Software Development Methodologies, Processes and Life-Cycle in a Project-Oriented Company

Katranček, Luka

University of Zagreb

Faculty of Economics and Business

Bachelor Degree in Business

Software Development Methodologies, Processes and Life-Cycle in a Project-Oriented Company

Undergraduate Thesis

Luka Katranček

Course: Informatics

Mentor: Mirjana Pejić-Bach, PhD

Luka Katranček, JMBAG: 0067532774

Zagreb, September 2019

Name and family name of student:_____

STATEMENT ON ACADEMIC INTEGRITY

I hereby declare and confirm with my signature that the _____ (type of the paper) is exclusively the result of my own autonomous work based on my research and literature published, which is seen in the notes and bibliography used. I also declare that no part of the paper submitted has been made in an inappropriate way, whether by plagiarizing or infringing on any third person's copyright. Finally, I declare that no part of the paper submitted has been used for any other paper in another higher education institution, research institution or educational institution.

Student:

In Zagreb, _____ _____

(date) (signature)

Abstract

With this paper, I would like to put on paper the whole lifecycle of Software Development. From the business case for a new software solution, business requirements creation, features creation to development, development methodologies, software testing methodologies and lastly software maintenance and software retiring. I will explain all this from the perspective of a project-oriented and heavily utilized company, which is working at full capacity requiring strict prioritization of projects and deliverables. I am using all available sources such as scientific articles and literature dealing with topics on software development methodologies, project and portfolio management and software release management available to me. Purpose for this paper is to paint a clear and concise picture on how project-oriented companies can tackle risks and issues managing software development in order to ensure the creation of added value to their products and customers.

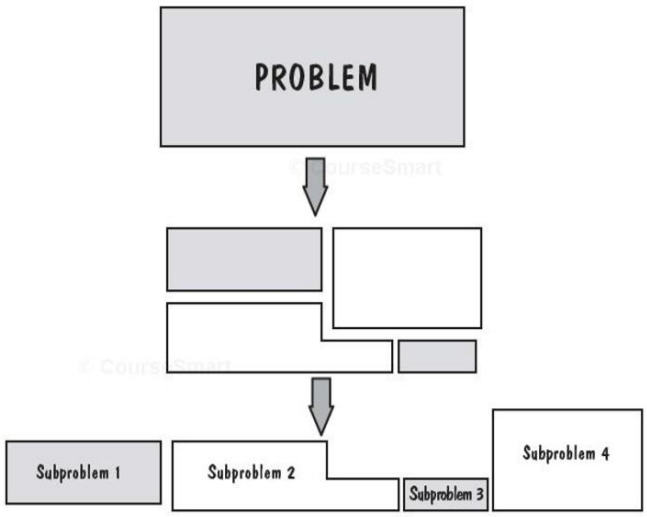
Table of Content

1 What is Software Development?

Software development is a tool used to help solve problems. More than often is the problem we are dealing with related to a computer or an existing computer system, but sometimes the difficulties underlying the problem have nothing to do with computers and for that reason one has to understand the nature of the problem. More specifically, one must be very careful not to impose computing machinery or techniques on every problem that comes his way. The problem must be identified and solved first and only then, if needed, can technology be used as a tool to implement the solution.

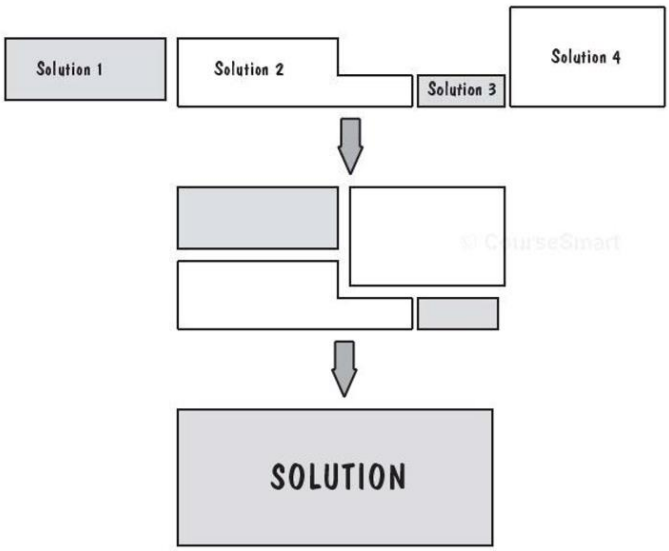
Most problems are large and sometimes tricky to handle, especially if they represent something new that has never been solved before. Firstly, the problem is analysed, meaning it is broken down into pieces that can be understood and then put into context. So the larger problem can be represented as a collection of small problems and their interrelationships. Figure 1 illustrates how analysis works. It is important to note that the relationships are as essential as the subproblems themselves. It could be that the relationships hold the key on how to solve the larger problem, rather than simply the nature of the subproblems.

Figure 1 Problem analysis



Once the problem was analysed, we must construct our solution from components that address the problem's various aspects. Figure 2 illustrates this reverse process. Synthesis is the putting together of a large structure from small building blocks. Same as analysis, the composition of individual solutions may be as challenging as the process of finding the solutions. Any problem-solving technique consists of two parts: analysing the problem to determine its nature, and then working out a solution based on the analysis.

Figure 2 Problem synthesis



To help us solve a problem, we employ a variety of methods, tools, procedures and paradigms. A method or technique is a structured procedure for producing a result. For example, a chemist may create a new substance mixing and altering other substances together in a careful and ordered fashion to that the new substance is created just the way it is supposed to. The procedure for creating a new substance involves timing and ingredients but may not depend on the equipment used.

A tool is an instrument or automated system for accomplishing something, for getting work done in a "better" way. "Better" meaning that the tool can make a certain task be performed more precisely, more efficiently, or more productively. For example a pair of scissors is a tool to make paper cutting faster and more straight, unlike if we were to tear a page.

A procedure is like a recipe: a combination of tools and techniques that make for a certain product.

Lastly, a paradigm can be explained with the help of examples of cooking styles. In the world we have Chinese cooking and French cooking, among others, and so too we differentiate between object-oriented development from procedural ones. One is not better than the other and each has its own pros and cons, but there are situation when one is better-suited than the other. A paradigm represents a particular approach or philosophy for building software.

One key component if software development is understanding that it is a creation of a product which has an end user who pays for that product/service. Communication between the customer and developer is essential; if that fails so does the system. We must understand what the

customer wants and needs before we can build a system to help solve the customer's problem. The number of people working on the software development depends on the project's size and degree of difficulty. However, no matter how many people are involved, the roles played throughout the life of a project can be distinguished. In general, the participation in a project falls into one of three categories: customer, user and developer. The customer is the company, organisation, or person who is paying for the software system to be developed. The developer is the company, organisation, or person who is building the software system for the customer, it encompasses any manager needed to coordinate and guide the programmers and testers. The user is the person who will actually use the system: the ones who sit at the PC and take advantage of the new tool.

1.1 Members of the development team

As said earlier, the first step in any development process is finding out what the customer wants and documenting the requirements. As already seen, analysis is the process of breaking things into components to understand them better. Consequently, the development team requires one or more business analysts to work with the customer and conclude what exactly the customer needs and transform the needs into business requirements.

Once the requirements are known and documented, analysts work with solution designers or software architects to generate a system-level description on what the system is to do. In turn, the solution designers work with programmers to describe the system in such a way that programmers can write code to implement what the requirements specify.

After the code is written, it must be tested. The first tests are usually done by the programmers themselves, but each team should have a testing team ready to test the code. A fresh set of eyes will catch the faults that the programmers overlook. When units of code are integrated into functioning groups, a team of testers works with the implementation team to verify that as the system is built up by combining pieces, it works as designed and according to specification.

When the development team is satisfied with the functionality and quality of the system, attention turns to the customer. The test team and the customer work together to verify that the complete system is what the customer wants; they do this by comparing how the system works with the initial set of requirements. Then, "trainers" show users how to use the system.

For many software systems, acceptance by the customer does not mean the end of the developer's job. Incidents are to be expected if the system is of high complexity, and the company should have a maintenance team ready to address these issues while in production.

Additionally, the customer's requirements may change as time passes, and corresponding changes to the system will have to be made.

Just as manufacturers look for ways to ensure the quality of the products they produce, so too must software developers find methods to ensure that their products/solutions are performing as imagined.

What is meant by quality[1]:

- The transcendental view – where quality is something we can recognize but not define; we can think of software quality as an ideal toward which to strive, but may never be implemented completely.
- The user view – where quality is fitness for purpose; measuring product characteristics such as reliability, to understand overall product quality.
- The manufacturing view – tries to measure quality during production and after delivery, it examines whether the product was built right the first time. It can also be called process view because it advocates applying efficient processes.
- The product view – looks at the product from inside to evaluate product's intrinsic values; what added value does it bring. It is assumed that good internal quality indicators will bring good external ones, such as reliability and maintainability.
- The value-based view – quality depends on the amount the customer is willing to pay for it

1.2 Constant change

Software development is a discipline which allows for the customer to review the plans at every step and to make changes in the design. After all, if the developer produces a great product that does not meet the customer's needs, that product is obsolete and has wasted everyone's time and effort. For that reason it is essential that developer's tools and techniques are used with an eye toward flexibility. As various stages of a project progress, constraints that were not anticipated arise. For example, after having chosen hardware and software to use for a project, a developer may find that a change in the customer requirements makes it difficult to use a particular database management system to produce menus exactly as promised to the customer.

It also must be recognized that most systems do not stand by themselves. They interact with other systems, either to receive or to provide information. Developing such systems is complex

[1] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J.

simply because they require a great deal of coordination with the systems with which they communicate. These problems are among many that affect the success of software development projects. Other reasons for an IT project to fail are[2]:

- Poorly defined applications (miscommunication between business and IT) contribute to a 66% project failure rate, costing U.S. businesses at least $30 billion every year (Forrester Research)
- 60% – 80% of project failures can be attributed di- rectly to poor requirements gathering, analysis, and management
- 50% are rolled back out of production
- 40% of problems are found by end users
- 25% – 40% of all spending on projects is wasted as a result of re-work
- Up to 80% of budgets are consumed fixing self- inflicted problems Defining "Process" and "Life-Cycle" in Software Development

2 Modelling the Process and Life-Cycle

2.1 The meaning of process

Whether it be developing software, writing a report, or taking a business trip, we always follow a sequence of steps to accomplish a set of tasks. The tasks are usually performed in the same order each time. We can think of a set of ordered tasks as a process: a series of steps involving activities, constraints and resources that produce an intended output. When the process involves the building of some product, we sometimes refer to the process as a life – cycle. Therefore, software life-cycle describes the life of a software product from the idea itself to its implementation, deliver, use, maintenance and shutdown.

Processes are important because they impose consistency and structure on a set of activities. These characteristics are useful when we know how to do something well and we want to ensure that others do it the same way. "A process is a collection of procedures, organised so that we build products to satisfy a set of goals or standards. In fact, the process may suggest that we choose from several procedures, as long as the goal we are addressing is met. For instance, the process may require that we check our design components before coding begins. The checking

[2] Kaur, R., Sengupta J., (2011) Software Process Models and Analysis on Failure of Software Development Projects, International Journal of Scientific & Engineering Research Volume 2, Issue 2, February-2011 ISSN 2229-5518

can be done using informal reviews or formal inspections, each an activity with its own procedure, but both addressing the same goal".[3]

2.2 Software Process Models (Methodologies)

Many process models are described in the software development literature. Some are prescriptions for the way software development should progress and others are descriptions of the way software development is done.

There are several reasons for modelling a process:

- When a group writes down a description of its development process, it forms a common understanding of the activities, resources and constraints involved in software development
- Creating a process model helps the development team find inconsistencies, redundancies and slip-ups. As these are noted and corrected, the process becomes more effective
- The model should reflect the goals of development, such as building high-quality software, finding faults early in development and meeting required budget and scheduled constraints. As the model is built, the development team may decide on certain measures to be implemented in the model, e.g. the team may include more frequent code reviews so that there are less errors.
- Every process should be tailored for the special situation in which it will be used. Building a process model helps the development team understand where that tailoring is to occur.

Every software development process model includes system requirements as input and a delivered product as output. Furthermore, I will explain two of the most common software development methodologies: Waterfall and Scrum.

2.2.1 Waterfall model

One of the first models to be introduced is Waterfall (Figure 3), where the stages are described as cascading from one to another. As shown in Figure 3, one development stage should be completed before the next begins. Waterfall can be described as a sequence of phases which form a software development life-cycle.

[3] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J. Page 46

Figure 3 Waterfall Model

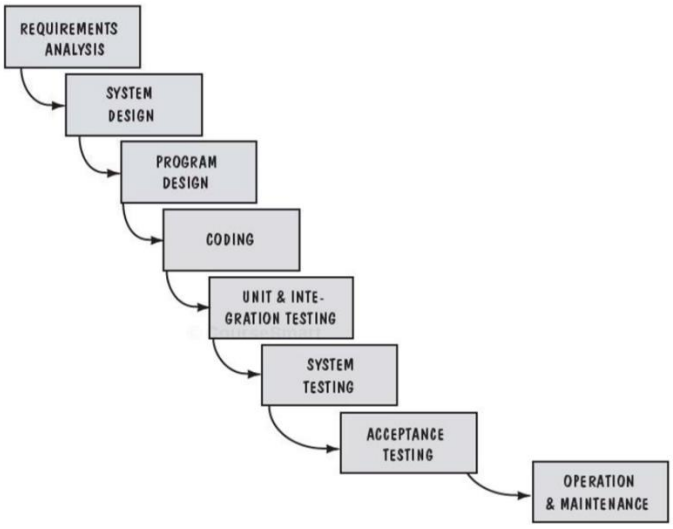


Figure 4 Waterfall Pros and Cons

Advantages[4] [5]	Disadvantages[6]
The staged development cycle enforces discipline	High effort and costs for writing and approving documents for each development phase.
Every phase has a defined start and end point, and progress can be conclusively identified	Extremely hard to respond to changes
The emphasis on requirements and design before writing a single line of code ensures minimal wastage of time and effort and reduces the risk of schedule slippage	When iterating a phase the iteration takes considerable effort for rework.
	When the system is put to use the customer discovers problems of early

[4] https://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/

[5] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J. Page 46

[6] Petersen, K., Wohlin, C., Baca, D., The Waterfall Model in Large-Scale Development, Blekinge Institute of Technology, https://www.researchgate.net/publication/30498645

	phases very late and system does not reflect current requirements.
	Problems of finished phases are left for later phases to solve
	Management of a large scope of requirements that have to be baselined to continue with development.
	Big-bang integration and test of the whole system in the end of the project can lead to unexpected quality problems, high costs, and schedule overrun.
	Lack of opportunity for customer to provide feedback on the system.
	The waterfall model increases lead-time due to that large chunks of software artifacts have to be approved at each gate.

2.2.2 Agile methods

Many of the software development processes proposed and used from the 1970s through the 1990s tried to impose some form of rigor on the way in which software is conceived, documented, developed and tested. In the late 1990s some developers who had resisted these too formal ways of working have constructed and modelled their own principles, trying to highlight the roles that flexibility could play in producing software quickly and capably. They codified their thinking in an "Agile manifesto" that focuses on 12 principles of an alternative way of thinking about software development[7]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

[7] https://agilemanifesto.org/iso/en/principles.html

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

The overall goal of agile development is to satisfy the customer by early and continuous delivery of valuable software. Many customers have business needs that change over time, reflecting not only newly discovered needs but also the need to respond to changes in the marketplace. For example, as software is being designed and constructed, a competitor may release a new product that requires a change in the software's planned functionality. In addition to that, a government agency or a standards body may issue a new regulation or standard that affects the software's design or requirements. General idea about agile is that keeping it flexible in the development process ensures constant availability to change and adapt. There are many examples of agile processes in current literature and each is layed out on a set of principles that implement the core of the agile manifesto. Examples:

- Extreme programming (XP) – A set of techniques for leveraging the creativity of developers and minimizing the amount of administrative overhead.
- Scrum – It uses iterative development, constructed of sprints (2 week – 4 week duration), to implement the product's backlog of prioritized requirements. Multiple self-organizing and autonomous teams implement product increments in parallel. Coordination is done at a brief daily status meeting called a "scrum".

- Lean[8] - 7 principles: Eliminate waste, Amplify learning, Decide as late as possible, Deliver as fast as possible, Empower the team, Build integrity, See the whole.
- Kanban[9] - Work items are visualised to give participants a view of progress and process, from start to finish – usually via a Kanban board. The aim is to provide a visual process management system which helps decision-making about what, when and how much to produce. Manages workflow.

3 Planning and Managing a Project

3.1 Tracking Progress

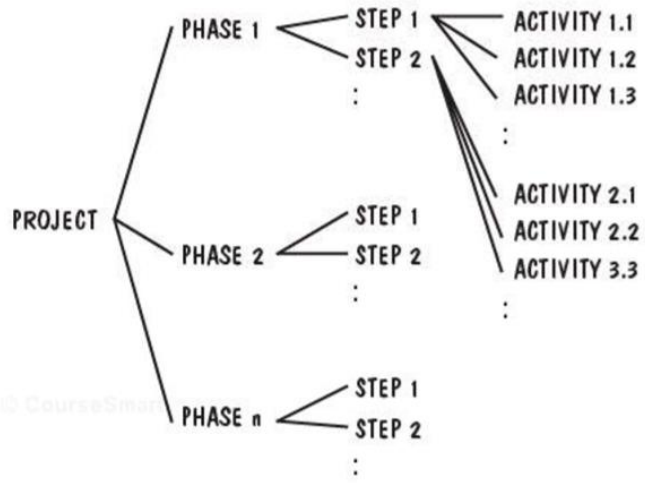
The software development cycle includes many steps, some of which are repeated until the system is complete and the customers and users are satisfied. However, before committing funds and resources for a software development or maintenance project, a customer usually wants an estimate of how much the project will cost and how long the project will take. A project schedule describes the software development cycle for a particular project by enumerating the phases or stages of the project and breaking each into discrete activities to be done. The schedule also portrays the interactions among these activities and estimates the time that each task or activity will take. That makes the schedule a timeline that shows when activities will begin and end, and when the related development products will be ready. To identify activities we must use analysis and synthesis to break down the problem into its component part, figure out a solution for each part, and then put the pieces together to form a coherent whole. We can use the same approach to determine project schedule. We begin by working with customers and potential users to understand what is it that they need and want. We list all project deliverables i.e. the items that the customer expects to see during project development. Next we determine what activities must take place in order to produce these deliverables. Certain events are designated to be milestones, indicating to us and our customers that a measurable level of progress has been made. For example, when the requirements are documented, inspected for consistency and completeness, and turned over to the design team, the requirements specification may be a project milestone. We must distinguish between milestones and activities. An activity ia a part of the project that takes place over a period of time, whereas a milestone is the completion of an activity – a particular point in time. To sum

[8] Poppendieck, M., Poppendieck, T., (2003): Lean Software Development: An Agile Toolkit. Addison-Wesley Professional

[9] Gross, J. M., McInnis, K. R., (2003): Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process. AMACOM

up, an activity has a beginning and an end, while a milestone is the end of a specifically designated activity.

Figure 5 Phases, steps and activities in a project



Analysis of this kind is also sometime described as generating a work breakdown structure for a given project, because it explains the project as a set of visible pieces of work. It is also worth noticing that the activities and milestones are items that both customer and developer can use to track development or maintenance. At any point in the process, the customer may want to follow the project's progress. One can then point to activities, indicating what work is under way, and to milestones, indicating what work has been completed. Modern project tools such as Microsoft Project can also visualise current activity status, indicate interdependence of the work units or of the parts of the project that can be developed concurrently.

Each activity can be described with four parameters: the precursor, duration, due date, endpoint. A precursor is an event or set of events that must occur before the activity can begin; it describes the set of conditions that allow the activity to begin. The duration is the length of time needed to complete the activity. The due date is the date by which the activity must be completed, often determined by contractual deadlines. Signifying that an activity has ended, the endpoint is usually a milestone or a deliverable.

There are many tools that can be used to keep track of a project's progress. Some are manual, others are simple spreadsheet applications, and still others are sophisticated tools with complex graphics. Many project management software systems draw a work breakdown structure and also assist the project manager in tracking progress by step and activity. A project management

package my draw a Gantt chart, a visualisation of the project where the activities are shown in parallel, with the degree of completion indicated by color or icon. (See Figure 5)

Figure 6 Gantt chart

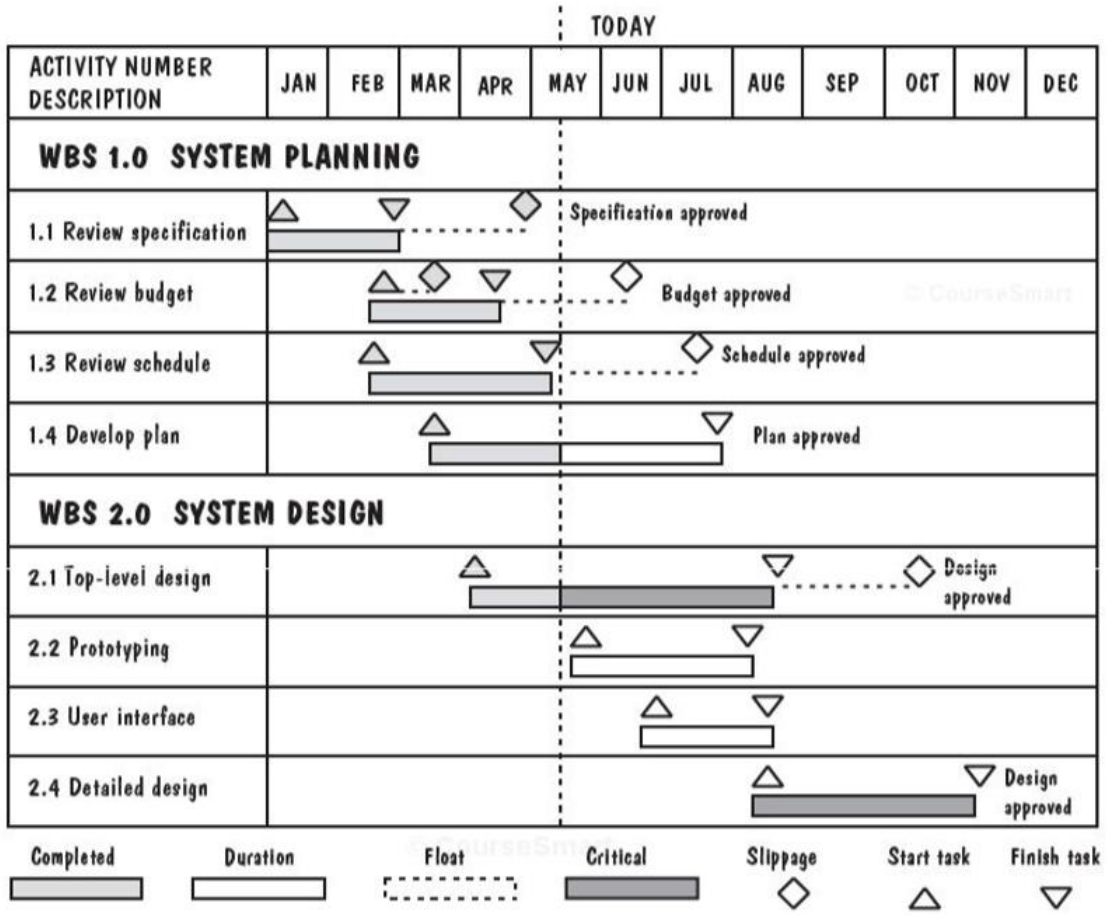


Figure 7 Resource histogram

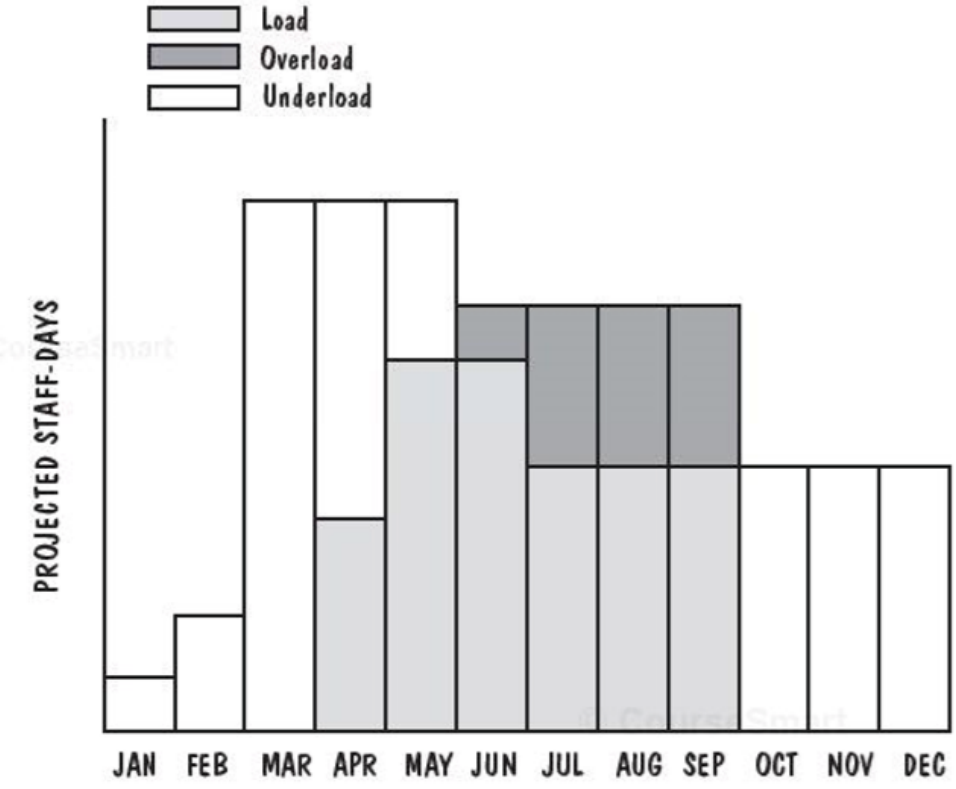
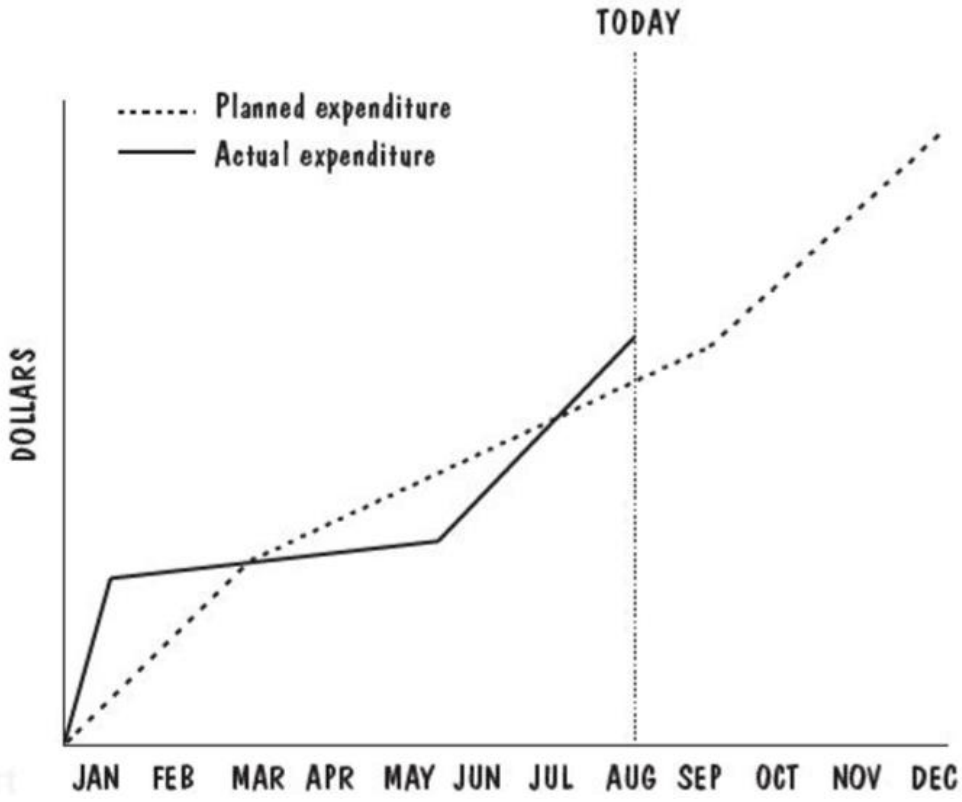


Figure 8 Tracking planned vs. Actual expenditure



3.2 Project personnel

To determine the project schedule and estimate the associated efforr and costs, we need to know approximately how many people will be working on the project, what tasks will they perform and what abilities and experience must they have in order to their jobs effectively and to the best of their ability.

No matter the model there are certain activities necessary to any software project. Key project activities are likely to include:

1. Requirements gathering / analysis
2. System design / Architecture
3. Development
4. Implementation / Integration
5. Testing
6. Training
7. Maintenance
8. Quality Assurance

Once we have decided on the roles of the project team members, we must decide which kinds of people we need in each role. Project team members may differ in many ways, and it is not enough to say that a project needs an analyst, two designers, and five programmers, for example. Two people with the same job title my differ in at least one of the following ways:

- Ability to perform the work
- Interest in work
- Experience with similar applications
- Experience with similar tools or languages
- Experience with similar techniques
- Experience with similar development environments
- Training
- Ability to communicate with others
- Ability to share responsibility with others
- Management skills

Each of these characteristics can affect an individual's ability to perform productively. These variations help to explain why one programmer can write a particular routine in a day while

another requires a week. The differences can be critical, not only to schedule estimation or cost estimation but also to the general success of the project.

3.3 Effort estimation

One of the crucial aspects of project planning and management is understanding how much is the project likely to cost. Cost overruns can cause customers to cancel projects and cost underestimates can force a project team to invest much of its time without financial compensation. A good cost estimate early in the project's life helps the project manager to know how many developers will be required and to arrange for the appropriate staff to be available when they are needed. The project budget pays for several types of costs: facilities, staff, methods and tools. For some projects the environment may already exist, so the costs are well understood and easy to estimate. But for other projects, the environment may have to be created.

For almost all software development projects the biggest component of cost is effort. We must determine how many mandays of effort will be required to complete the project. Effort is surely the cost component with the greatest degree of uncertainty.

Cost schedule and effort estimation must be done as early as possible during the project's life-cycle, since it affects resource allocation and project feasibility. Estimations should be done repeatedly throughout the life-cycle: as aspects of the project change, the estimate can refined.

3.4 Risk Management

As previously stated, many software project managers take steps to ensure that their projects are done on time and within effort and cost constraints. however, project management involves far more than tracking effort and schedule. Project manager must determine whether an unwelcome events may occur during development or maintenance and make plans to mitigate those events or, if they are inevitable, minimize their consequences. A risk is an unwanted event that has negative consequences. Project managers must engage in risk management to understand and control the risks on their projects. Boehm's Risk items.[10]

[10] Boehm, B.W. (1989): Software Risk Management, IEEE Computer Society Press

Figure 9 Boehm's Top 10 Risk items

1. Personnel shortfalls
2. Unrealistic schedules and budgets
3. Developing the wrong software functions
4. Developing the wrong user interface
5. Gold plating
6. Continuing stream of requirements changes
7. Shortfalls in externally performed tasks
8. Shortfalls in externally furnished tasks
9. Real-time performance shortfalls
10. Straining computer science capabilities

We distinguish risks from other projects by looking for three things[11]:

1. A loss associated with the event. The event must create a situation where something negative happens to the project: a loss of time, quality, money, control, …
2. The likelihood that the event will occur. We must have some idea of the probability that the event will occur. The likelihood of the risk, measured from 0 (impossible) to 1 (certain) is called risk probability. When the risk probability is 1, then the risk is called a problem, since it is certain to happen.
3. The degree to which we can change the outcome. For each risk, we must determine what we can do to minimise or avoid the impact of the event. Risk control involves a set of actions taken to reduce or eliminate risk.

We can quantify the effects of the risks we identify by multiplying the risk impact by the risk probability, so that we would get risk exposure. E.g. if the likelihood that the requirements will change after design is 0.3 and the cost ro redesign to new requirements is HRK50,000.00 then

[11] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J. Page 119

the risk exposure would be HRK15,000.00. Naturally, the risk probability can change over time, as can impact, so it is project manager's duty to track these parameters and plan for the change of events accordingly.

There are three strategies for risk reduction[12]:

1. Avoiding risk - by changing requirements for performance or functionality
2. Transferring risk – by allocating risks to other systems or by buying insurance to cover any financial loss should the risk become a reality
3. Assuming the risk – by accepting it and controlling it with the project's resources

To help make a decision on risk reduction, project manager must take into account the cost of reducing the risk. That is called risk leverage – the difference in risk exposure divided by the cost of reducing the risk. In other words, risk reduction leverage is

(Risk exposure before reduction – risk exposure after reduction) / (cost of risk reduction)

If the leverage value is not high enough to justify the action, then the project manager can look for other less costly or more effective reduction techniques.

It is useful to record decisions in a risk management plan, so that both the customer and the development team can review how problems are to be avoided, as well as how they are to be handled should they arise. Then, the project manager should monitor the project as development progresses, periodically re-evaluating the risks, their probability and their likely impact.

3.5 The Project Plan

To communicate risk analysis and management, project cost forecasts / estimates, schedules, activities and milestones, there should be a project plan. The plan puts in writing the customer's needs, as well as what is to be done to meet those needs.

A good project plan should include the following items:

1. Project scope
2. Project schedule
3. Project team organisation
4. Technical description of the proposed system

[12] Boehm, B.W. (1989): Software Risk Management, IEEE Computer Society Press

5. Project standards, procedures and proposed techniques and tools
6. Quality assurance plan
7. Test plan
8. Documentation plan
9. Training plan
10. Risk management plan
11. Maintenance plan

The scope defines the system boundaries. It explains what will be included in the system what will not be included. It provides clarification with the customer that we understand what is wanted, and vice versa. The schedule can be expressed using a work breakdown structure, the deliverables, and a timeline to show what will be happening at each point during the project life-cycle. A Gantt chart can be useful in illustrating the parallel nature of some of the development tasks.

The project plan also contains lists of people on the development team, how they are organised, and each of them will be tasked with. Not everyone will be needed during the whole duration of the project, so the plan usually holds a resource allocation chart to show staffing levels at different times.

For large projects, it may be appropriate to include a separate testing team to be included. Testing requires a great deal of planning to be effective. The test plan should state how test data will be generated, how each program module will be tested, how program modules will be integrated and then tested, how the entire system will be tested and who will perform each type of testing. Sometimes, systems are produced in stages or phases and the test plan should explain how each stage will be tested. When a new functionality is implemented to a system in stages, the test plan should address regression testing, to ensure that the existing functionalities still work correctly.

There are three milestones common to all software development processes that can serve as a basis for both technical process and project management[13]:

- Life-cycle objectives
- Life-cycle architecture
- Initial operational capability

[13] Boehm, B.W. (1989): Software Risk Management, IEEE Computer Society Press

Life-cycle objective milestone is to make sure the stakeholders agree with the system's goals. The key stakeholders act as a team to determine the system boundary, the environment in which the system will operate and the external systems with which the system must interact. Stakeholders then work through the scenarios of how the system will be used. The scenarios can be expressed in terms of prototypes, screen layouts, data flows, … If the system is business-critical, the scenarios should also include instances where the system fails, so that designers can determine how the system is supposed to react. The end result is an initial life-cycle plan[14]:

- Objectives: Why is the system being developed?
- Milestones and schedules: What will be done by when?
- Responsibilities: Who is responsible for a function?
- Approach: How will the job be done, technically or managerially?
- Resources: How much of each resource is needed?
- Feasibility: Can this be done, and is there a good business case for it?

Life-cycle architecture is coordinated with the life-cycle objectives. The purpose of the life-cycle architecture milestone is defining both the system and the software architectures. The architectural choices must address the project risks addressed by the risk management plan, focusing on system evolution in the long term as well as system requirement in the short term.

The key elements of the initial operational capability are the readiness of the software itself, the site at which the system will be used and the selection and training of the team that will use it.

4 Capturing the Requirements and System Design

4.1 Defining the requirement

A customer who asks a software development company to build a new application or system should have some notion or a business case on what the system should do. Often, the customer wants to automate a manual task, such as paying bill electronically instead of heading to the post office. Other times, a customer wants to enhance a current manual or already automated system. For example, the company is accepting VISA cards and now wants to accept MasterCard too.

A requirement is an expression of desired behaviour. For example, suppose a customer wants to build a system to create a weekly report of all newly onboarded merchants on their payment processing platform. One requirement may be to be able to filter the merchants by transaction

[14] Boehm, B.W. (1989): Software Risk Management, IEEE Computer Society Press

volume, second requirement may be to filter by the region the merchant operates from. Bear in mind that neither of these two requirements specify how the system is to be implemented. There is no mention of what database-management system to use, whether a client-server architecture will be employed, how much memory the computer is to have or what programming language must be used to develop the system. These implementation-specific descriptions are not considered to be requirements. The primary objective of the requirements gathering phase is to understand the customer's problems and wants. So, the requirements gathering phase is focused on the customer and the problem, not the solution or the implementation. One can say that requirements designate what behaviour the customer wants, without saying how that behaviour will be realised.

Stakeholders[15] to the requirements gathering phase:

- Clients – ones paying for the software to be developed.
- Customers – ones who buy the software after it is developed.
- Users – ones who are familiar with the current system and will use the future system
- Domain experts – ones who are familiar with the problem that the software must automate
- Market researchers – ones who have conducted surveys to determine future trends and potential customers' needs
- Software engineers or other technology experts – ones who ensure that the product is technically and economically feasible. They can educate the customer about innovative hardware and software technologies. They can advise on a functionality that takes advantage of these technologies. They can also estimate the cost and development time of the solution

Each stakeholder has a particular perspective on the system and how it should work. Often are these perspectives in a conflict. One of the many skills of a requirements analyst or a business analyst is the ability to understand each perspective and capture the requirements in a way that it reflects the concerns of each participant.

[15] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J. Page 119

4.2 Types of requirements

We differ between two types of requirements:

- A functional requirement – defines the boundaries of the solution space for the problem. The solution space is the set of possible ways that software can be designed to implement the requirements. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. Functional software requirements help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.[16]
- Non-functional requirement – defines a quality characteristic that the software solution must possess. defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. [17]

A design constraint is a design decision, such as choice of platform or interface components, that has already been made and now restricts the set of solutions to our problem. A process constraint is a restriction on the techniques or resources that can be used to build the system.

We can also differentiate between three categories of requirements:

1. Requirements that must be met – Essential
2. Requirements that are highly desirable but not necessary (Desirable)
3. Requirements that are possible but could be eliminated (Optional)

4.3 Design process

At this point in the development process the development team has a good understanding of the customer's problem and the requirements specifications which describe what an acceptable software solution will look like is known. Design is the creative process if figuring out how to implement all of the customer's requirements. Early design decisions address the system's

[16] https://www.guru99.com/functional-vs-non-functional-requirements.html
[17] https://www.guru99.com/functional-vs-non-functional-requirements.html

architecture explaining how compartmentalize the system into units, how the units relate to one another, and describing any externally visible properties of the units. [18]

6 ways to use architectural models:

1. To understand the system: what it will do and how it will do it
2. To determine how much of the system will reuse elements of previously build systems and how much of the system will be reusable in the future
3. To provide a blueprint for constructing the system, including where the load bearing parts of the system may be (design decisions that will be difficult to change later)
4. To reason about how the system might evolve, including performance, cost and prototyping concerns
5. To analyse dependencies and select the most appropriate design, implementation and testing techniques
6. To support management decisions and understand risks inherent in implementation and maintenance[19]

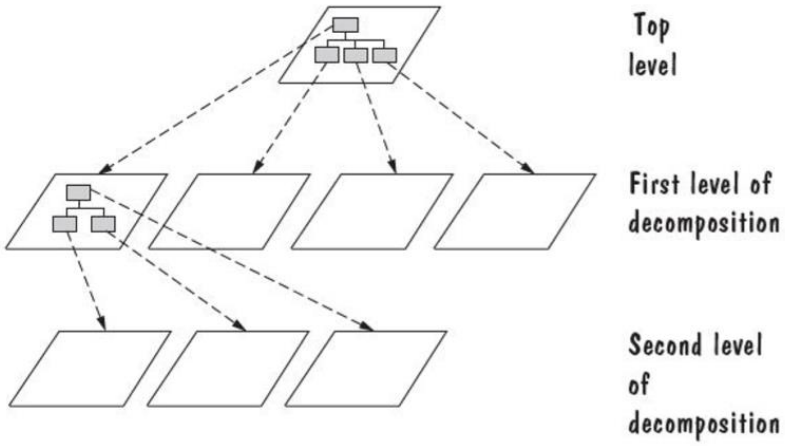
Popular architecture design methods:

- Functional decomposition: This method breaks down an operation to its foundational steps. System-level functions are decomposed to subfunctions which are then assigned to smaller modules. The design also describes which modules (subfunctions) call each other.
- Feature-oriented design: A type of functional decomposition that assignes features to modules. The high-level design describes the system in terms of a service and a collection of features. Lower-level designs provide detail as to how data are distributed among modules and how the distributed data realise the conceptual models.
- Process-oriented decomposition: This method partitions the system into concurrent processes. The high-level identifies the system's main tasks, which operate mostly independently of each other, assigns tasks to runtime processes and explains how the tasks coordinate with each other. Lower-level designs describe the process in more detail.

[18] Len, B., Clements, P., Kazman, R., (2013): Software Architecture in Practice, 3rd Edition. Addison-Wesley Professional
[19] Garlan, D., (2000): Software Architecture: a Roadmap

- Data-oriented decomposition: This method partitions the system into concurrent processes. The high-level design describes conceptual data structures, and lower-level designs provide detail as to how data are distributed among modules and how the distributed data realise the conceptual models.
- Event-oriented decomposition: This method focuses on the events that the system must handle and assigns responsibility for events to different modules. The high-level design catalogues the system's expected input events, and lower-level designs decompose the system into states and describe how events trigger transformations
- Object-oriented design: This method assigns objects to modules. The high-level design identifies the system's object types and explains how objects are related to one another. Lower-level designs detail the object's attributes and operations.[20]

Figure 10 Levels of decomposition



How we choose which design method to use depends on the system we are developing:

- Which aspects of the system's specification are most prominent (functions, object, features)?
- How is the system's interface described (input events, data streams)?

A good design is one that describes a system able to meet all of the requirements. However, other high-level concepts are important, too. For example, it is important that the design is

[20] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J. Page 232 - 233

adequate for the long-term intended use of the system and encompasses the following high-level notions of quality:

- Reusability: Are components of this design likely to be reused in later systems? If so, are they of sufficient quality to be reused?
- Understandability: is the design well structured and documented so that it will be easy for maintenance department to understand where in the system modifications need to be made?
- Modifiability: Will the system described in this design be easy enough to maintain after implementation is over? Or will changes likely have unintended consequences?

During the process of design it is important that other developers are an implicit factor, since the choice of design method depends on who will have to read and understand the design. Also, since designs are built from components, the interrelationships among components and data must be well-documented. Other developers should also participate in design reviews, evaluating the design at several stages and making suggestions for improvement. For all these reasons, it is essential that a design is documented clearly and completely, with discussions of the options faced and the decisions made.

5 Writing the Programs and Testing

5.1 Code

The task of writing code that implements the design can be intimidating for a couple of reasons. First, the designers may not have addressed all of the edge cases of the platform and programming environment; structures and relationships that are easy to describe with charts and tables are not always straightforward to be written as code. Second, code must be written in a way that is understandable not only to the author but also to all others that will be testing the code, taking over that part of the backlog or as the system evolves over time by someone else. Third, programmers must take advantage of the characteristic of the design's organisation, the data's structure, and the programming language's constructs while still creating code that is easily reusable.

When writing code, the following items should be considered:

- Organisational standards and guidelines
- Reuse of code from other projects

- Writing code to make it reusable on future projects using the low-level design as an initial framework, and moving in several iterations from design to code
- Incorporating a system-wide error-handling strategy
- Using documentation within programs and in external documents to explain the code's organization, data, control and function, as well as design decisions
- Preserving the quality design attributes in the code
- Using design aspects to suggest an implementation language[21]

Many corporate or organisational standards and procedures focus on the description accompanying a collection of programs. Program documentation is the set of written description that explain to a reader what the programs do and how they do it. Internal documentation is descriptive and is written directly within the code.

Standards and procedures can help a developer organise his/her thoughts and avoid mistakes. Some of the procedures involve methods of documenting your code so that it is clear and easy to follow. Such documentation allows you to leave and return to your work without losing track of what you had been doing. Standardised documentation also helps in locating faults and making changes, because it clarifies which sections of the program perform which functions.

5.1.1 Extreme programming[22]

One of the most popular programming methods in Agile is extreme programming. It is made up of a set of simple, yet interdependent practices. These practices work together to form a whole that is greater than its parts.

Customer team member

The customer and developer are to work closely together with each other so that they are both aware of each others problems and are working together to solve those problems.

The customer of an XP team is the person who defines and prioritizes features. The best case is for the customer to work in the same room as the developers. Next best is if the customer works 100 feet of the developers.

[21] Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J.
[22] Robert, C. M. (2002): Agile Software Development: principles, patterns, and pracites. Pearson.

User stories

When using XP, developers get the sense of the details of the requirements by talking them over with the customer, but the details need not to be captured. Rather, the customer writes down a few words on an index card that all agree will remind them of the conversation. The developers write down an estimate on the card at roughly the same time that the customer writes it. A user story is a mnemonic token of an ongoing conversation about a requirement. It is a planning tool that the customer uses to schedule the implementation of a requirement based upon its priority and estimated cost.

Short cycles

An XP project delivers working software every two weeks. Each of these two-week iterations produces working software that addresses some of the needs of the stakeholders in order to get their feedback. An iteration is a representation of minor delivery that may or may not be put in production. It is a collection of user stories selected by the customer according to a budget established by the developers.

The developers set a budget for an iteration by measuring how much they got done in the previous iteration. The customer may select any number of stories for the iteration, so long as the total of their estimates does not exceed that budget.

XP creates a release plan that maps out the next six iterations or so. A release is usually three months worth of work. It represents a major delivery that can be put in production. A release plan consists of prioritized collections of user stories that have been selected by the customer according to a budget given by the developers.

Acceptance tests

The details about the user stories are captured in the form of acceptance test specified by the customer. The acceptance test for a story are written immediately preceding, or even concurrent with, the implementation of that story. Acceptance tests are written in a scripting language that allows them to be run automatically and repeatedly. Together, they act to verify that the system is behaving as the customer have specified. Once an acceptance test passes, it is added to the body of passing acceptance test and is never allowed to fail again. This growing body of acceptance tests is run multiple times per day, every time the system is build. If an acceptance test fails, the build is declared a failure.

Pair programming

All production code is written by pairs of programmers working together at the same workstation. One member of each pair drives the keyboard and types the code. The other member of the pair watches the code being typed, looking for errors and improvements. The two interact intensely. Both are completely engaged in the act of writing software. The roles change frequently. The driver may get tired or stuck, and his pair partner will grab the keyboard and start to drive. The keyboard will move back and forth between them several times in an hour. The resultant code is designed and authored by both members. Pair membership changes at least once per day so that every programmer works in two different pairs each day. This helps increase the spread of knowledge between the team.

Test-driven development

All production code is written in order to make failing unit tests pass. First developers write a unit test that fails because the functionality for which it is testing does not exist. Then the developers write the code that makes that test pass.

5.2 Testing

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves execution of a software component to evaluate one or more properties of interest. Software testing also helps identify errors, gaps or missing requirements in contrary to the actual requirements. It can be done manually or using automated tools. [23]

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss.

There are multiple reasons for a program failure:

- The specification may be wrong or has a missing requirement
- The specification may contain a requirement that is impossible to implement, given the prescribed hardware and software
- The system design may contain a fault
- The program code may be wrong

[23] https://www.guru99.com/software-testing-introduction-importance.html

Testing is typically classified into three categories as shown in the table below.

Figure 11 Types of Testing

Testing Category	Types of Testing
Functional Testing	Unit Testing
	Integration testing
	Smoke testing
	User Acceptance Testing (UAT)
	Localisation
Non-Functional Testing	Performance
	Endurance
	Load
	Volume
	Scalability
Maintenance	Regression
	Maintenance

Unit Testing – a level of software testing where individual units / components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. Unit testing increases confidence in changing/managing code. Catching a bug or a defect during Unit testing is much more cost-efficient than realising there is a bug to be fixed during integration testing.

Integration testing – a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

System testing – a level of software testing where a complete and integrated software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements. Underlying analogy can be explained on an example of a car. All the parts such as the wheel, steering wheel, lights, … all is produced separately and unit tested separately. When two or more units are ready, they are assembled and Integration Testing is performed. When the whole car is integrated, System Testing is performed.

User Acceptance Testing (UAT) – level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's adherence to business requirements and assess whether it is acceptable for delivery. Acceptance testing is performed to act as a final confirmation that the car is ready to be made available to the end-users.

6 Delivering and Maintaining the System

6.1 Delivery

Many software developers assume that system delivery is a formality. However, even with plug-and-play systems (where developers hand over the system to the customer and are not responsible for its maintenance), delivery involves more than putting the system in place. It is the time during development when the development team helps users to understand and feel comfortable with the product. If delivery is not successful, users will not use the system properly and may be unhappy with its performance. In either case, users are not as productive or efficient as they could be, and the care taken to build a high-quality system will be wasted.

As the system is designed, aids that help users learn how to use the system are planned and developed. Accompanying the system is documentation to which user refer for problem-solving or further information.

6.2 Maintenance

Delivery of a system to the customer does not have to mark the end of the software developers' involvement with the system. In fact, many systems require continuous change, extending past delivery. In general, the more closely a system is correlated to the real world, the more likely it will be to require changes (and the more difficult those changes will be to make). Software maintenance deals with managing change in this part of the life-cycle.

Maintaining a system requires its own set of skills, in addition to those required for software development. Team in charge of maintaining a system must continuously interact with colleagues, customers, and users in order to effectively define problems and find their causes. Maintenance team must also need to understand the "big picture" of how software systems, with many complex interactions among their components, interact with the environment. Impact analysis, which builds and tracks links among requirements, design, code and test cases, is necessary to evaluate the effects of a change in one component on the rest of the systems.

Software maintenance is a part of the Software Development Life Cycle. Its main purpose is to modify and update software application after delivery to correct faults and to improve the performance of the system. It is a very broad activity that takes place soon after the development completed. It optimizes the system's performance by reducing errors, eliminating useless development and applying advanced development. Software development gets completed

within 5 years (depends on the complexity) while software maintenance is an ongoing activity and can be extended up to 15-20 years.[24]

Software Maintenance falls into the following categories[25]:

1. Adaptive – some modifications are being done in the system to keep it compatible with the changing environments
2. Perfective – checks for fine tuning of all elements of the system, functionalities and abilities to improve system performances
3. Corrective – detects bugs and errors in the existing solution to fix them and make the system work efficiently
4. Preventive – preventive software maintenance helps in preventing the system from any upcoming vulnerabilities

Software maintenance is required for several reasons that are listed below:

1. Bug fixing – error searching in the code and fixing it
2. Capability enhancement for changing environment – improvement of current features and functions to make the system more compatible for changing the environment
3. Removal of outdated functions – functionalities that are not in use anymore are being removed
4. Performance improvement – it is done to cope up with new requirements

7 Conclusion

Software development is hardly more than a few decades old and has only gained the attention of non-tech people in the last 20 years. Even though I have in this paper stated a number of ways how to approach creating a project plan, gather requirements, design and create a software system these are not standards, but ways of working. Software development industry is not as mature as the construction industry for example. While there are some differences to the construction industry there are similarities in a sense that both industries are creating something to according to the customer's needs and wants. Traditionally, companies have built up their own software development groups, hiring architects and developers to build software for them. But in the construction world, what company hires a full-time architect as an employee, hires

[24] https://scideas.in/2018/12/17/why-software-maintenance-is-important/
[25] https://www.coderhood.com/software-maintenance-understanding-the-4-types/

their own construction crew or purchases equipment needed to do the job? Unless they are a construction company or their industry revolves around construction, the answer is none.

There is no need for companies that do not deal with software to have their own internal software groups. A company can be good at what it originally is doing, it is hard for a company to be equally great at software development and at what they sell. Software is slowly being outsourced to companies who actually only offer those services. Some advantages to hire an outside software company:

- IT is not the company's core business, they are not the best at that
- Technology companies bring knowledge from the outside which helps create new ideas
- It creates a more competitive environment to help drive down costs and better the quality of service
- External companies are more easily held accountable
- Easier to ramp up and down, as needed
- The world of software is getting more complex and it takes a full time effort to stay in the loop of the latest technologies

As the world turns more toward outsourcing software services such as software development projects, companies will get better software at a greater value to the company and at a much lower risk.

8 Sources

Pfleeger S.L., Atlee, J.M. (2009): Software Engineering: Theory and Practice. 4th Edition. Pearson, Prentice Hall, N.J.

Royce, W.: Managing the development of large software systems: Concepts and techniques. In: Proc. IEEE WESCOM, IEEE Computer Society Press (1970)

Sommerville, I.: Software Engineering (7th Edition). Pearson Eductation Ltd. (2004)

Fundamental Approaches to Software Engineering: 13th International Conference, FASE 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings

Kaur, R., Sengupta J., (2011) Software Process Models and Analysis on Failure of Software Development Projects, International Journal of Scientific & Engineering Research Volume 2, Issue 2, February-2011 ISSN 2229-5518

Understanding the Pros and Cons of the Waterfall Model of Software Development https://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/

Petersen, K., Wohlin, C., Baca, D., The Waterfall Model in Large-Scale Development, Blekinge Institute of Technology, https://www.researchgate.net/publication/30498645

Agile Manifesto https://agilemanifesto.org/iso/en/principles.html

Poppendieck, M., Poppendieck, T., (2003): Lean Software Development: An Agile Toolkit. Addison-Wesley Professional

Gross, J. M., McInnis, K. R., (2003): Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process. AMACOM

Boehm, B.W. (1989): Software Risk Management, IEEE Computer Society Press

Functional Requirements vs Non Functional Requirements: Key Differences https://www.guru99.com/functional-vs-non-functional-requirements.html

Len, B., Clements, P., Kazman, R., (2013): Software Architecture in Practice, 3rd Edition. Addison-Wesley Professional

Garlan, D., (2000): Software Architecture: a Roadmap

Robert, C. M. (2002): Agile Software Development: principles, patterns, and pracites. Pearson.

What is Software Testing? Introduction, Definition, Basics & Types https://www.guru99.com/software-testing-introduction-importance.html

Why Software Maintenance is Important? https://scideas.in/2018/12/17/why-software-maintenance-is-important/

Software Maintenance, Understanding the 4 Types https://www.coderhood.com/software-maintenance-understanding-the-4-types/

9 List of Figures