, testing, and support;
- and **tools** (automated and semi-automated)to be used and the controls and deliverables required.

1.2.1. Application of Process Models

- They organize software development and thus increase the quality of a process
- Process models represent:
  - Repeatability: can someone re-run through the steps in this process?
  - Manageability and measurability: are milestones and timelines visible? Is the development still within scope? What should be done? Which sequence should be followed?
- They more clearly define and justify interim work products; since there are well and pre-defined deliverables, it enables easy tracking of progress.
- Using process models doubles as an activity inventory/reminder and therefore one is 'less likely to forget activities'.

1.3.The Software Development Life Cycle and Process Models

1.3.1. Waterfall Model

This process model, also called the classical lifecycle is a traditional approach that was first described in 1970s for aerospace and defence projects. Ideally, it is a linear sequential model that suggests a systematic, sequential approach to software development progressing through analysis, design, implementation, testing and maintenance. However in practice the stages might overlap, and feed information to each other: during design, problems in requirements are identified; during coding, design problems are found and so on.

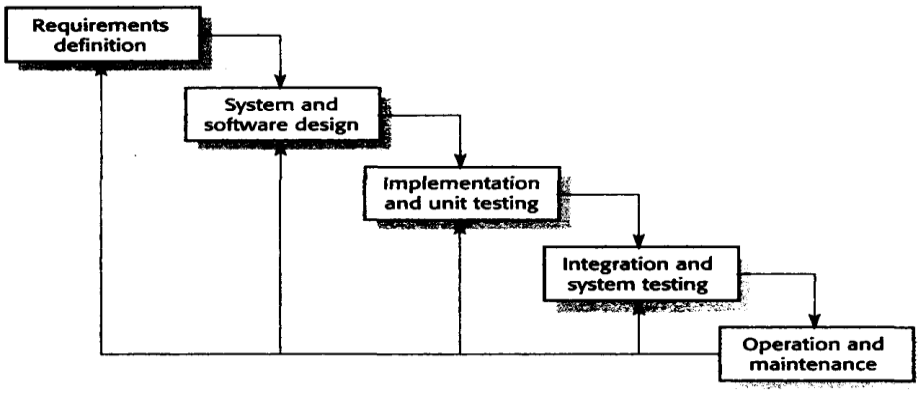

Figure 1: The Waterfall Model with feedback

Where the Waterfall model is Applicable:

- On projects with few requirements changes (stable product definition).
- Where requirements are well understood.
- When the technology to be used is well understood.

- When developing a new version of an existing product.
- When porting an existing product to a new platform.

Strengths

- It allows for compartmentalization and managerial control as scheduling is easily set.
- Milestones are easily monitored and managed using step deliverables.
- Works well for small projects.
- It is easy to use and understand
- It provides structure for inexperienced staff

Weaknesses

- The waterfall method is inflexible as ideally it discourages revisiting and revising a phase once it's complete.
- Highly risky and not good for long or complex models which would normally require re-visiting of phases.
- It is usually difficult for the client to state all the requirements explicitly and so requirements may not be known well, thus it is not good for changing requirements.
- A working version of the system is only available late in the project time-span, thus requiring customer patience as they cannot preview the product.
- It is a documentation-heavy approach, requiring documentation at every phase.
- Can give a false impression of progress – Poor progress visibility.
- Does not reflect problem-solving nature of software development – iterations of phases is not possible.
- Requirements for the system are determined and often "frozen" at the beginning of the development project thus the cost of changing the requirements at a later stage in the project can be very high.

1.3.2. Evolutionary Software Process Models

These process models interleave specification, development and validation activities. They are iterative and characterized in a manner that enables software engineers to develop increasingly more complete versions of the software. They take into consideration the evolutionary nature of software compared to the waterfall model. There are two types of evolutionary process models: Incremental Delivery and Spiral Model

A. Incremental Model/Delivery

This type of process model combines elements of linear sequential model applied repetitively with the iterative philosophy of prototyping. The software specification, design and implementation are broken down into a series of increments and each are developed in turn. Customers outline the services to be provides and identify the most and least important, then a number of delivery increments are then defined, each increment providing a subset of the system functionality.

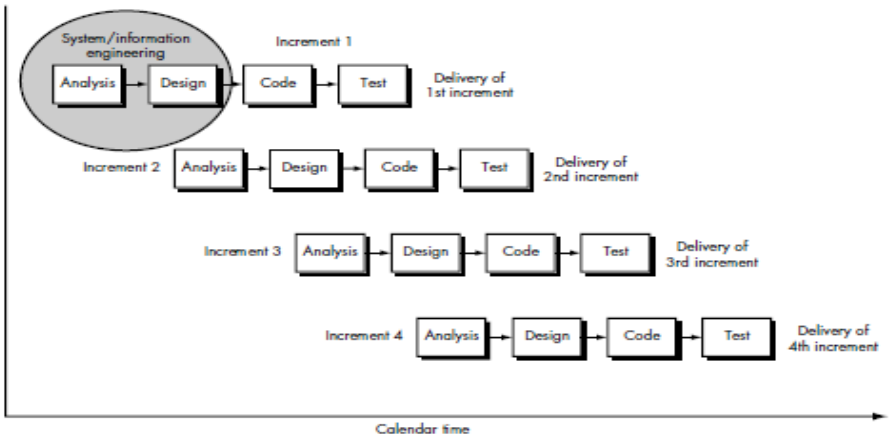

Figure 2: Incremental Delivery

The model is designed, implemented and tested as a series of incremental builds until the product is finished.

The first increment is often the "core product", and subsequent iterations are the supporting functionalities or the add-on features that a customer would like to see.

Iterative Delivery is Applicable:
When staffing is unavailable for a complete implementation

Advantages of the Increment Delivery Model

- Early delivery of part of the system so the customer doesn't wait until the entire system is delivered before they can gain value from it
- Customers can use early increments as prototypes and use the experience to obtain requirements for later increments.
- There is lower risk of project failure.

B. Spiral Model

Originally proposed by Barry Boehm in his article "*A Spiral Model of Software Development and Enhancement*" from 1985, this model represents the process as a spiral rather than a sequence of activities with back tracking. It couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. The development of the system progresses outwards in a spiral from an initial outline through to the final developed system. Each loop in the spiral represents a phase in the process.
There are no fixed phases e.g. specification, design in this model as loops are chosen depending on what is required. Risks are addressed and resolved throughout the process.

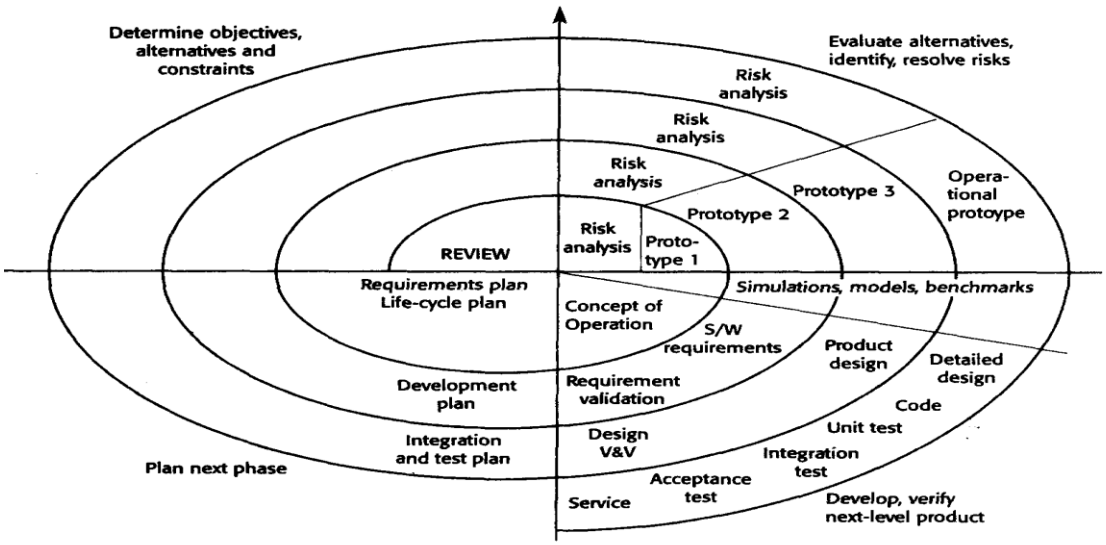

Figure 3: Spiral Model

As shown in figure 3 each loop is split into four sectors:

- **Objective Setting**: Specific objectives for the phase are defined, constraints on the process and product are identified and a management plan drawn up. Project risks are identified and alternative strategies planned.

- **Risk assessment and reduction:** A detailed analysis is carried out for each of the identified project risks and steps are taken to reduce the risk. In case the risks indicate any kind of uncertainty in requirements, prototyping may be used to proceed with the available data and find out possible solution in order to deal with the potential changes in the requirements E.g. for the risk of inappropriate requirements, a prototype system may be developed.

- **Development and Validation**: After risk evaluation, a development model is chosen. E.g. If user interface risks are dominant, an appropriate development model might be evolutionary prototyping, if the main identified risk is sub-system integration, the most appropriate model would be waterfall model. The output of this phase is passed through all the phases iteratively in order to obtain improvements in the same.

- **Planning**: The project is reviewed and a decision made to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up or the next phase of the project.

Advantages of the Spiral Model

- Delivers initial value early thus;
    - Mitigating the risk of failure
    - Allowing focus on high-priority functionality
- Frequent requirements refinement enables use of feedback from one iteration to refine requirements for the next.
- Mitigates impact of change.

Disadvantages of the Spiral Model

- Requires considerable risk-assessment expertise in order to work well.
- Might need special skills e.g. rapid prototyping

1.3.3. Rapid Application Development Model (RAD)

This is an incremental software development process model that emphasizes an extremely short development cycle, usually between 60 to 90 days. The software developers and end-users work together in real time to develop the product and if requirements are well understood and project scope is constrained, the RAD process enables the development team to create a "fully functional system" within the short time.

This model is primarily used for information systems and if a business application can be modularized so that each major function can be completed within the development cycle then it is a candidate for the RAD model. In this case, each team can be assigned a model, which is then integrated to form a whole.

Fundamentals of the RAD methodology thus include:

- Combining the best available techniques and specifying the sequence of tasks that will make those techniques most effective.
- Using evolutionary prototypes that are eventually transformed into the final product.
- Using workshops, instead of interviews, to gather requirements and review design.
- Selecting a set of CASE tools to support modelling, prototyping, and code re-usability, as well as automating many of the combinations of techniques.
- Implementing time-boxed development that allows development teams to quickly build the core of the system and implement refinements in subsequent releases.
- Providing guidelines for success and describing pitfalls to avoid.

The aforementioned practices also propose that products can be developed faster and of higher quality.

The RAD model encompasses the following phases:

a) Business modeling

The information flow among business functions is modeled in a way that the following questions are answered:

What information drives the business process?

What information is generated?

Who generates it?

Where does the information go?

Who processes it?

b) Data modeling

The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business. The characteristics (called attributes) of each object are identified and the relationships between these objects are defined.

c) Process modeling

The data objects defined in the data-modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

d) Application generation

RAD assumes the use of the RAD fourth generation techniques and tools like Visual Basic, Visual C++, Delphi etc rather than creating software using conventional third generation programming languages. The RAD works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

e) Testing and turnover

Since the RAD process emphasizes reuse, many of the program components have already been tested. This minimizes the testing and development time.

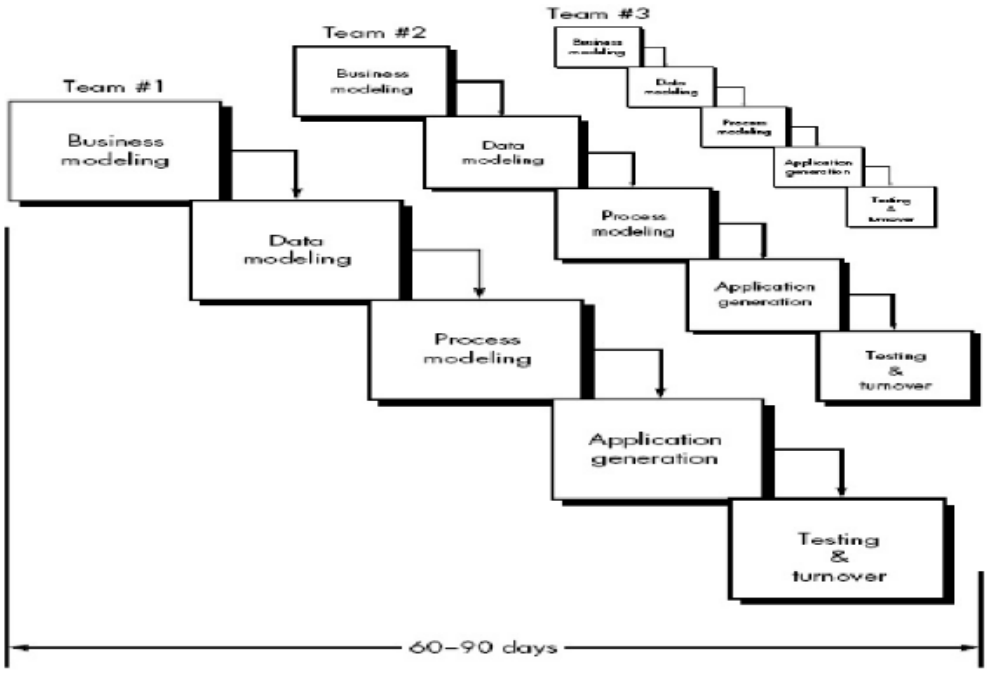

Figure 4: RAD Model

Applicable:

- Primarily for information systems applications.
- Where requirements are well understood.
- If a business application can be modularized so that each major function can be completed within the development cycle.

RAD Advantages

- A system can be developed in a short time.
- There is early visibility due to prototyping.
- The methodology provides greater flexibility because the developers can redesign almost at will.
- Reduced manual coding since automation tools, code generators and code reuse are involved. This also means less defects possible
- Increased user involvement because they are represented on the team at all times.

RAD Disadvantages

- Requires sufficient human resources to create the right number of RAD teams, for large but scalable projects.
- Requires committed developers and customers to the 'rapid-fire' activities to get a system complete in a short time frame otherwise RAD fails.
- If a system cannot be properly modularized, building components for RAD will be problematic.
- It is not appropriate when technical risks are high, e.g. when a new application makes heavy use of new technology.

1.3.4. V-Model

This model is presumed to be the extension of the waterfall model. The only difference is that instead of moving down in a linear way, the process steps are bent upwards after the coding phase (implementation), to form the typical V shape.

Presumed to be the extension/variant of the waterfall model, it emphasizes the verification and validation of the product - demonstrating the relationships between each phase of the development life cycle and its associated phase of testing. Thus testing of the product is planned in parallel with a corresponding phase of development, for example, the system test is carried out on the basis of the results specification phase, and test plans may be created after the requirements have been specified.

The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

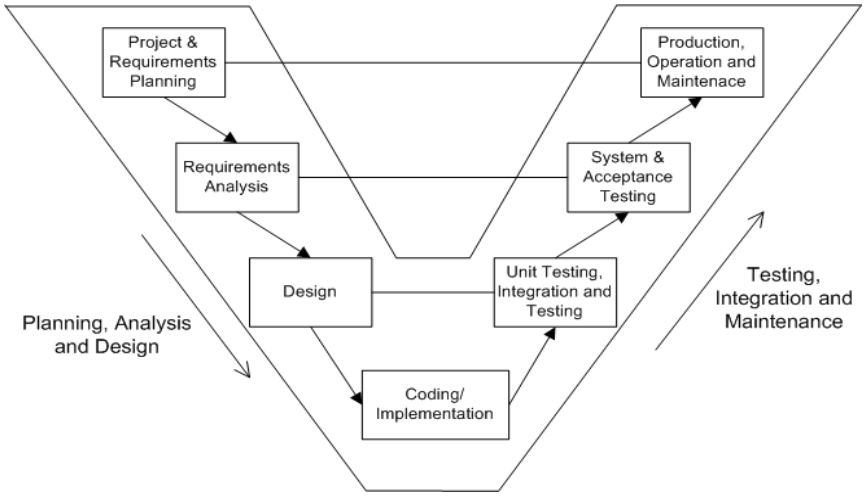

Figure 5: V-Model

Advantages

- It is simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

Disadvantages

- Very rigid, like the waterfall model.
- Adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- This model doesn't provide a clear path for problems found during testing phases.
- It does not easily handle concurrent events.
- It does not contain risk analysis activities.

1.3.5. Prototyping Model

A prototype is a working model that is functionally equivalent to a component of the product. This model is designed to assist the customer (or developer) who usually has only a general view of what is expected from the software product in understanding requirements. It allows a client to interact and experiment with a working representation of the product. It is **NOT** designed to deliver a production system. It works in such a way that a throwaway prototype is built from currently known user needs but the application/system so developed is without functionality. The development only proceeds when the client is satisfied with the functioning of the prototype and therefore at this stage the developer determines the specifications of the client's real needs.

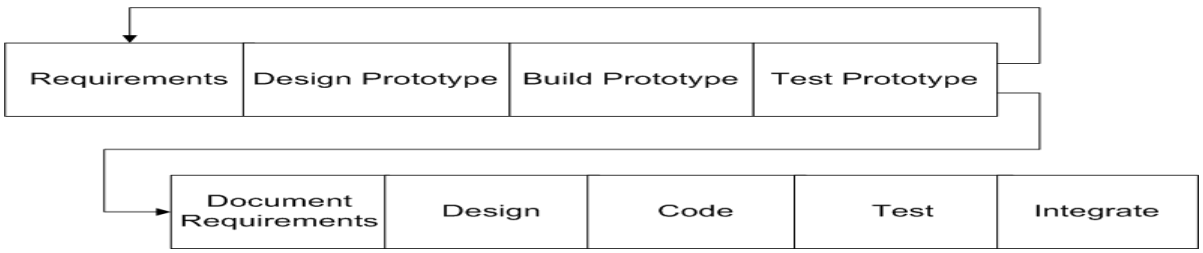

Figure 6: Prototyping Model

Prototyping Model is used for:

- Understanding the requirements for the user interface
- Examining feasibility of a proposed design approach
- Exploring system performance issues

Disadvantages:

- Clients usually treat the prototype as the final solution or expect only a few changes to the prototype and thus try to rush through, giving little consideration to the quality requirements for the solution.
- On the other hand there is the temptation for the development team to loose focus of the real purpose of the prototype and thus compromise the quality of the product e.g. using an inappropriate programming language.
- Prototype is only a partial specification and not the product itself.

1.3.6. Rational Unified Process (RUP)

The Rational Unified Process (RUP) is an example of a modern process model derived from the work on UML and associated processes. It specifically embeds object-oriented techniques and uses UML as its principle notation.

This process recognizes that the traditional waterfall approach can be inefficient because it idles key team members for extended periods of time. Many feel that the waterfall approach also introduces a lot of risk because it defers testing and integration until the end of the project lifecycle.

RUP is highly flexible in its developmental path, as any stage can be updated at any time. It uses an iterative, incremental approach for object -oriented development.

Different process stakeholders may see the RUP from different perspectives which also double as its definition. Below are three examples of typical perspectives into a RUP based process:

- A dynamic perspective.
- A static perspective.
- A practice perspective.

A. Dynamic perspective

This shows the phases of the model over time. It is four phased:

1. **Inception**
   This phase centres on assessing needs, requirements, viability and feasibility of the program or project as the developers define the scope of the project and its business case.
2. **Elaboration**
   The developers analyze the project's needs in greater detail and define its architectural foundation. They measure the architecture of the system's appropriateness based on the project needs.
3. **Construction**
   In this phase, the actual software system is made by developing components and features. It also includes the first release of the developed software.
4. **Transition**

This stage marks the end of the development cycle, if all objectives are met. It includes training of the end users, beta testing and the final implementation of the system.

Each phase is organised into a number of separate iterations which must satisfy certain criteria. At the end of each iteration, RUP provides a prototype.

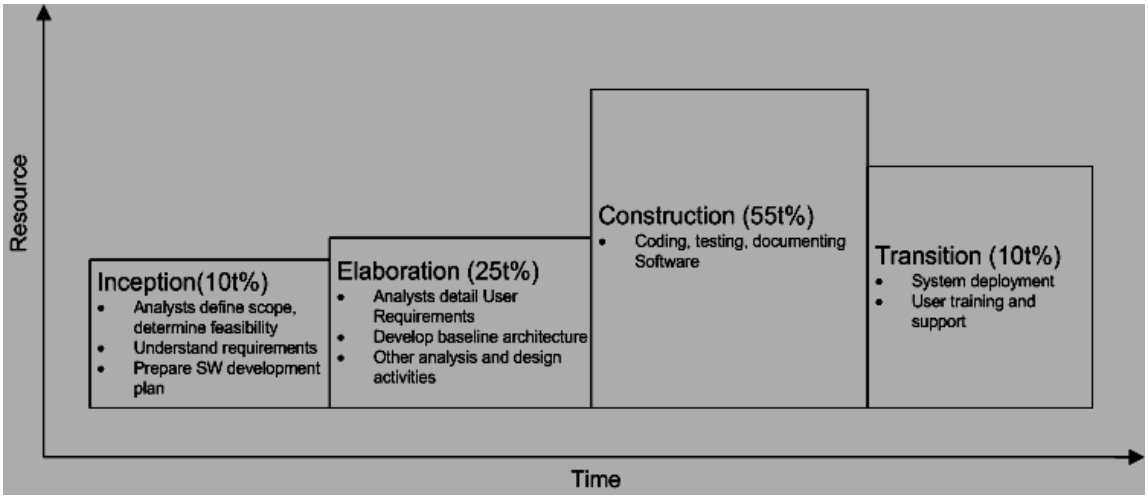

Figure 7: Rational Unified Process Phases

B. A static perspective

This perspective shows the process activities (workflows) that are enacted within the phases. A workflow is a set of activities.

Sample core workflows include:

- Business modeling – Requirements capture
- Requirements – Use case model
- Analysis – Analysis model(i.e conceptual object model)
- Design – Design model(e.g classes, use case realizations)
- Implementation – Implementation model (components)
- Test – Test model (test cases)

C. A static perspective

This suggests **good practices** to be used during the process:

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Benefits of the RUP iterative approach

- Risks are mitigated earlier
- Change is more manageable
- Higher level of reuse
- The project team can learn along the way
- Better overall quality

1.3.7. Agile Development

This model appeared in the 90's as a reaction to the strict Waterfall method. Software is developed progressively with each new release adding more capabilities and it eemphasizes flexibility during the software development process.

It reduces risk by breaking projects into small, time-limited modules ("iterations"), each of which is approached like a small, self-contained mini-project, lasting only a few weeks (analysis, design, production, testing and documentation). It is intended that even though iterations may not add enough functionality to warrant releasing the product, there will be a new software release at the end of every iteration. In addition, at the end of each iteration, the project's priorities, direction and progress are re-evaluated.

Agile methods emphasize real-time communication, preferably face-to-face, over written documents and thus produce very little documentation compared to other methods. Combined with this, they also emphasize working software as the primary measure of progress.

Variants of Agile development (methodologies) include; Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, Dynamic Systems Development Model (DSDM).

Applicable for projects involving:

- Unpredictable or dynamic requirements
- Responsible and motivated developers
- Customers who understand the process and will get involved

The goal of agility

The goal of agility is to develop software in the face of changing environment and constrained resources

- More a set of principles than a fixed model; many variations of agile processes
- Developers are indoctrinated with the principles and then trusted to act with less oversight

1.3.7.1. eXtreme Programming (XP)

This approach of software development was put together by Beck (2000) and one of the many popular Agile Process models. It registers success as it stresses customer satisfaction and delivers what you need when you need it and relies on constant code improvement.

The main goal of XP is to lower the cost of change in software requirements.
It is characterized by short cycles, incremental planning, focus on automated tests written by programmers and customers to monitor progress and reliance on an evolutionary approach to development lasting throughout the lifetime of the system.

Key Emphases:

- Two-person(pair) programming teams - Two programmers work together on the problem exchanging information and insight and sharing skills and the advantages of this pair programming setup include:
    1. There is more and better communication among the developers.
    2. Higher levels of productivity
    3. Higher quality code
    4. Reinforcement of code-and-test discipline with test suites designed first before any coding has to be done.

- Customer on-site during development (part of developer team)

All phases of the life cycle converge into a series of activities based on coding, testing, listening, and designing.
Coding and testing are closely related parts of the process and the programmers write code and develop tests for it too and this code is tested immediately after it is written; if the test is successful, development proceeds; otherwise rework code. It is emphasised that testing is only to be done not on everything, but on those things that can break or go wrong.

Strengths

- Empowers developers to confidently respond to the changing customer requirements, even late in the lifecycle.
- Emphasizes collaborative teamwork as the team self organizes around the problem at hand.

Weaknesses

- Detailed specifications are not created or preserved.
- Programmers are required to work in pairs - not all software developers expect to be asked to work this way.
- Most of the design activity takes place on the fly and incrementally, starting with "the simplest thing that could possibly work" and adding complexity only when it's required by failing tests. This could result in more re-design effort than only re-designing when requirements change.

The role of the ‘customer representative is attached to the project’ can become a single-point-of-failure for the project and some people have found it to be a source of stress.