Agile Software Development

Key Principles.

Customer Involvement: Customer Should be Closely involved throughout the Development Process Their role is provide and priorities new system requirements and to evaluate the iterations of the system.

Incremental Delivery: The Software Developed in Increments with the customer specifying the requirements to be included in each increment.

People Not Process: The Skills of Development Team should be recognized and exploited. Team member should be left to develop their own ways of working without prescriptive process.

Embrace Change: Expect the System Requirements to Change and so design system to accommodate these changes.

Maintain Simplicity: Focus on Simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate the complexity from the system.

Extreme programming Perhaps the best-known and most widely used agile method. Extreme Programming (XP) takes an 'extreme' approach to iterative development. New versions may be built several times per day; ncrements are delivered to customers every 2 weeks; All tests must be run for every build and the build is only accepted if tests run successfully.

XP and agile principles

Incremental development is supported through small, frequent system releases.

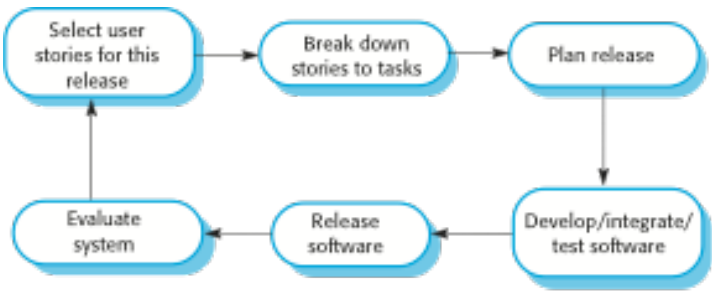
Customer involvement means full-time customer engagement with the team.

People not process through pair programming, collective ownership and a process that avoids long working hours.

Change supported through regular system releases.

Maintaining simplicity through constant refactoring of code.

The extreme programming release cycle



Extreme programming practices

Incremental planning: Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.

Small releases: The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

Simple design : Enough design is carried out to meet the current requirements and no more.

Test-first development: An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.

Refactoring: All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming: Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership: The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration: As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace: Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Testing in XP

Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
XP testing features:
§ Test-first development.
§ Incremental test development from scenarios.
§ User involvement in test development and validation.
§ Automated test harnesses are used to run all component tests each time that a new release is built.

Pair programming

In XP, programmers work in pairs, sitting together to develop code.
This helps develop common ownership of code and spreads knowledge across the team.
It serves as an informal review process as each line of code is looked at by more than 1 person.
It encourages refactoring as the whole team can benefit from this.
Measurements suggest that development productivity with pair programming is similar to that of two people working independently.
Pairs are created dynamically so that all team members work with each other during the development process.
The sharing of knowledge that happens during pair programming is very important as it reduces the overall risks to a project when team members leave.
Pair programming is not necessarily inefficient and there is evidence that a pair working together is more efficient than 2 programmers working separately.

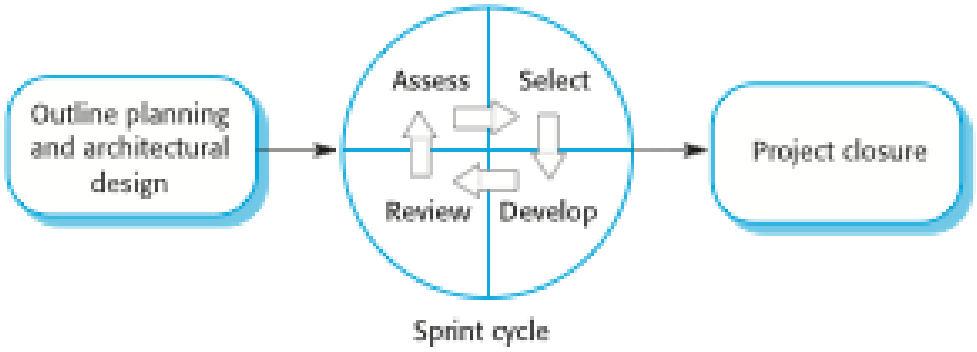
Advantages of pair programming

1. It supports the idea of collective ownership and responsibility for the system. Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems.
2. It acts as an informal review process because each line of code is looked at by at least two people.
3. It helps support refactoring, which is a process of software improvement.

Scrum

The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.
There are three phases in Scrum.
The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
This is followed by a series of sprint cycles, where each cycle develops an increment of the system.

The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.



Scrum Process

Scrum benefits

The product is broken down into a set of manageable and understandable chunks.
Unstable requirements do not hold up progress.
The whole team have visibility of everything and consequently team communication is improved.
Customers see on-time delivery of increments and gain feedback on how the product works.
Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Scaling out and scaling up

'Scaling up' is concerned with using agile methods for developing large software systems that cannot be developed by a small team.
'Scaling out' is concerned with how agile methods can be introduced across a large organization with many years of software development experience.

Requirements Engineering

Requirements engineering definition: The process of establishing the services that the ustomer requires from a system and the constraints under which it operates and is developed. The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process. A requirement may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification. This is inevitable as requirements may serve a dual function as described in the following scenarios:

1) May be the basis for a bid for a contract - therefore must be open to interpretation;
2) May be the basis for the contract itself - therefore must be defined in detail;
3) Both these statements may be called requirements.

Types of requirements:

Requirements can be basically categorised into:

1. User requirement which are statements in natural language plus diagrams of the services the system provides & its operational constraints. Basically specifies external system behavior. These are the requirements which are written for customers

2. Systems Requirements are a structured document setting out detailed descriptions of the system‟s functions, services & operational constraints. It is necessary that the systems requirements

should reflect accurately what the customer wants and should also precisely define what should be implemented. This could be a part of a contract between client and contractor

Example of a user requirement:

In a Library management system, the following functionalities are expected to be present

1. The system will maintain records of all library items including books, serials, newspapers, magazines, video and audio tapes, reports, collections of transparencies, computer discs and CD-ROMs.
2. Paper-based library items are stored on open shelves in the library and the system records their reference position in the library.
3. No item will be removed from the library without the details of its borrowing being recorded in the system.
4. All items will have a bar code containing a unique reference number by which an tem can be identified within the system.

For the same application the example of a System requirement could be:

1. The system will permit all users to search for an item by title, by author or by ISBN
2. Staff will be able to search for an item by bar code ref. number
3. Books can be borrowed for 15 days while CD-ROMs, Audio tapes & reports can be borrowed only for 3 days
4. Borrowed items that are one day overdue in their return will cause a reminder letter to be printed
5. Librarian should be able to find out details like Number of books & materials borrowed (on a given day, by a given client)
6. Selected items may be temporarily blocked by authorized staff

Classification of requirements – Another way

1. Functional requirements: Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. These requirements may also state what the system should not do.
2. Non-functional requirements: Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.These requirements often apply to the system as a whole rather than individual features or services.
3. Domain requirements: Constraints on the system from the domain of operation of the final system.

Each of the above requirements is explained in detail.

Functional Requirements:

Describe functionality or system services and depend on the type of software, expected users and the organization where the software is used. This could be high-level statements of what the system should do. However, it should describe all the system services in detail which could include its inputs, its Outputs, exceptions and so on. Such requirements are generally described in fairly abstract but precise way.

Non-Functional Requirements:

Define system properties and constraints e.g. reliability, response time and storage occupancy, Security, etc.. Alternatively they may define platform constraints like, I/O devices capability, data representations. Process requirements may also be specified mandating a particular CASE system, programming language or development method. These requirements arise through organizational policies, budget limits, interoperability needs etc. They may be considered to be more critical than functional requirements because if these are not met, the system will be useless.

Non-functional requirements can be further classified as shown in Fig

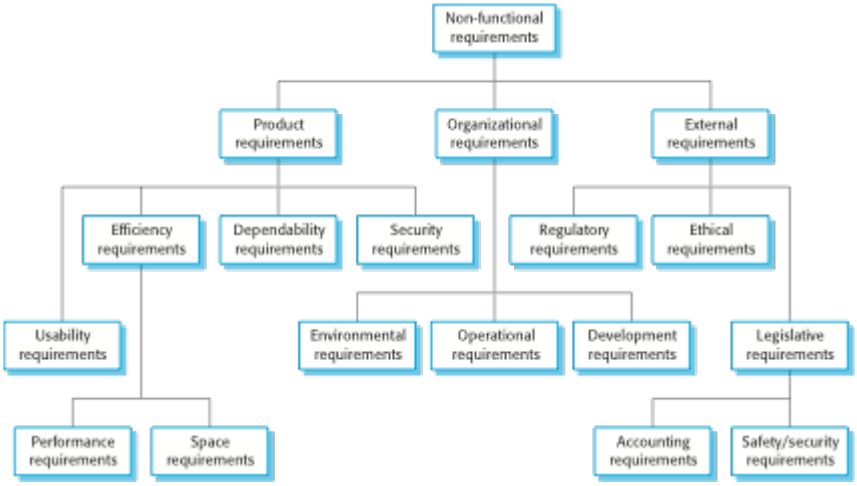


Fig : Classification of Non-functional requirements

The classification can be described as given below:

1. Product requirements: Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
2. Organisational requirements: Requirements which are a consequence of organisational policies and procedures e.g.process standards used, implementation requirements, etc.
3. External requirements: Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, egislative requirements, etc.

Metrics for specifying nonfunctional requirements

Since non-functional requirements are equally important for the operation of a software metrics have been defined to indicate non-functional requirements as a part of requirements specification document.

Speed Processed transactions/second User/event response time Screen refresh time
Size Mbytes Number of ROM chips
Ease of use Training time Number of help frames
Reliability Mean time to failure, Probability of unavailability, Rate of failure occurrence
Robustness Time to restart after failure, Percentage of events causing failure, Probability of data corruption on failure
Portability Percentage of target dependent statements Number of target systems

The software requirements document

The software requirements document is the official statement of what is required of the system developers. It should include both a definition of user requirements and a specification of the system requirements. It is not a design document. As far as possible, it should be a set of what the system should do rather than how it should do it.

Fig shows the users of a requirements document. The illustration is self-explanatory. It should be noted that requirements specification is not just for end users or for developers. It is for all those entities who are involved through out the software development process.

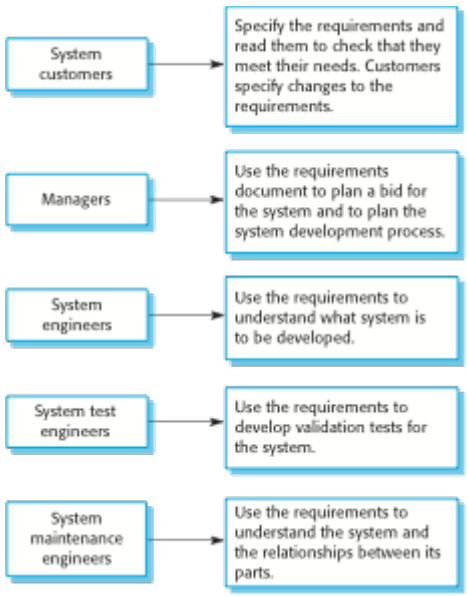


Fig 13: Users of a software requirements document

IEEE structure of a requirements document

The table below describes the various chapters and its description for a standard requirements document. This is given by the IEEE. All software do not necessarily follow this exact structure. But a standard structure tuned to the needs of the application is required to be followed.

The IEEE structure of a requirements document

Preface This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.

Introduction This should describe the need for the system. It should briefly describe the system‟s functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.

Glossary This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.

User requirements Here, you describe the services provided for the user. The nonfunctional definition system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.

System architecture This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

System requirements This should describe the functional and nonfunctional requirements in specification more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.

System models This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.

System evolution This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.

Appendices These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Ways of writing system requirements specification
There are different ways of writing requirements specification.
Natural language The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don‟t understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Guidelines for writing requirements
1. Invent a standard format and use it for all requirements.
2. Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements.
3.Use text highlighting to identify key parts of the requirement.
4. Avoid the use of computer jargon

Problems with natural language
1.Lack of clarity: Precision is difficult without making the document difficult to read.
2 Requirements confusion: Functional and non-functional requirements tend to be mixed-up.
3. Requirements amalgamation: Several different requirements may be expressed together.

Requirements Engineering (RE) processes
The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements. However, there are a number of generic activities common to all processes which are:
1. Feasibility Study
2. Requirements elicitation;
3. Requirements analysis;
4. Requirements validation;
5. Requirements management.

1. Feasibility Study : Decides whether or not the proposed system is worthwhile attempting. A short focused study that checks the following:
a) If the system contributes to organisational objectives;
b) If the system can be engineered using current technology and within budget;
c) If the system can be integrated with other systems that are used
d) If the system can fit into the cultural framework of the organizational culture and acceptable to all the stake-holders

2. Requirement Elicitation

Involves Interacting with technical staff working with customers to find out about the application domain, the services that the system should provide and the system‟s operational constraints. This may involve stakeholders like end-users, managers, Engineers involved in maintenance, domain experts, trade unions, etc. Domain requirements are also discovered at this stage.

Domain Requirements are derived from the application domain rather than specific needs of a customer in that domain. They usually refer to specialized domain terminology / concepts. They describe system characteristics & features that reflect the domain.

The requirements elicitation and analysis process

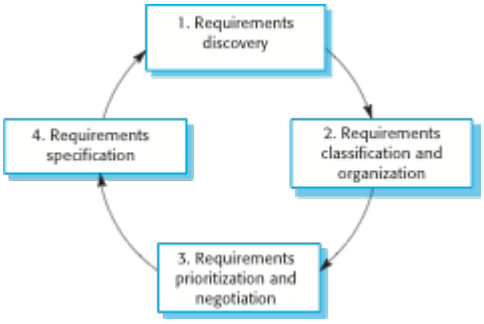


Fig : Requirements elicitation and analysis process

Requirements Discovery is done through

Interviewing: The RE team puts questions to stakeholders about the system that they use and the system to be developed. There are two types of interviews:

Closed interviews where a pre-defined set of questions are answered.

Open interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

Normally a mix of closed and open-ended interviewing is good for getting an

Scenarios

2 Scenarios are real-life examples of how a system can be used. They should include

§ A description of the starting situation;

§ A description of the normal flow of events;

§ A description of what can go wrong;

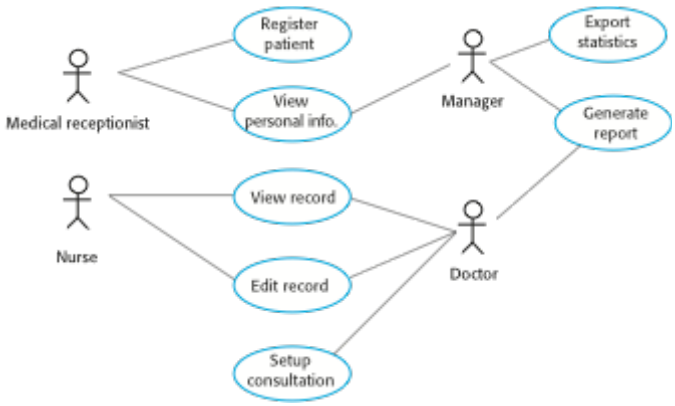
§ Information about other concurrent activities;

§ A description of the state when the scenario finishes.

Use cases

Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself. A set of use cases should describe all possible interactions with the system. High-level graphical model supplemented by more

detailed tabular description. Sequence diagrams may be used to add detail to use- cases by showing the sequence of event processing in the system.



Ethnography

A social scientist spends a considerable time observing and analysing how people actually work. People do not have to explain or articulate their work. Social and organisational factors of importance may be observed. Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

Problems with Requirement Elicitation

1. Stakeholders don‟t know what they really want.
2. Stakeholders express requirements in their own terms.
3. Different stakeholders may have conflicting requirements.
4. Organisational and political factors may influence the system requirements.
5. The requirements change during the analysis process
6. New stakeholders may emerge and the business environment change

Requirements checking

To check the requirements for a software, the following parameters need to be considered:

1. Validity. Does the system provide the functions which best support the customer‟s needs?
2. Consistency. Are there any requirements conflicts?
3. Completeness. Are all functions required by the customer included?
4. Realism. Can the requirements be implemented given available budget and technology
5. Verifiability. Can the requirements be checked?

Requirements validation techniques

Various ways exist for validating the requirements:

1. Requirements reviews: This involves systematic manual analysis of the requirements.
2. Prototyping: This involves using an executable model of the system to check requirements.
3. Test-case generation: Developing tests for requirements to check testability.

Requirements management

Process of understanding and controlling the changing requirements during the requirements engineering process and system development forms the major activity of requirements management. Requirements are inevitably incomplete & inconsistent. New requirements emerge during the process as business needs change and a better understanding of the system is developed. Different viewpoints have different requirements and these are often contradictory. All this needs reconciliation & management.

Reasons for change in Requirements

1. The business and technical environment of the system always changes after installation. New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.

2. The people who pay for a system and the users of that system are rarely the same people. System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery; new features may have to be added for user support if the system is to meet its goals.
3. Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory. The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements management planning

This activity establishes the level of requirements management detail that is required. It is essential that Requirements management decisions need to be taken for the following purposes:
1. Requirements identification- Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
2. A change management process -This is the set of activities that assess the impact and cost of changes.
3. Traceability policies- These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
4. Tool support- Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

Requirements change management

Deciding if a requirements change should be accepted

Problem analysis and change specification

• During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.

Change analysis and costing

• The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.

Change implementation

• The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.