The Notebook of a System Architect

Understanding the Software Development Life Cycle

Juuso Landgren

Master's thesis
March 2024
Master's Degree Programme in Information Technology
Full Stack Software Development

Description

Landgren, Juuso

The Notebook of a System Architect – Understanding the Software Development Life Cycle

Jyväskylä: Jamk University of Applied Sciences, March 2024, 81 pages.

Master’s Degree Programme in Information Technology. Master’s thesis.

Permission for open access publication: Yes

Language of publication: English

Abstract

Software plays a huge role in people's daily lives. Countless things usually involve some software or information system designed and implemented by someone.

The aim was to take a practical look at application development from a system architect's holistic perspective and experience. The objective was to help IT architects and developers understand application development as an extensive, structured process. The result was a notebook that visualizes the development process from idea to working application.

The work followed a diary-like approach, focusing weekly on application development topics for about six months. The selected cases were based on the Software Development Life Cycle (SDLC) model and decades of experience in software development. The software production process was described according to the SDLC steps, and the meaning and content of each step were examined from the perspective of modern development techniques. The work did not go into details such as programming languages, tools, data warehouses, and processes, which each company chooses according to its objectives and strategies.

The notebook highlighted that application development is applied, multifaceted, and solution-oriented brain work in which IT architects and application developers play an essential role. These professionals' key characteristics are problem-solving, technical skills, collaboration, and communication skills. Managing the whole requires a broad knowledge of multiple technologies, tools, and business processes.

The content of the notebook is used to systematically develop the software production process of the commissioner's company. The work will be used to define working software development methods, tools, and guidelines for the future.

Keywords/tags (subjects)

software architecture, software development, software development life cycle

Miscellaneous (Confidential information)

jamk

Description

Landgren, Juuso

Järjestelmäarkkitehdin muistikirja – Ohjelmiston kehityksen elinkaarimalli

Jyväskylä: Jyväskylän ammattikorkeakoulu, maaliskuu 2024, 81 sivua.

Master's Degree Programme in Information Technology. Opinnäytetyö, ylempi AMK.

Julkaisulupa avoimessa verkossa: Kyllä

Julkaisun kieli: Englanti

Tiivistelmä

Ohjelmistoilla on valtava merkitys ihmisten päivittäisessä elämässä. Lukemattomiin asioihin liittyy yleensä jokin ohjelmisto tai tietojärjestelmä, jonka joku on suunnitellut ja toteuttanut.

Tavoitteena oli syventyä käytännönläheisesti sovelluskehitykseen järjestelmäarkkitehdin kokonaisvaltaisen näkemyksen ja kokemuksen perusteella. Tarkoituksena oli auttaa it-arkkitehtejä ja sovelluskehittäjiä ymmärtämään sovellusten kehitysprosessia isona, jäsenneltynä kokonaisuutena. Lopputuloksena syntyi muistikirja, joka auttaa hahmottamaan kehitysprosessia aina ideasta toimivaan sovellukseen asti.

Työssä sovellettiin päiväkirjamaista opinnäytetyömallia, jossa viikoittain, noin puolen vuoden ajan, keskityttiin eri sovelluskehitysteemoihin. Valitut teemat perustuivat sovelluskehityksen elinkaarimalliin (SDLC) ja vuosikymmenten kokemukseen ohjelmistojen kehittämisestä. Ohjelmiston tuotantoprosessi kuvattiin SDLC:n vaiheiden mukaisesti, ja jokaisen vaiheen merkitystä ja sisältöä tarkasteltiin modernien kehitystekniikoiden näkökulmasta. Työssä ei otettu kantaa yksityiskohtiin, kuten ohjelmointikieliin, työkaluihin, tietovarastoihin ja prosesseihin, jotka jokainen yritys valitsee omien tavoitteidensa ja strategioidensa mukaisesti.

Muistikirja toi esille, että sovelluskehitys on soveltavaa, monipuolista ja ratkaisukeskeistä aivotyötä, jossa sekä IT-arkkitehdilla että sovelluskehittäjällä on merkittävä rooli. Näiden ammattilaisten keskeisiä ominaisuuksia ovat ongelmanratkaisukyky, tekniset taidot sekä yhteistyö- ja kommunikointitaidot. Kokonaisuuden hallinta vaatii laaja-alaista useiden teknologioiden, työkalujen ja toimintaprosessien osaamista.

Muistikirjan sisältöä hyödynnetään yrityksen ohjelmistotuotantoprosessin systemaattiseen kehittämiseen. Työn pohjalta määritellään sovelluskehityksen toimintatapoja, työvälineitä sekä tulevaisuuden suuntaviivoja.

Avainsanat (asiasanat)

järjestelmäarkkitehtuuri, ohjelmistokehitys, ohjelmiston kehityksen elinkaarimalli

Muut tiedot

Contents

1 Introduction

The world we live in has become digital. Our products and services rely on computers, software, information, data, ones, and zeroes. Our daily lives are filled with digital activities, whether at home, in our cars, during sports, at work, while engaging in leisure activities, or communicating with others – everywhere. In the workplace, most of us use computers and information systems daily. For a modern organization, they are indispensable. The Internet and the World Wide Web have helped information sharing, promoted real-time communications, enabled online shopping, expanded educational opportunities, fostered remote work, and created countless applications ranging from games to enterprise software.

Software development is essential for these things to work. Building software needs a diverse team of people: visionaries who generate innovative ideas, experts capable of refining those ideas into practical application designs, and skilled software engineers who can turn these plans into functional products and services. Of course, numerous other professionals are also involved in achieving a successful product – without forgetting the actual users.

In the core of software development, there are two pivotal roles: the architect and the developer. The architect is responsible for designing and supervising complex computer systems and software implementation. The developer brings the information system to life by meticulously planning, programming, configuring, and installing all necessary components. Typically, architects and developers work closely to achieve optimal results.

There are many different things to consider in software development, such as strategies, teams, roles, processes, and operating models. There is not fundamentally correct or incorrect approach to development. However, excellent developers must show an active interest in the entire development process and comprehensively understand it. They should also adhere to various best practices, such as code management, security, database handling, and testing.

The purpose of this thesis is to explore topics related to system production throughout the entire software development life cycle. The goal is to provide insights to fellow system architects and system developers, aiding their comprehension of the development process. Whether the reader is a system architect, a full-stack developer, a frontend coder, or a backend specialist, the thesis can provide valuable information for enhancing one's processes and professional work. This leads to faster product development, better quality, and more cost-effective processes.

The topics encompass various aspects, including conceptualizing application ideas, project management and designing, navigating development processes, managing data and databases, addressing security concerns, conducting testing, and contemplating future evolution. However, as previously mentioned, it is possible to approach software development in countless ways. Hence, this thesis is structured to reflect the author's perspectives and experiences, primarily within the domain of web-based software development relevant to the employer company's context. Nonetheless, the topics discussed are substantiated through reference to existing literature and established best practices. The level of detail in addressing these issues is reasonable, allowing developers the flexibility to make their own choices and decisions in their work.

The author's employer publishes valuable information websites used by hundreds of thousands of Finnish healthcare professionals and millions of ordinary people interested in their health. The production, functionality, and reliability of web services are essential to the entire company's business. The thesis topic is selected to enhance the development processes in the company. Written information, a guide, or a notebook, helps developers to unify work methods and make them work more efficiently. The author's motivation is to become a better architect by studying and reflecting on his thoughts, skills, and competence throughout the work.

2 Building a Successful Software

2.1 The Power of the Software

The rapid adoption of computers to help and support human labor has happened quickly. In nearly 50 years, microprocessor-equipped devices have conquered homes, workplaces, libraries, schools, stores, hospitals, and numerous other places. The most significant advantage of computers compared to traditional mechanic devices is their versatility, allowing users to do a wide range of tasks on a single machine, such as information retrieval, communication with other people, bill payment, book writing, gaming, programming, digital image creation, and editing. The rapid development of increasingly faster and more powerful computers has spawned virtually limitless opportunities in the digital world. (CHM 2020.)

The development of information technology has created its own industry dominated by multibillion-revenue companies like Google, Microsoft, Apple, and Amazon (Ponciano, 2023). Remarkably, Google was founded only 25 years ago, and Amazon just 30 years ago. They grew from garage companies of one or two people to giant companies; Amazon has nearly 1.5 million employees (Statista, 2022). Besides these multibillion-enterprises, hundreds of thousands of information technology companies have been born, developed, and grown to fulfil the needs of modern, digital-age people.

The growth of the technology companies has been staggering. Notably, all the above companies produce digital information systems and applications for everyday use. For instance, Google search has become synonymous with searching for information on the Internet, Microsoft Windows is the dominant operating system in desktop computers, Apple's devices and iOS ecosystem compete against Google's Android system on billions of mobile phones, and Amazon is for many the way to buy everything they need from food to furniture online.

Applications have become digital gold. An electronic product, application, or system, offered at the right time, in the proper context, and to the right people, has made its developers millionaires and can create new ways for users to engage, communicate, travel, and conduct their work. The significance of various applications in our daily lives is enormous because they practically surround everything we do. The natural question arises in a world surrounded by software: How are systems conceived, created, and maintained? Who are their creators, and how can they sustain long-term success?

2.2 Thousands of Ways of Developing an Application

There are so many types of software that there is no universally applicable or standard template for producing them. A program can be made and developed by one person as a hobby, or an engineering army of thousands of people can be behind it. The size of a program, as measured by the lines of code, can vary enormously - from a few hundred lines in some applications to an extensive 30 million lines, as seen in the Linux operating system kernel (Linux, 2023).

Some applications can renew themselves and develop under the pressure of competitors, while others may experience a flourishing period followed by declining use and disappearing from the software landscape. For example, according to Statcounter (2023), in the approximately 25-year history of Internet browsers, the leading stars have changed from Netscape to Microsoft Internet Explorer to Google Chrome. Even substantial resources supporting software development do not necessarily guarantee leadership. The application producers should be aware of the significance of comprehending the context, the users, and the purpose of the developed application. This is the process of building innovative software.

The continuous development of the Internet and Internet browsers has created an entirely new platform for creating applications. The use of browser technology is beneficial for both the provider of the service and the people using it. Information networks enable information retrieval, forwarding, and processing in near real-time worldwide. Browser-based applications, from basic homepages to complex ERP systems and online stores featuring vast inventories, are at the user's fingertips by simply typing the service's web address.

A modern web browser offers countless possibilities to implement a digital application, its appearance, functionality, data handling, and interactivity with the users as the result of the designer who created it. As is usually the case in any fashion, web trends come and go; different new things are tried, some of which are here to stay, and some are forgotten as gimmicks that work poorly and mostly annoy users. Today, artificial intelligence, mobile friendliness, dark mode, scrolling effects, bold typography, and gradients are hot (Hesterberg, 2023; Dribble, 2023). Application producers must understand and stay informed about trends, even if they choose not to adopt them. As the world changes, so should the application.

2.3 Know the Users and Their Needs

According to Standish Group (2023), a software development consulting and research firm, yearly studies show that about 30% of software projects are successful, and of these successful projects, only about half return high value to the stakeholders. Although these numbers can be viewed critically (like Eveelens & Verhoef, 2008 did), the planned costs of many software projects multiply, the time is spent significantly more than expected, and above all, the results do not meet the users' specifications and needs. Finnish electronic health and social record "Apotti" has spent about 230 million euros more than expected, 40% more than initially estimated, the implementation of the system has been delayed, and the users do not find the EHR system easy to use (Kolehmainen, 2022).

Koikkalainen (2023) proposed various activities to promote the success of the application development process. No single factor is responsible for the success or failure of a project, but the combined effect of many factors. Primary success criteria are communication, project manager capability, and successful risk management.

Ease of use is one of the goals of most applications. However, many software, even those built with millions of euros, still cause problems and headaches for their users because their usability could be better. Especially in business applications, like ERP's, financial software, and HR management, several systems are similar in their usability, appearance, and functionality to the applications created over twenty years ago, compared to modern, genuinely easy-to-use applications that meet users' needs.

One of the main reasons for failed products is the design gone wrong. Not enough effort has been put into it, or it is based on wrong premises, assumptions, and insufficient information. Application designers and developers might consider the systems with their needs and concept models rather than the users.

The paradigm of good usability and user-centered design has been around for about as long as personal computers have existed. Donald Norman's books (1986, 1988) laid the groundwork for distinguishing between good and bad design. In the '90s, Jakob Nielsen (1993) summarized sound usability principles as learnability, efficiency, memorability, satisfaction, and low error rate. ISO Standard 9241 (ISO, 2018) adds effectiveness. Moreover, of course, there are more, like utility and likeability. Despite the differences in the views of different schools of usability and good user experience, it is essential to recognize that the program's suitability for its task can be studied and examined.

The earlier the application idea is tested with various design papers, prototypes, and other methods, the easier it is to make changes and corrections. Overall, changing the functionality once the application has been released to production would be significantly more expensive and complicated. The cost of defects multiplies to even 100 times when identified and fixed late in the deployment and maintenance phase, compared to early requirements or design phases. (Tassey, 2002.)

2.4 Building on a Solid Foundation

The possibility of creating new releases is both a plus and a minus of electronic software. The application is changed, improved, fixed, and then released as an updated version. That has just been installed and used in place of the previous one. Suppose new features and functionalities are added to the application uncontrollably. In that case, the functioning of the whole can get weaker, especially if the system's technical architecture is not in order.

Enterprise architecture defines the architecture of the complete company. It aligns business and IT together. Solution or functional architecture describes the functionalities needed to meet business objectives. Technology or system architecture describes comprehensively the structures and operation of the various components of the system. Software architecture focuses more on the factors that are significant for the application functionality, such as technical and business requirements, application framework choices, database solutions, and building user interfaces (Widjaja, 2022; Satayabrata, 2023.)

Although all architectural terms have nuances, in practice, good experts understand and know how to consider the whole on many different levels and do not build artificial boundaries between architectural terms.

The technical architecture has a significant impact on making software. Considered and justified solutions, like programming languages, application frameworks, data warehouses, production deployment, cloud service usage, browser and mobile application development, maintenance, integrations, documentation, and testing, make application development faster, more efficient, and better quality. Above all, architecture means technological choices that fit the company's goals, strategy, and operations that produce the application. These architectural choices and decisions have long-lasting effects, among other things, on what kind of resources and knowledge are required to maintain the technological entity.

Resourcing a company's IT management and application development can be implemented in many ways, and there are certainly as many ways as there are companies, depending on their size, technological strategy, products, and competence. A large company may need several levels of management, administration, and the executive ladder. In a small company, technical management, know-how, expertise, and doing can be the responsibility of one person. In terms of the systematic development of applications, the roles of designers, architects, and developers are vital, without discounting project management or the contribution of sales and marketing professionals to the success of the entire product.

The practical implementation of applications is mainly the responsibility of the architect and developer. When implementing even a slightly more versatile program, countless issues must be considered, including those not considered in the planning process. The tasks of the application developer can be the design and use of the database, the implementation of integrations, the user interface, testing, and documentation. If a single person can make background systems, the database, and the front end, like the user interface, this person is called a full-stack developer. Especially in smaller companies, people more easily end up as experts in the entire technology stack. Some people specialize in a narrower area and know it well.

Good application development professionals think about software production holistically, from the first idea to implementation and continuous development. It is crucial to have a process that systematizes the work. Many companies have diverse ways of managing and implementing the software. IEEE Standard Glossary of Software Engineering Terminology (IEEE Std-610-1990) defines the Software development life cycle (SDLC) as the period of time that begins when a software product is conceived and ends when the software is no longer available for use. So, SDLC defines the various stages of production that have been appropriately identified and can be developed (See Figure 1).

The most well-known SDLC models are the waterfall, the spiral, and the agile development model. In practice, all models include typical phases like requirements analysis, planning, architecture design, implementation, testing, and production deployment. (IEEE, 1990; Leon, 2015.)

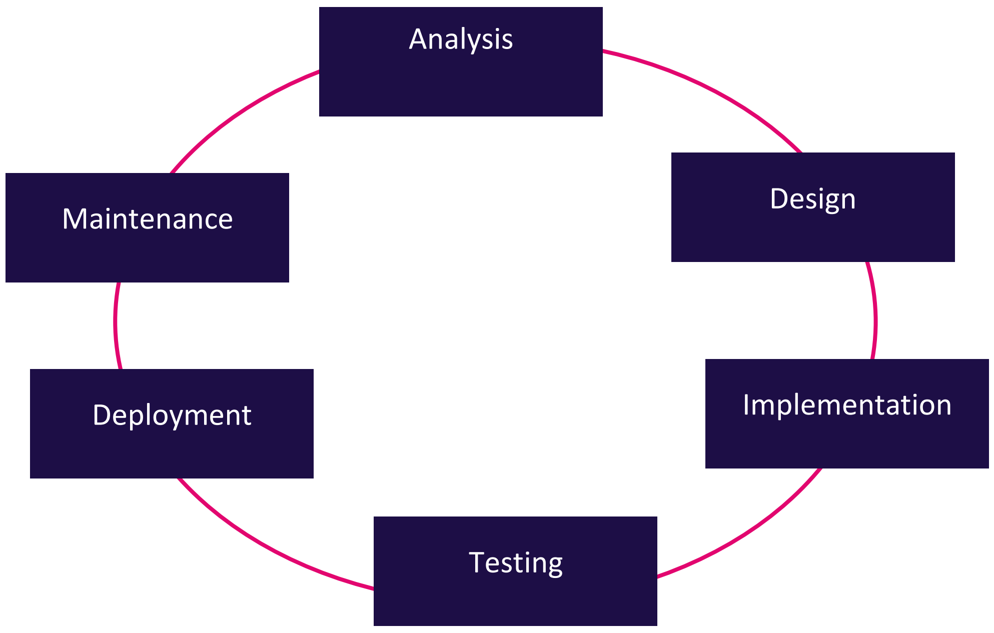

Figure 1. Software development life cycle (Chavathe, 2023).

In addition to more or less structured development models, countless best practices in application development apply to user interfaces, usability, database solutions, security, and performance. Good practices can apply to general technical solutions, such as normalizing a relational database or the company’s internal operating methods, like managing passwords. Knowing and using various good and best practices will make program development more efficient, faster, and overall better because, with them, application structures are likely to be more durable, more precise, and maintainable. Ignoring the best practices of information security can be a significant threat to the software and even the company that produced the application.

Waddington (1995) already described in the early days of the Internet that data and the information refined from it are often among the most critical assets in a company, giving many businesses a competitive advantage. Patient information records, tax systems, banking systems, library systems, online shops, social media, and many websites rely on massive information processing. From the point of view of application development, it is essential to know where and how the information is stored and how it can be used by searching, modifying, and removing. Data warehouses, either traditional relational databases or newer object-based databases, are vital tools for data processing, and application developers must know how to use and utilize them according to the application's needs.

The fast-paced change in information technology challenges the people responsible for application development. Today's technology may be a thing of the past tomorrow. For a professional application developer, continuous learning, training, developing skills, and the development of technology knowledge is very important. The role of a system architect and full-stack developer includes understanding and being able to take over new technologies and utilize them smoothly and in a controlled manner.

2.5 Summary

Application development is not rocket science but requires expertise and know-how in many different areas (See Figure 2). Competent people who understand not only the possibilities of technology but also the users' needs and business goals create high-quality and functional applications. A controlled application development process and a supportive, solid architecture foundation contribute to successful applications.

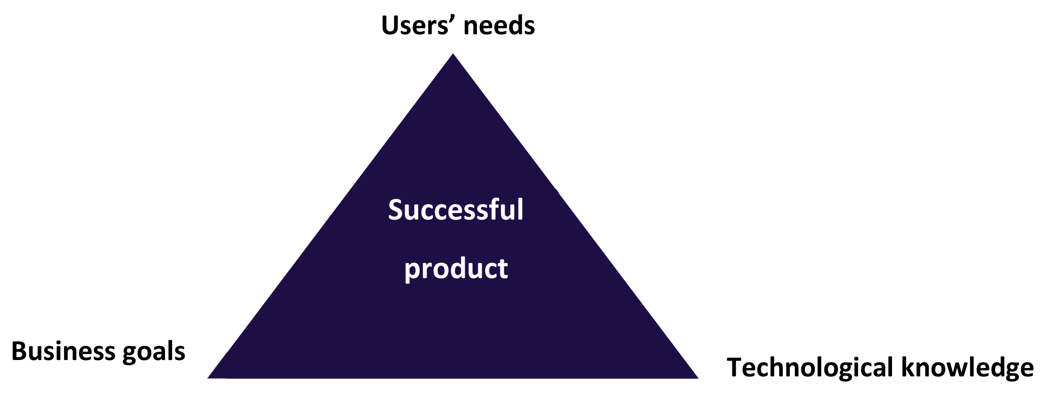

Figure 2. Building a successful software product based on user's needs, business goals, and technological knowledge.

3 Research Focus

The commissioner's technological infrastructure has transitioned over the past few years. Old technologies have reached the end of their life, and the support of their manufacturers has ceased. New and fresh solutions have been developed based on modern technologies. At the same time, employees have changed, bringing business and technical challenges to converting existing systems to new platforms. The new structures and solution models of software development have had to be developed together with the applications themselves, and it has brought other challenges in the demanding environment. The architecture and implementation models have reached a stable state where they can be utilized and further developed.

In this thesis, a practical focus is placed on modern software development during the entire application life cycle, from the idea to the end of the product. The development work aims to describe and present things that system architects and application developers should consider when creating a uniform, maintainable, and best-practice program. The goal is to improve, enhance, and support the commissioner's software production process, where development methods, solutions, and concepts are familiar to everyone.

As research work, the following are the essential questions:
1) What is an effective process in software development?
2) How can technical architectures aid and support application development?
3) What aspects should application developers consider in software production?

The outcome of the work is a notebook or manual that allows full-stack application developers to understand the conditions related to development work and the importance of business and user needs, implement their application development environment in harmony with others, and utilize other good practices in the application development process.

The work describes issues related to the application development process primarily to solutions relevant to the commissioner organization, and of course, they are only suitable for some organizations. For instance, cloud services are not extensively used and, therefore, not discussed in detail in the notebook.

4 Research Implementation

4.1 Methodology

The thesis research is a variation of the diary-based method. This implementation method is suitable for persons who are strongly engaged in expert duties in their work. The data is usually collected in a diary thesis as the work progresses. A diary-form thesis represents autoethnographic research. (Jamk 2023). Adams (2017) describes autoethnography as a research method that uses personal experience (“auto”) to describe and interpret (“graphy”) cultural texts, experiences, beliefs, and practices (“ethno”). The diary-based approach provides a more convenient way to link daily work activities.

The author of this thesis has over twenty years of experience designing, developing, and maintaining various software specialized in web-based health care information systems. In a small company, the tasks have been similar to a full-stack developer’s work today, where a single person designs, implements frontend and backend, tests, and maintains the system. Over the years, as the company has grown, the development team has also increased, and the author’s role has become more like a system or software architect, where one task is to aid and support the development team members in their work. Overall, this fits well with the diary-based methodology.

The author picks up a weekly theme or topic of software development, describes it, gives some thoughts, and expands the views based on literature research and best practices. After 12–14 weeks, a coherent “notebook” is gathered. Focusing on one theme per week should make the writing process more manageable. The preliminary list of topics is listed in *Table 1*. They are selected based on a typical software development project and the author’s experience with what issues and questions typically arise in an application development project.

Table 1. List of main topics covered in the thesis.

Application idea
System architectures
Planning and design
Development processes and methods
Development environments
Implementation and coding
Data and databases
APIs and integrations
Security
User and access management
Testing
Production usage
Future growth and evolution

4.2 Data Collection and Analysis

The primary target group of the development work is the commissioner's Information Technology department, its developers, and management, who benefit most from the definition of operating methods and analysis. Secondarily, the work also positively affects the entire organization's business when the operation of the IT department is enhanced and improved. IT's position, which crosses the organization's unit boundaries horizontally, gives broad perspectives and opportunities for developing the entire organization's operations.

The research material includes:

- existing IT documents and memos related to application development;
- material collected by interviews and surveys of application developers;
- source codes of various applications;
- the author's archives, emails, and other material created through the creation of the work; and
- the literature related to the topic.

The existing documents are mainly in Microsoft Teams Channels of the IT department or the entire organization. Personal notes are taken from the software developers' interviews, and the survey results are saved in Microsoft Forms. The source codes are in the version control system and the database. The author's archives include emails, OneDrive files, and Teams. The selected literary material is gathered as the work progresses. The author has access to all relevant materials and information without special permission. The data management plan is in the Appendix 1.

Before starting the notebook writing process, the status quo is researched by conducting a small survey and interviews with the developers. A survey gives a formal picture of the current and target situation when the answers are quantitatively measurable. There are about a dozen people in information management, all of whom should participate in filling out the survey. Due to the small number of people, the purpose is not to do a unique or complex statistical analysis but rather to find out the current competence of developers and the challenges in their work and direct the writing of the notebook to the things that need the most support. The depth of various topics can be determined based on the results obtained. The survey results are also intended to benefit information management decision-makers from the perspective of developing skills and training. The planned survey questions are presented in Appendix 2.

There is also regular communication between the author and the developers, so many things will undoubtedly come up that can be considered during the thesis writing process.

4.3 Schedule Plan

The thesis is written along with other work. There are about 12-14 topics to be covered, and the plan is to write about one topic per week. It would take 3-4 months to be completed. One topic writing should be about 5-6 pages, which is a little, but would cover the most essential things the developer would need in the first place. Plenty of references should help to discover more about the theme.

4.4 Ethicality and Reliability

“Don’t be evil”. This has been Google’s unofficial motto in its corporate code of conduct since 2000, which was removed years later (Google, 2000). The text said, “It’s about providing our users unbiased access to information, focusing on their needs, and giving them the best products and services that we can”.

Although the current status of Google’s significant technological power is questionable in many respects, the idea of the young company’s goal to make its products on the terms and needs of the users is nevertheless one of the reasons why Google is what it is today. The company has considered the user’s needs, outlined clear business goals, and hired the most capable technology gurus.

IEEE Computer Society (1999) lists eight main ethical principles for software engineering:

- act consistently with the public interest,
- act in a manner that is in the best interests of their client and employer,
- ensure that their products and related modifications meet the highest professional standards possible,
- maintain integrity and independence in their professional judgment,
- managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance,
- advance the integrity and reputation of the profession consistent with the public interest,
- be fair to and supportive of their colleagues and
- participate in lifelong learning regarding the practice of their profession and promote an ethical approach to the practice of the profession.

Because we are rounded with software, they can influence, control, manipulate, and change people’s opinions. Software is used for many evil things: spam, viruses, ransomware, spyware, and others cause much trouble for users. A good IT professional should know the ethical principles and implement the software accordingly. The principles are also acknowledged in this thesis. Ethical sustainability is also sought alongside best practices when addressing and displaying topics.

While many tools in software development, programming languages, server settings, and database queries must often be meticulously accurate, software production is still not an exact science. There are many things to consider, several ways to do things, and different outcomes depending on these factors.

The thesis will propose some approaches to software development. They are based on a mix of author's experience and shared best practices. The chosen topics are universal in developing applications, but the focus is still on how things are done in the commissioner's environment. This limits the usability and partly reliability of the work in general, but on the other hand, it focuses it to suit the employer's needs.

5 The Idea Analysis

All the great things start from the idea. Tim Berners-Lee wanted to organize documentation and make it more accessible via hypertext and a browser (Berners-Lee & Cailliau, 1992). Sergey Brin and Larry Page had an idea of how to find relevant results from the Internet (Brin & Page, 1998). Steve Jobs brought us mobile phones with touch screens. Jeff Bezos saw a business idea by selling books online. Daniel Ek and Martin Lorentzon created an application to stream music from the web rather than download and own the records.

All these ideas revolutionized our digital world, and millions of others have developed and formed into working applications and software. Although not every bright idea produces multibillion-dollar products, services, and enterprises, innovations emerge in everyday business.

5.1 The Vision

So, what makes the idea worth looking at? The analysis phase is essential for structuring and evaluating the idea and defining the scope of work. There are multiple things to consider for the idea's feasibility, like market, economy, schedule, and technical feasibility. Businesspeople can do market research, evaluate the cost-benefit ratio, compare to competitors' products, check possible target days, and so on. IT specialists must consider technical feasibility. When IT works together with the business, it results in the best possible outcome.

The beginning is the best place to challenge the visionary mind. Is the idea clear enough to continue to the next phase? The most important thing is to listen and ask questions. These questions are not meant to shoot the idea down unless the project is immediately recognized as undoable, but rather gather information about the case and help understand what the concept is all about.

Good questions to ask are like:

- What is the main point of the idea? If this is not clear enough, it would be hard to make software from it.
- What is the idea's overall purpose, goals, and objectives?
- Why is this idea relevant?
- What problem does it solve?
- Who are the users, and how do they benefit from the idea?
- Who are the other stakeholders, and what do they expect?
- How does the idea reflect to business?
- What is the business case? Does it have good value?
- Is the idea aligned with the business strategy?
- How much money is available?
- When the application should be ready?
- How to measure the success of the idea?

These high-level questions help to produce a vision for the targeted software. The vision is a high-level description of the application and what it aims to achieve in the long term. It ensures everyone understands the project similarly, aligning the development team, stakeholders, and users toward a common goal (see *Figure 3)*. The vision serves as a guiding principle and an overarching goal for the development and evolution of the application. It is the basis for the direction and decisions related to application features, design, and implementation.

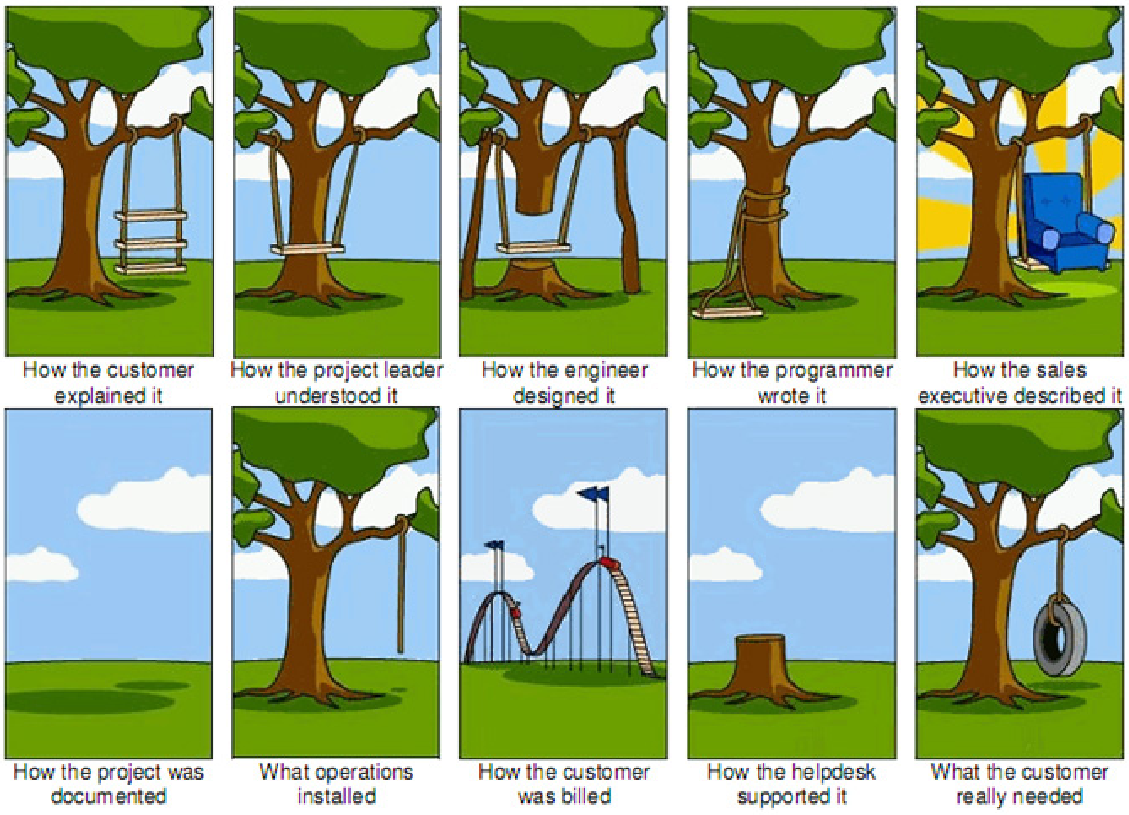
How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

Figure 3. The classic tree swing cartoon emphasizes the pitfalls of the communication and interpretation of the product. Its original source is unknown.

5.2 Technological Analysis

The importance of the analysis phase cannot be understated. The decisions made, requirements gathered, and interviews with stakeholders create a solid basis that can last for years and be the foundation of the software throughout its life. It would be easy to jump directly into technology issues, as developers love to solve problems by coding. Too often, they try to get started without proper feasibility review, architecture thinking, and even precise planning and designing. There is no need to rush into the development phase. The time spent at the beginning of the process reduces the time and money spent later. (Khan & Kumari, 2021.)

A technological feasibility check is crucial in assessing whether a software project or application is technically viable and can be realistically implemented. After the high-level questions and a glimpse of the vision for the future, IT experts can do the technology evaluation. Depending on the target, the technical analysis can take time, as today's systems are complex, and even new software is usually more and more dependent on existing software, integrations, and information. The main tasks for technical analysis are identifying functional and non-functional requirements, reviewing technological infrastructure, thinking about use cases, and, therefore, defining the technology scope of the project.

Core functional requirements are the things and features the software offers the user and what the application does. Non-functional requirements are related to, for example, performance, reliability, scalability, usability, and security issues (Laplante & Kassab, 2022).

Technological infrastructure review involves servers, platforms, other software, integrations, databases and data warehouses, frameworks, tools, and other aspects of the company's technology environment. Does the idea fit into existing infrastructure, or is something else needed? It is better to reuse existing solutions before buying or building new software. So, if something similar exists or is easily adjustable, it's better to use it.

Use cases in their first form explain generally how the application will be used: What are the main tasks users usually do with the software? This should not get yet into the details, but it helps to define the scope and again, increases understanding of what will be needed. Good analysis can also point out things that are left out.

Companies live on data, so evaluate the currently available information. Many ideas fall short because of the need for more data. Existing information, if it's usable, structured, and correct, is the best possible thing to start with. Some data might need to be gathered before the project begins, and some information is gathered during or after the software is ready and in production.

It is also wise to think of any technological metering for the application. How does one measure the success of the application? What are the "Definition of Done" criteria for this case? DoD represents the criteria or conditions that must be met for the application to be considered completed and ready for delivery or deployment. DoD should be in line with the vision.

A good outcome from the analysis phase is a document describing the project's facts: requirements, stakeholders, risks, resource and timeline estimations, and reasons to continue or halt the project.

5.3 Idea Jar

In innovative companies, ideas storm, and new things pop out faster than previous ones are even analyzed. Not all ideas pass to the next level. It is still important to document the concept and describe why it was put aside this time. The timing may be wrong, the resources needed were too big, or the technology was not mature enough.

Documenting the ideas in the idea warehouse is valuable because it is common for them to come up again after some time. Indeed, something might have changed since the last review, or there might be something new so that the idea can be evaluated again. But it is easier to do with some prior knowledge available.

5.4 Summary

The analysis phase is essential for evaluating and defining the scope of an idea. When the project starts, the vision serves as a guiding principle for the development and evolution of the application. Businesspeople do their research, and IT specialists must consider technical feasibility, gather functional and non-functional requirements, and do a technical infrastructure review to define the technology scope of the project. The time spent at the beginning of the process reduces the time and money spent later.

6 System Architectures

Software is created for many purposes, systems, devices, and users. For web applications alone, there is a wide range of implementation possibilities, as the programmer has the choice of programming language, data repository, user interface framework, development processes, programming environment, etc.

Even programmers in the same company can use different solutions to develop systems. But this freedom can come at a high price. Challenges can arise in skills, system maintenance and upgrades, resource management, integration implementation, security threat management, development cost estimation, project lead times, user and customer satisfaction, and a common understanding of how things are done. Different solutions also tend to multiply the problems.

Controlled system development requires a systematic approach and standard rules of the game. An information system architecture is needed to support these. ISO 42010 (2011) defines architecture terms as follows:

- fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and the principles of its design and evolution, and
- conventions, principles, and practices for describing architectures established within a specific application domain and community of stakeholders.

In other words, architecture defines the principles for designing, building, and managing systems so that the whole works as intended, scales, and responds to changes. The IT architecture guides how things are done, for example, through decisions on common technical solutions, data, functions, tools, and processes.

The IT architecture must align with business objectives. The IT architecture uses technology to provide technical and systematic solutions to an organization's business challenges. From a business perspective, it is about the strategic issues of an organization's technology infrastructure, including software, servers, networks, databases, procurement, IT partners, integrations, and how they all work together. (Widjaja, 2022.)

The architecture defines the structures and interconnections at a high level so that the guidelines can guide practical implementations. Because architecture plays such a crucial role in the development of applications, it must be well thought out, reliable, and robust. The architecture will also need to change over time, but at its best, its basic principles will work throughout the life cycle of most applications.IT architecture promotes the efficient design, development, and maintenance of systems, supporting the organization's business objectives in the best possible way. The hallmarks of a sound systems architecture include:

- Clarity and simplicity: modelling a complex real-world environment in a simple yet functional way is challenging but provides the basis for sustainable application development.
- Flexibility and extensibility: the architecture should allow easy customization and extension to meet future needs.
- Reliability and stability: the application should be reliable and stable in different situations.
- Performance: good performance and scalability are essential, especially as usage grows.
- Security should be considered at all levels.

- Modularity: Breaking down functionality into smaller modules makes development and maintenance more manageable.
- Testability: detecting and correcting errors and problems as early as possible is vital for quality.
- Clear and comprehensive documentation helps understand the implementation, maintenance, and training.
- Cost-effectiveness: systems and their maintenance are often expensive, but good architecture can make a difference to the resources used.
- Business-oriented: the architecture must be based on a genuine business need, on solutions that serve customers.

Practical examples of the impact of good architecture include:

- Faster development by making use of ready-made existing structures.
- Better maintainability of the system when you know how to use, modify, and repair its components.
- The system will likely scale better when its foundations are well thought out.
- Risks are easier to identify and manage.
- Collaboration and interfaces are more usable and producible when the architecture is considered in a larger context.
- Decision support when questions that need answers arise in developing systems.
- Consistency, using standards and de facto standards, is essential in making things repeatable and automated.
- Overall, a more cost-effective way of working.

6.1 A Variety of Architectural Models

Information technology architecture is a vast complexity when considering the technical infrastructure of an entire organization. Therefore, architecture work and solutions are often divided into subcategories: enterprise architecture, solution architecture, technology architecture, and application architecture. See *Figure 4.* (Widjaja, 2022).

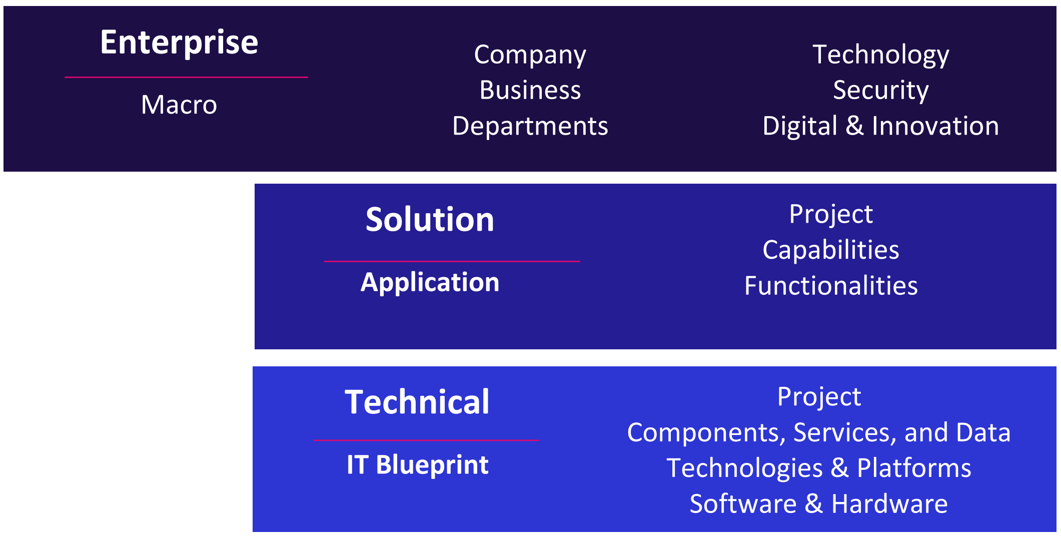

Figure 4. Various architecture models hierarchy (Widjaja, 2022).

The enterprise architecture describes and defines the entire corporation. It includes all applications, services, functions, data flows, and integrations the enterprise uses. The enterprise architecture aims to conceive and align IT functions within the context of the company's business, strategy, and objectives. Defining the enterprise architecture requires knowledge and understanding of the business, industry, and technology solutions.

As the name implies, solution architecture focuses on the technical requirements and solutions for systems to achieve business objectives. It defines the necessary technological functionalities for software implementation.

The technology architecture describes in more detail the different technological components of the systems and their interrelationships. Compared to a solution architecture, a technology architecture addresses system-level issues such as servers, networks, software, and data warehouses. Application architecture focuses solely on the application, where it guides software development, e.g. in terms of user interface functionality.

In practice, different architectural terms can be confusing, and the same people can make decisions and perform tasks related to several different sub-architectures. In large companies, the boundaries of architectural roles can be more precise when there are several experts. Still, in this thesis, we will talk about IT architecture, architect roles, and developers.

While the infrastructure used to build any application can be called architecture, excellent and sustainable architectures do not emerge alone. The role of the IT architect is to identify, define and design a business-oriented framework for implementing systems. The architect's role also includes overseeing and supporting application development to ensure the implementation conforms to the architectural framework. Understanding the big picture, identifying technology opportunities, systematic design, and implementation, leveraging best practices, and collaborating with multiple stakeholders are essential qualities of an architect. IT architects usually have a solid technical background, e.g. as an application developer, and a business understanding to ensure that technology and business are aligned.

Architectures are, therefore, the first and foremost choices. It is not the latest and greatest technology that determines the architecture but the business and its needs. However, a good architect sniffs out and explores the potential of new technologies to replace old systems that may be at the end of their life cycle. A systems development roadmap helps you understand both the big picture and where it is heading in the future.

Different paradigms from different eras influence application development. In IT, many things are changing and evolving so rapidly that riding the wave of development can be an excellent opportunity for business development but also a significant risk if you get too attached to a technology that later turns out to be a dead end. Even big companies have failed when they made wrong or bad technological choices.

Therefore, architectural decisions should be based on the best possible knowledge, skills, and understanding. One good way is to bring together a group of experts, not necessarily all architects, to consider the issues relevant to the whole. The purpose of such architectural boards is to take a position in architectural matters, support application development projects, and ensure the practical implementation of commonly agreed technical solutions.

6.2 Modern Software Architecture

Gone are the days when applications were built as one extensive monolithic system, with all components and functions, the user interface, business logic or database processing, designed and tied into one code base. However, as the complexity of the systems grows, building, maintaining, and distributing such an application becomes increasingly challenging and more expensive. There are still countless monolithic applications worldwide, especially in the corporate world, e.g. in banks and insurance companies, CRM, ERP and HR systems. These applications are complex to replace because the systems are large and often business-critical.

However, the architecture of modern applications is entirely something else. Systems are complex and have broad data requirements, globally distributed deployment, a rich user experience, high-security requirements, and overall flexibility and scalability, all affecting the implementation of applications (Gorton, 2011).

Although today, three-tier architecture cannot be considered a modern paradigm, it is still one of the best architectural paradigms, where the functional parts of an application are divided into a presentation layer, an application/business layer, and a data layer. Users access the application through a user interface communicating with the business logic layer. The business logic layer, in turn, passes requests to the data layer, which handles its way of retrieving and storing data. The advantage of a three-tier architecture is modularity, where different parts of the application can be better implemented and maintained independently of each other. The various layers can also be allocated the necessary resources according to the situation. For example, if a database server runs out of steam, its resources are increased. Three-tier architecture is a reasonable basis for traditional systems, which are still the majority of applications being built today.

More and more applications today run in a web browser. Modern browsers such as Google Chrome, Microsoft Edge or Mozilla Firefox offer countless possibilities for implementing a rich user interface. The development of essential technologies such as HTML, JavaScript and CSS has been phenomenal in recent years, and it is now possible to build even large applications using only browser technologies. Because browser technologies are so versatile, there are plenty of skilled web developers. On the other hand, it is very different to make a website like a home page than an enterprise-level application, so web developers often specialize as front-end coders. The good thing about browser technologies is their shallow learning curve. The more knowledgeable and skilled the developer, the more familiar and capable they are in using the implementation features of browsers.

The competitors to browser technologies are native applications on mobile devices, desktop software, and software technologies for embedded systems. Even these boundaries are no longer so clear, as users can launch browser-driven applications in the same way as native applications. In modern application development, knowledge of browser technologies is a significant advantage.

The user interface layer needs data, i.e., content users can search, view, add, edit, and delete. Initially, the only way to pass data between the browser and the server was to fill in and send form data, meaning the page always needed a reload. The concept of Ajax, or Asynchronous JavaScript and XML, developed by Microsoft, significantly changed the whole production of web applications. Ajax made it possible to send and retrieve information without reloading a page, making the application experience much better and much richer in functionality.

The so-called CRUD model, or Create, Read, Update and Delete model, generally describes the basic operations that can be performed on data. When using a browser interface, middleware techniques are needed to pass these operations to the backend services. Unsurprisingly, various options for implementing these middleware services exist, including Java, Python, PHP, ASP.net, Ruby and Node-based solutions. There are dozens of popular and widely used middleware implemented in Java alone. Node.js was a significant newcomer to the middleware scene. Google's V8 engine-based runtime environment was not the first JavaScript-based system designed for servers, but it was the first to solve critical technical problems that server software could not at the time. Node.js is an OS-independent runtime environment that provides the tools and functionality a web developer needs to build a middleware layer (Node.js, 2024). Using the same programming language in different application parts is valuable and efficient for application development.

Communication and data transfer between the browser-based interface application and the middleware is done according to agreed rules using application programming interfaces. APIs provide applications with access to information from the backend systems. There are also many options for communication protocols, such as REST, GraphQL, SOAP, gRPC, OData, etc. REST (Representational State Transfer) and GraphQL are the most popular for building modern web applications. The advantages of REST include simplicity, statelessness, scalability, different data formats and reliance on basic web technologies such as HTTP and URL syntax. GraphQL has emerged as an alternative to REST because it provides the functionality to transfer only the necessary data, easy extensibility, data typing and efficient use of resources.

Data is the new gold (Li, 2023). Virtually all applications need data, such as user or product information, to function, whatever the application is doing. Beneath the surface is a data warehouse, which does the searches, changes, and other operations.

Traditional relational databases such as Oracle, MySQL, and Microsoft SQL Server have long dominated data warehouse solutions. Relational databases are based on organizing data into different tables, between which various relationships are defined. For example, a customer places an order that involves one or more products. Data is queried from the relational database using the SQL language. Data integrity and correctness are of paramount importance in a relational database.

An alternative to relational databases is to use so-called NoSQL databases such as MongoDB, CouchDB or Redis. The data storage structure of NoSQL databases differs from the strict predefined structure of relational databases. In contrast, a NoSQL database can store various data that may evolve. NoSQL databases also have the advantage of good scalability and decentralization, which is useful when building a global application. 100% data integrity is not the priority for NoSQL databases, but scalability and high throughput.

The choice of a data warehouse is often an important strategic choice. All major data warehouse vendors are constantly developing and improving their products. So, for modern application development, using relational and NoSQL databases is a viable solution, and it is more a matter of the application's functional logic and needs, which model you want to use. The choice is also influenced by existing skills, the cost of acquisition and maintenance, and the vision for the application and ideas for future development.

6.3 Cloud Computing Architecture

Cloud computing cannot be ignored when considering modern architecture. The cloud is a set of off-the-shelf solutions and infrastructure that a company can use to deliver its services. When a cloud provider manages its infrastructure, a company can focus on its core business without acquiring its own servers and maintenance expertise, which means using the cloud can be more cost-effective.

Several different providers offer cloud services. Big ones include Microsoft Azure, Amazon Web Services and Google Cloud, each offering a breathtaking range of solutions to meet business needs. However, there are countless different providers, from generic solution providers to specialized functions, so one must consider their suitability for your business before committing.

The benefits of cloud computing include scalability, cost-effectiveness, automation of updates and many maintenance tasks. The downsides of cloud computing can be security and data location issues, limited configuration options, and even significant cost increases as space, data, and computing power increase. The location of data centers can play a role in data latency. Storing critical data in the public cloud is also always a risk. However, the organization will likely invest in data management and implement business-oriented end-to-end solutions, including staffing, technical infrastructure, and application services.

Hybrid solutions, or semi-clouds, are also common. It allows operations to be decentralized and organized as required, rather than everything in the cloud or on-premises. Flexibility is essential, and IT should adapt to business needs, not vice versa.

6.4 Summary

The system architecture is essential in controlled software development. It provides the foundation for designing, building, and managing systems. Technical architecture should always align with the business objectives while delivering high-level guidelines for practical implementations. Sound architecture embodies many qualities: clarity, simplicity, flexibility, extensibility, reliability, stability, performance, security, modularity, testability, and comprehensive documentation. Good architecture impacts several areas: faster development, better maintainability, improved scalability, risk management, better collaboration and communication, decision support, consistency, and cost-effectiveness.

7 Design of Systems

The next step after analysis in the software development life cycle is design. The analysis, or definition, tells you what and why the system is being implemented. The critical question in the design phase is how to implement the system successfully. Unfortunately, people often neglect planning and move to implementation too quickly.

Software planning can be divided into project and resource planning and implementation planning. Project planning involves allocating the necessary resources, such as people, systems, meetings, costs to produce the software, assessing risks, and planning schedules. Technical implementation planning includes describing the architecture, infrastructure, processes, components, user interface, database, interfaces, integrations, and security. If existing systems, components, processes, and methods are usable, the more efficiently the software can be implemented - provided that the existing solutions are of sufficient quality and flexibility to serve as a basis for new implementations. Building a house of cards with poorly documented, obsolete technology and unmaintained components should be avoided.

Technical design should involve architects, application developers and specialist designers, together with the business and users of the future system. User-centered design, service design and overall user-centeredness are the cornerstones of a successful application.

The technical design of a system starts with the top-level system architecture:

- What information exists, in what format, what is still needed, where is it stored, and how can it be accessed?
- What kind of interface is needed, and what are its main functionalities?
- What technology solutions will be used to build the system?
- What automation or manual processes are involved in the operation of the application?
- What existing resources are available and can be used?
- What methods will be used for implementation?
- How will security be taken into account in the implementation?

- Where will the system be installed and used?
- How will the system scalability be ensured in the face of potential growth in use?

Information is the foundation of many systems. That's why it's worth taking the time to understand, process, store, and format it. It is good to think about and plan for access to data. Do you need real-time data processing? Do you use batch processing? Where is the data available in the first place? Data can be in a structured format, e.g. database, JSON, or XML formats, which is likely easier to process than unstructured formats such as text files, word, or PDF documents. Many systems are also hybrids of these, where well-structured solutions help to find unstructured information. Information always has its quality, how it can be used and exploited in a system. In general, structured information is of higher quality and more accessible to process and edit. It can also be enriched, supplemented, and, if necessary, modified compared to its source. A good example is library systems, where metadata such as title, author, publisher, year of publication and ISBN of books are stored in the information system to support volume retrieval and lending.

The user interface is often the most challenging to design. However, it is the primary means of accessing and using the information underlying the system. A good user interface supports the user, improving productivity, making use more efficient and increasing engagement with the system. A poor user interface spoils the user experience, slows down, and creates errors, misunderstandings, and other problems in the processes that the system is intended to support.

Ease of use is a quality that is often cited as a software goal, and designers do not usually intentionally create poor user interfaces. Yet countless programs have the most incomprehensible user interface solutions. Its design should be guided by user-centered design, consistency of function and appearance, clarity and simplicity, efficiency, uninterrupted feedback to the user on their actions, and tolerance and recovery from errors. There are many general good practices in user interface design, which, if followed, will lead to an exemplary user interface that supports the user's goals.

The design phase also involves a lot of technology choices. What programming language to use, what framework, what database, whether to use cloud services, how to test, etc. All these choices impact the system's implementation, maintenance, and further development. If any significant component turns out to be a wrong choice, the whole architecture of the application will suffer. Therefore, the technology solutions chosen should be proven, continuity-tested and maintained systems.

It makes sense to leverage existing services, functions, and solutions, speeding up development and focusing on the essentials, usually the core functionality of the application in question. For example, user authentication and management can be outsourced to a user directory such as AzureAD rather than doing the same in the application itself.

The choice of implementation methods is also essential. Nowadays, various agile development methods are favored, where work is divided into small parts and focused on for a week or two. The advantage of agile is that it adapts to changing needs and requirements, iterative development, aiming at rapid results, and improves with feedback at each cycle. These reduce the risk of failure of the whole project by allowing a change of direction and speed as needed. Conversely, agile loses some predictability regarding when the work is done, whether the scope remains manageable, and whether the team developing the system can remain cohesive and motivated.

Information systems are built so that data can be stored, processed, and edited in an automated way. Machines perform routine tasks at many times the speed of human manual labor. Therefore, when designing systems, it is important to consider what can be automated and what people should do themselves. Automation requires an understanding of how things work so that it can be modelled as an automation process. Therefore, it is best to learn the process manually first, especially for new systems, and then develop more efficient automation. Complex automation technologies are often challenging to maintain, so their functionality needs to be well described.

The importance of information security in the system development cannot be underestimated. It must be taken into account in the implementation of the system from the beginning because it is not something that can be easily added to the functionality of a program afterwards. A coherent approach to security builds a secure, resilient, and low-risk system. Investing in security and data protection from the outset minimizes the cost of errors and problems, better applies legislation and regulations, protects critical and sensitive information, better identifies and models threats, and generally makes the system architecture secure throughout, including for potential new, as yet unidentified threats. Because, indeed, new security threats will constantly emerge.

It's crucial to take the time necessary to design the technical infrastructure, especially if the application is developed in a new environment. Existing infrastructure, such as servers, communication solutions, and operating systems, provide ready-made solutions as a platform for the application. When considering the technical infrastructure, it is also essential to consider the scalability of the whole, e.g. in terms of increasing usage, data, or other resource requirements. For example, the statefulness of interfaces, where middleware software handles requests autonomously and atomically, is important if applications are distributed across multiple servers.

The architecture guidelines provide a framework for the application to be designed and implemented. The simplified model of modern architecture application is presented in *Figure 5*. More detailed plans are needed for the data warehouse structure, interfaces, user interface, data handling, and security.

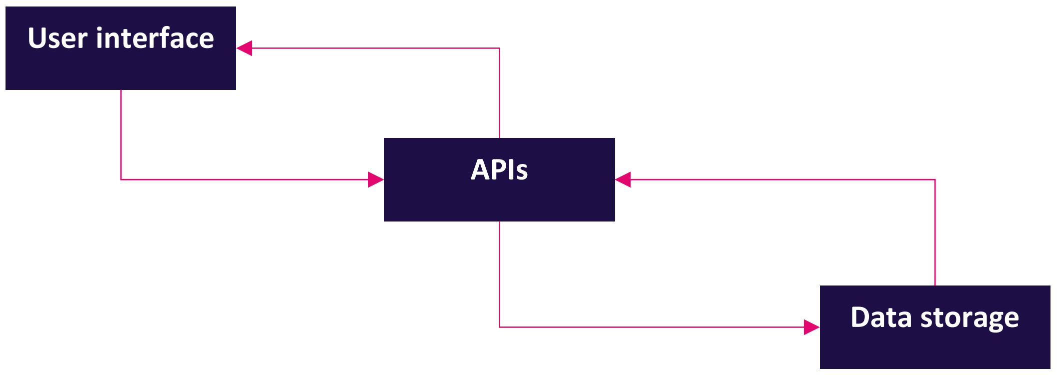

Figure 5. Building applications in a modern three-tier architecture using UI, APIs, and data storage.

7.1 User Interface

Often, the design of an application starts with the user interface. The idea and vision of what the application should do are also shaped by what it should look like. As a visible part, it is also easy to get input from various stakeholders.

In the past, it was customary to give the idea to a graphic designer who would create a clip or two of it based on a person's vision. The result was often sleek, polished-looking images that, with a bit of pixel tweaking and color changes, were easy for management to accept for production.

The problem with this approach is the almost total absence of user voice and the fact that more than a few images are needed to design complex and interactive applications. Nor is the visual interface a lifesaver if the rest of the user experience is poor.

Instead, a better way to model a future application is to use wireframing: Start by sketching and drawing rough patterns, boxes, etc., on paper or a notebook. Of course, you can draw on a computer, but it's almost always slower than traditional pen and paper. The key is to create the foundations for building more features: Where is the core content? How to lay out the navigation elements? Where to put the search box?

Another thing to think about on paper is the coarse level interactions: how to display the content, is there a separate page for search results, how does the layout change according to the screen size, and what are the main interaction components? It is easy to draw sketches, experiment, think aloud, and draw more. It's not worth focusing on the details as long as you can get the core idea of the program working on paper and even tested with suitable users.

With a good foundation, you can start iteratively drawing the user interface on the computer. Various design programs are available, which save time and make work more efficient. After four or five rounds of feedback, testing and comments on the design table, the result is often a pleasant and feasible design. Under no circumstances should the plan be too detailed and pixel-perfect, as it is rare that everything works the same way in practice. Even keeping the color scheme in black and white at this stage can keep the focus on the use of the program, not its appearance. Once the design also considers interactions, it is easier to build on it for implementation.

7.2 Data and Data Structures

Data processing, storage and structures are also a key design focus. Computer design is an art in itself, with many good practices regardless of the system used. As with user interfaces, it is a good idea to start by drawing the most important things in the application processes on paper, i.e., the entities. For example, do you need products, customers, orders, spare parts, vendors, etc.? What are their relationships, and what information do they use to relate to each other? This creates the basic data model for the application, i.e. the basis for the data and its processing. Equally, the data model must meet the objectives set for the software product to work as intended.

You can continue detailing the data model on the computer. In a relational database, critically examine the model. How is data processed in the different entities, how is the data identified, is the repetitiveness of the data decomposed by normalization, is there some form of data protection at the data warehouse level, etc.?

Data processing efficiency is usually reflected in the duration of the various queries; in principle, a short response time for an operation that produces the right results also means a good and optimized data structure. In a relational database, using different search indices also makes the queries more efficient. When designing operations, it is also good to consider the mass to be processed. Updating 1,000 database rows is often almost instantaneous. But what if there are 1,000,000 rows to be updated? With interdependencies with other entities and the application being used by 100 different users in the exact second? One should also consider scalability in the database.

Building a repository should also be iterative. Once the main entities are in place, the database often has much more functionality to model. It is possible to add data attributes progressively and modify and delete them as necessary. If the base structure is good, it will withstand various changes and additions over time. Poor solutions create bottlenecks and other challenges for data processing.

7.3 Application Programming Interfaces

The user interface and the data warehouse meet at the application programming interface (API). It provides a means for the two to communicate. API includes both services and a mutual data transfer format that the different layers of the application can then use. From the user interface perspective, the API hides all the detailed logic for retrieving, processing, and modifying data. And from the data warehouse perspective, the API hides all the visual niceties and details. A working application can use one or more APIs and sub-APIs, called microservices. In any case, the implementation and maintenance of an application are facilitated by a good, clear, simple, but extensible application programming interface.

The design of the API is based on the objectives and needs of the application on the one hand and the possibilities of the available information on the other. For example, many applications include a search function, which, as a user interface, means a search box and search results. The interface takes a search term entered by the user, searches the database for matching results and returns a hit list for presentation in the user interface layer.

7.4 Summary

The design phase in the software development life cycle emphasizes the need for proper planning and allocation of resources before moving to implementation. The technical design of a system involves designers, architects, developers, businesspeople, and users. The design phase also consists of making technology choices that impact the system's implementation, maintenance, and further development. The system architecture design involves information processing, scalability, automation, security, and implementation methods.

8 Development Processes and Methods

Application development is full of processes. There are development processes, design processes, go-live processes, maintenance processes, support processes, and so on. A process refers to the things and tasks that are performed to achieve a goal. It is characterized by its repeatability, making it more predictable to accomplish the objective. A process ensures that work proceeds systematically and by the objectives (Cambridge Dictionary, 2023).

The software development life cycle is also a process that systematically goes through different stages. But what distinguishes software production from, for example, an automated factory process? As software has been written for decades, why has it not been delegated to computers? Or if the application development process has been polished to the last detail, why are there still bugs or user dissatisfaction with the services they receive? Why was the old waterfall model approach to application development perceived as wrong? Why are agile processes any better? What might the future hold?

Every application development process is different. The critical variable is the human being who defines the goals and needs, creates designs and code, makes decisions, tests, comments, approves and pays for the application and the work involved. Human intervention always makes the process more challenging, less predictable and, in different ways, more risky. Programming is, first and foremost, problem-solving, and different things can be solved in different ways. Experience and skills influence the outcome.

Another factor is the constantly and rapidly changing technology. New solutions to old challenges are continually invented. Countless software, development frameworks, paradigms, production solutions, programming languages, data storage, operating environments and new versions of all existing technologies are released yearly. A software project that was well done five years ago may nowadays mean using an obsolete technology with no foreseeable life cycle in the next five years. Therefore, making a similar application requires new thinking and ways of doing things, as the old process might now work as such.

The third factor is the changing environment. A company can only function with cooperation, contacts, customers, competitors, and the need for change and process improvement that arises from internal activities. Good old processes and practices can be left behind as competitors improve their processes and operations, making their products better, cheaper, or more attractive to customers. If a company does not respond to environmental changes and evolve with the rest of the world, it will soon be a business of the past. Even big companies such as Nokia, Kodak and Blockbuster have been hit hard by the digital revolution, where they have not kept up despite their old innovations.

Failed IT development projects have been around as long as software has been around. Despite advanced technologies, skilled people, agile methodologies and business-aligned goals, new software is created that does not meet user goals, deliver the expected benefits, or add efficiency to existing work. Possible causes include project leadership, lack of necessary management support, and lack of project ownership and commitment. Many projects are under-resourced or have the wrong type of resources. Lack of communication and communication skills have also posed challenges in many projects. By investing in good planning, ensuring management support, allocating sufficient and the right resources and working towards a common goal, an application development project is already reasonably well placed for success.

8.1 Agile Methods

Today's development processes swear by agility. Agile development is a software development methodology based on flexibility, iteration, and close collaboration with the customer. The goal is to produce high-quality software quickly and efficiently to meet changing requirements (Agile Alliance, 2001.)

Agile development processes have sub-methods like Scrum, Kanban, or SAFe (Dhirak, 2023). What they all have in common is iterative and incremental development, working close to the customer, flexible change processes, short and fast development cycles, and continuous improvement. The key is finding a systematic development model that fits the organization's needs. The organization's size, level of IT competence, resources available, application, and objectives will all influence good and effective practices.

Iterative and incremental development means that the application is developed in cycles. Developers try to achieve something visible in each cycle and improve the previous work (See *Figure 6*). This makes it easier to react to possible changes and focus on the essentials instead of long development cycles that may take months to produce visible results. In the iterative implementation, the challenge is to decide what is essential and central at any given moment, especially in the next development cycle. Different roles, from product owner to scrum master, seek to address these issues.

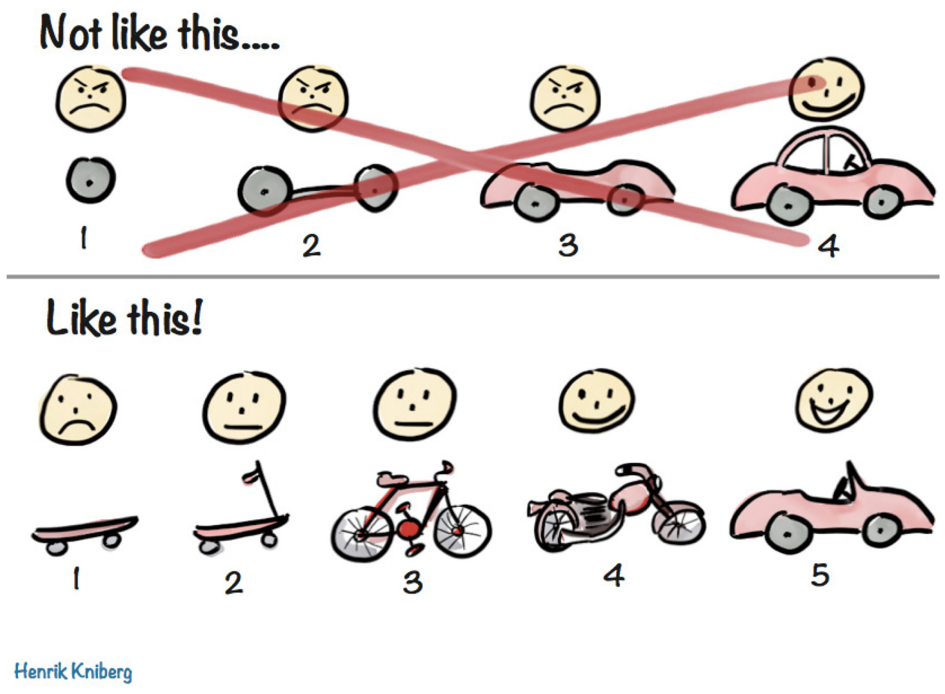

Figure 6. Agile development targets happy users in every development cycle (Kniberg H).

8.2 User-centered Methods

The problem with many applications is that they are not very usable, i.e. difficult, complicated, and unintuitive to use, contain bugs and are aesthetically unpleasing, if not ugly. User-centered processes, design and testing methods can reduce this risk of failure. They consider, design and implement things from the user's perspective, not the product's or the author's. It is a better way to create a service that meets the user's needs and objectives, to which the user is more committed and ready to pay (Novoseltseva, 2019).

Information about real users, their goals, and ways of doing things is required since program designers, developers, or testers are rarely the actual users: Who is using the application, for what purpose, in what situation, in what place and for how long? All of these have a bearing on the application's design, operation, and use.

A representative persona of the app user can be created, including name, age, gender, job, and hobbies. Design solutions are validated against this persona. Gathering feedback at different stages of the development process is an integral part of user-centered design. The earlier the developers receive information and feedback, the better they can focus the development of the application to meet the user's objectives, for example, by prioritizing the functions available in the application.

Iterative development and testing are also part of the user-centered process, which is why it is more easily integrated into agile development. Almost anything can be tested with users during the development process, even if it's just a paper wireframe model where different functionalities are demonstrated by swapping one piece of paper for another. More advanced prototypes can be tested on a computer, starting from a prototype to the finished product. When tests are carried out frequently, and they guide development priorities, the outcome is more in line with users' expectations and objectives.

8.3 Ticketing Systems

Agile development requires a controlled process, where the development team must understand the big picture of what they are doing and all the details that make up the whole. A sensible way to manage the process is to store and manage the various tasks in a specialized system such as Jira, Planner, Trello, etc.

In Jira, for example, large entities can be made into so-called epics, which are broken down into smaller stories. A story is a user-oriented description of a problem, need or function related to the use of an application. The stories are the pieces that the implementation teams should pick up and implement in a predefined time window, a so-called sprint. A good story describes what is wanted and defines the outcome. The story should be precise and implemented in a reasonable timeframe. If it is too broad, vague, or inherently laborious, the story must be broken down into smaller parts to make it more rational.

Ticketing systems are also helpful for maintenance. When problems, bugs, development suggestions, and change requests are stored in tickets, the whole system is better controlled, and the history of changes and their causes are stored.

8.4 Proof of Concept

Proof of Concept (PoC) is a common practice to demonstrate some idea in an environment that is incomplete or otherwise usable in a production environment. The primary purpose is to make minimal effort to create and evaluate an idea and determine whether it's feasible for further development. If the proof of concept works, it can be studied and developed further. In case of failure, the PoC can be discarded and abandoned. But even if it fails, the lesson is learned, and the money spent is justified. One can reform the idea later, and maybe another PoC is created.

The proof of concept is a very early version of something: A little bit more than just a theoretical idea, but no more than a simple prototype. The good thing is that it is not required to be fully working, can use mockup data, and the user interface does not need to be polished. The PoC can then be used to present the idea or the product vision and its benefits to stakeholders, like investors, which is common in startups looking for funding. Of course, it's also common for startups to fail to do a fully working product even after a PoC.

A good PoC is created with a good vision: what is the purpose of the idea, product, or application? Is there a need or demand for this kind of product? Who would use it? How will it be used? The vision points out the critical issues and functions that must be addressed. It makes the core of the PoC: are there practical ways to do them, or are they impossible to create in real life, in current technology, resources, or economic aspects? The definition of done should be defined before the process so it can be seen and said whether the PoC has succeeded or not.

Successful proof of concept is usually developed into a working prototype, demo and finally, a fully working product. However, as PoCs are usually done lightweight, using mockup data and other not-so-bullet-proof work, even commercial products may have bugs and errors dating back to the original PoC.

8.5 Using Artificial Intelligence

Artificial Intelligence (AI) has recently emerged as a hot trend, with OpenAI's ChatGPT and other generative AI services providing easy and simple user interfaces for non-technically oriented users (OpenAI, 2023). ChatGPT language models are driven by massive amounts of data and can naturally answer almost any question or issue.

The use of AI to support software development has also grown. In addition to ChatGPT, Microsoft's CoPilot and Amazon's CodeWhisperer, among others, help in many ways at different stages of application development. For example, ChatGPT can be used to help with general questions and learning new things, but also to answer more specific, code-generating questions. ChatGPT also has a good feature that explains the produced code line by line, making the code easier to understand and follow.

AI also helps with things like documenting code, making test cases, and even finding and fixing bugs. Above all, AI service can be like a friend who can be asked anything at any time, and with human-like conversational clarification, adding more information, and more questions, you can move towards a workable outcome. ChatGPT remembers and understands previous inputs and is, therefore, able to refine, correct and modify its prior answers.

AI that provides rich and varied answers can also blind its users. In the background, the learned machine can produce equally wrong, harmful, and biased answers. The results are no substitute for the skills or judgement of a competent human, and the code produced by AI should be critically evaluated in the same way that, for example, StackOverflow's human-written examples for different problems should be critically evaluated.

Even if the code-generating answers are not entirely wrong or non-functional, they can be unnecessarily complicated and complex solutions, even if the work can be done more efficiently and simply. For example, in many cases, the proposed solutions for database SQL queries have been based on specific features of some other database. It is also essential to know that all the questions and materials put to AI prompts can also be used to train the language model further. It is not advisable to copy and paste any proprietary code for AI to debug. In particular, the terms of use of the free language templates often reserve the right to use user-supplied data (OpenAI, 2023). In paid tools, user data is usually kept private.

Using AI to support software development is helpful, practical and makes sense. AI often speeds up and streamlines the work, providing forward-looking answers faster than searching and applying information using traditional search engines. Various functions and AI services will also become more prevalent in software developers' tools, making their use a natural support for programming.

8.6 Summary

The application development process involves various processes and methods, such as design, coding, maintenance, and support. Even with new technologies, changing environments, automation, and multiple configurations, the human factor plays a crucial role when developing software. Agile, iterative, and incremental design and development methods enable professionals to achieve visible results quickly, react to changes, and focus on essentials. Successful application development requires good planning, management support, allocating sufficient resources, and working towards a common goal. The rise of artificial intelligence will help developers in their tasks more and more.

9 Application Implementation

After a good analysis and thorough planning, it's time for implementation. The application is created, coded, and developed in this software development life cycle phase. The implementation phase is crucial for communicating between the architect and developer so that results are robust, consistent, and maintainable. The development phase is usually the longest, but it's essential to deliver results of even unfinished parts of the application for testing, evaluation, and decision-making.

9.1 Development and Production Environments

Developing applications takes time, experimentation, new ideas, maintenance, and testing. The software also requires data to be added, changed, and deleted. Once the application is in actual use, making changes, additions, and corrections is much more challenging. It is impossible to change an application in production use, as this can cause interruptions, errors, and incorrect information when the application is in use. For some systems, it is virtually impossible to perform comprehensive testing without an environment that does not change the state of the production service. For example, e-commerce products use purchases and customer data. Testing by making actual purchases involving monetary transactions becomes costly.

So-called cowboy coding is a synonym for making application changes in production and testing with accurate data, which is not recommended in a controlled application development process. Instead, it makes sense from the outset to consider different application development environments in addition to the production environment.

An application development environment is a production-like environment where an application can be developed in a controlled and secure manner without any real-world impact. It contains the systems needed to run the program and the tools and facilities to build the software. A local development environment is a set of applications, techniques, and procedures for programming work on the developer's machine.

A local development environment is the minimum requirement for a managed process. Still, often, the application developer's computer does not match the production server in terms of operating system, performance, network traffic, or data. As the aim is to use and test the application in an environment that is as close as possible to the production, because the closer the test and production environments are to each other, the more reliable the reproducible behavior of the application can be considered, for example in the event of errors.

A dedicated test environment is a prerequisite for creating a high-quality application so that new versions of the application can be tested and put into production in a controlled way. The test environment is used to run extensive automatized test protocols and to test the functionality of things and processes manually. If the system is extensive, in addition to testing, a staging environment, which is already almost identical to the production environment, can be set up before production to ensure that the application goes into production smoothly and that the latest changes work.

9.2 Programming Languages and Frameworks

A programming language is needed to produce software, and today, many different languages exist. *Figure 7* lists most used programming languages among developers globally. The programming language you choose to create software is, first and foremost, a strategic choice that has far-reaching implications for the future and the skills and capabilities needed in the organization. Traditional and long-established languages such as Java, Python, JavaScript, PHP, and C# often have more support and talent, while the newer languages like Dart, Kotlin or Go offer innovative and efficient solutions for application development but with far fewer skills and solutions (GitHub Innovation Graph, 2023).

Ensuring skills and continuity should be considered when choosing a programming language. The language used in the existing code is also a strong candidate for new code if it is expected to be carried into the future. If the old code and programming language are perceived as cumbersome, it is worth assessing which new model fits in with the existing knowledge and skills of the company as a whole.

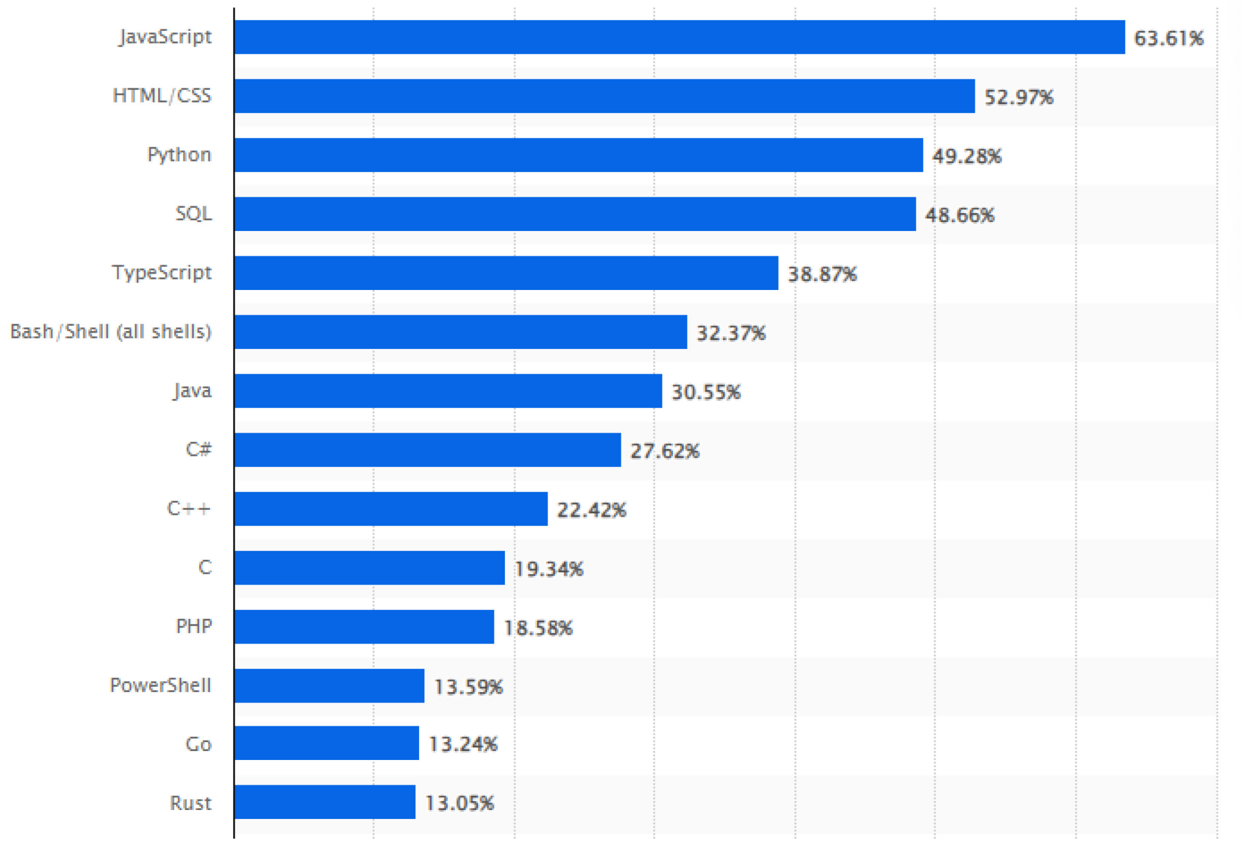

Figure 7. Some of the most used programming languages among developers worldwide (StackOverflow, 2023).

JavaScript is necessary for web development, as it is the native language supported by browsers, to which many other languages eventually compile their code. With browsers, JavaScript has evolved to work seamlessly with the HTML markup language and CSS style sheets. With Node.js, JavaScript has become a popular language in backends and other applications outside the browser. Programming becomes more efficient if the same language can be used in all layers of the application.

Choosing a programming language is still not enough. For many languages, different frameworks provide foundations, application interfaces and development models for programming. Although the language of the frameworks is the same, the way of doing things can be very different, and it is essential to adapt to the way the framework does and implements the program. Examples of popular JavaScript-based frameworks for building web applications focusing on user interfaces include Angular, React and Vue, which have different development philosophies. Express, Koa and Meteor, which have twists and turns for creating APIs, run on backend Node.js.

The choice of framework is equally strategic, but it is also a good idea to consider the strengths and weaknesses of different frameworks for building different types of applications. For example, data-intensive applications that use a variety of interface components familiar to desktop applications will benefit from a complete library of gestures built into a single framework. In another application, flexible building blocks are needed for different screen sizes on mobile, tablet and computer screens, for example, which may lead to the choice of another framework.

Often, applications are built using many different libraries, making it easier to have various functionalities ready to go rather than implementing them yourself, reducing the need to maintain your code. On the downside, the application can become a patchwork, with different parts of the application updating and changing at different rates or even conflicting with each other in the worst case.

As with any programming language, one needs to consider continuity, especially as popular frameworks evolve rapidly, with new versions appearing several times a year. Taking advantage of new features can help with an application’s functionality, but updates can be about security and fixing vulnerabilities. Therefore, keeping application components up to date is essential, albeit challenging.

9.3 Developer's Tools

A full-stack application developer's toolkit must include several different tools for making a program. The tools used depend on the technologies used, but the main applications include IDEs, version control, automation utilities, project management tools, collaboration, and communication tools.

IDE, an integrated development editor, is the primary tool for code generation. Even though it is possible to use a simple text editor to write programs, modern IDEs offer many programming aids, including readability-enhancing code coloring and automatic formatting, automatic completion of program fragments, integrated documentation, reference management, refactoring tools, testing, debugging and correction, code analysis, variable and internal state checking, compilation into a finished program, and program execution. Many IDEs vary according to the programming language and framework used, while some tools are tailored to a specific purpose. For example, Visual Studio Code, IntelliJ IDEA, Webstorm and Eclipse are good and sophisticated code editors.

Version control is a vital tool for managing code and for simultaneous development by multiple programmers. Even if you write code alone, version control software such as Git is a critical and quality-enhancing utility. Version control includes not only the different versions of the code over time but also allows the program to be written in parts and with different features in mind. However, simplicity is essential, as it is easy to get tangled up in various branches and tags of a version control application. When the version control is linked to the different application development environments, i.e. the code that goes to the test branch is published in the testing environment, production has its branch, etc., the codebase remains more manageable.

Combining various automation functions from testing to release to migration is a good idea. Agile development involves continuous and rapid release, which means it should be as automated a process as possible, following the DevOps continuous integration and deployment model. Automated testing and release allow faster and easier controlled changes to be made to the software, improving the quality of the code. Good continuous integration and continuous delivery (CI/CD) tools include Jenkins, CircleCI and Gitlab functions.

Managing an application development project is challenging, focusing on the activities to be performed. Using epics and stories in tools like Jira and Trello in the design phase makes it easier to keep track of the whole project, serve the necessary activities, manage time, and prioritize the work. A versatile project management tool provides functionality for the day-to-day work of the application developer, the project manager, and the product owner.

Application development is often done in teams, where the interaction and communication between groups is crucial, e.g., in producing features, testing, and managing work. However, in the modern age, where developers can be located worldwide, there is a demand for near real-time communication. Email is not a good choice for this purpose, but tools such as Microsoft Teams, Slack, Google Chat and Discord offer chat, voice or video conferencing, and file-sharing functionality.

Other valuable tools include database management software, testing software, utilities for checking code quality, formatting and security, containerization tools and various logging and monitoring tools.

9.4 Zen and the Art of Programming

Programming can be called an art, a product of self-expression. At the same time, it is precisely defined within the syntax and structures of the programming language. Still, on the other hand, there are many ways to achieve the same result, and the solution to the means is the programmer's production.

However, since the code is used to make a functional product or service, the quality of the code matters. Reliability, maintainability, portability, reusability, and testability are critical characteristics of quality code. Many developer tools help in writing good quality code and should be used.

Of course, the most important thing for code is that it does what is specified and intended with certainty. For example, in the case of a calculation, it should always give the right results with different parameters. A good program does what it is supposed to; an excellent program also considers error situations when inputs or data are incorrect.

Although everyone has a hand in writing code, it should be clear and understandable, affecting clarity and maintainability. Complex and uncommented solutions, using short and obscure variables, unstructured indentation, hard coding, confusing naming conventions, unresponsiveness to errors, and over-optimization in some situations make code difficult to maintain. Every line of code is a potential bug spot, and the more you can do with less code, the better – as long as the handwriting is clear and understandable to the rest of us.

Good code can be reused and, if necessary, ported to a program running on another operating system, for example. Using various libraries, your own or those of others will make your work more efficient and faster if they are of good quality. For example, the number of downloads of Node NPM packages and the number of stars and development activity on the GitHub repository usually indicate the usability of third-party code.

For a program to do what it’s supposed to, it must also be tested. If the code is too complex, depends on many things or requires a lot of data, even real-time data, it makes it challenging to try, test and thus find errors. Finding problems as early as possible, on the other hand, affects the cost and the quality of the product. Testability, whether automated or manual, is an important consideration in programming and code quality.

Other programming pitfalls include security, efficiency, consistency, repeatability, lack of documentation and interoperability of components. You learn programming by doing, and usually, the next project will be better because of learning, new skills, and development.

9.5 Using Data

Applications need data to work, and the possibilities for using it must be clear at the latest when the development of the application begins. Often, data already exists that can be made available to the application. However, only some applications can cope entirely with existing data, which is why new data structures are also needed. For example, a relational database must create the necessary tables, fields, views, and indexes and define the relationships between the data.

When the data and its management are well-designed and implemented, the data structures and the application's functional logic support each other. This is particularly evident as the use of the application grows, so does its data, bringing extensions to existing storage and processing. Poorly implemented data structures cause problems and additional functional logic in the application. For example, if only a person's full name is stored in a database, but later the given name and surname are needed, this can create challenges for data processing.

Data can be stored in structured or unstructured form, usually involving a choice of data storage technology and data manipulation languages used. Structured data storage is good if the data structures can be identified and created in advance. Unstructured data can be composed of several different formats just when data is entered into a data warehouse.

If data needs to be moved or merged between systems in integrations, the most straightforward way is to have a one-to-one mapping between data. This is often possible for basic data, names, IDs, addresses and phone numbers. There is always a risk of data distortion, unexpected changes, and even data replication when converting, merging, splitting, and using many data combinations. Master data, i.e. the data to be trusted, especially in conflict situations, must be carefully defined.

Care must be taken when handling data, as changes affect the system's state. Data quality, accuracy and integrity should be monitored and checked by various means. Simple checks are, for example, type-checking of inputs: whether the data are numeric, dates, text, etc. A database can validate the data in more complex cases, for example, for uniqueness. When using interfaces, different inputs should also be checked in the backend for security and data integrity reasons, even if the user interface does some basic checks.

For data-intensive applications, i.e. those that handle a lot of form data, such as CRM, HR, and other management applications, you should consider automatic data saving for a better user experience. It works better than forms with a save button, making the application more efficient, allowing immediate feedback, and eliminating errors caused by forgetting to save.

Data retrieval and processing often need to be improved in systems because there is simply a lot of data, as business operations can quickly generate a significant amount of data, even daily. Another possibility is that the searches are inefficient. It can be due to poorly done data queries, lack of database search indexes or programming problems, typical of which are database searches within different loops. Identifying and solving bottlenecks is part of the programmer's job to control response times, system load, and reliability.

9.6 Security and Data Protection

Application development always involves information security and data protection, which must be considered at all stages of development, from design to implementation and from release to maintenance. It makes the application stable, secure, resilient, and protected against various threats, attacks, and malware.

Security is not a feature or function of an application that can be created or installed afterwards. It is possible to improve the security of an application, but if the right infrastructure is not in place, it is challenging to add protections and other security on top of it. It is also not just a technological issue; security issues are related to the whole life cycle of the product, laws and regulations, product ownership, use and training, among others.

Security and privacy work starts at the design stage, where essential security requirements are defined depending on, for example, the criticality, sensitivity and purpose of the application and the data it uses. For instance, GDPR describes certain aspects of user data protection. Data security threats and protection issues must be identified, classified, and linked to the relevant application functionalities. For example, user identification and access rights processing are integral to the whole application.

The security, control, verification and integrity of the application and its data are part of a good application development process, regardless of the technology, platform or data repository used to build the application. Each solution has its own set of security challenges that the application developer needs to be aware of and manage. Still, many different general guidelines, documentation, standards, best practices, and broad knowledge of threats and vulnerabilities related to information security and privacy help make applications more secure, even if the developer is not a security expert.

The non-profit OWASP organization's top-10 list is one of the best documentation defining security risks for web applications (OWASP, 2021). The list, updated annually, includes timely topics, but year after year, the list contains risks from various out-of-application injections when user input is not reviewed and cleaned up. User authentication, permission handling and session management are often challenging, and gaps in these can lead to serious security problems. Improper settings in any application stack are a risk and an attack vector for an intruder, whether firewall rules, default passwords or open maintenance pages. Since applications are often built using components made by others, ignoring these updates can also create security vulnerabilities. Application services, from the operating system's core to higher-level functions, are subject to frequent updates that should be brought into production in a controlled manner through testing. A good application also logs sufficient usage data at a level that allows anomalies to be monitored and, if necessary, resolved after the event.

When building an application, it must be understood that each line of code is a potential bug spot and a potential security hole. From a security perspective, code clarity, commenting, modularization, error handling, and consistency are also important. At the very least, a code review will be required, but sometimes it is recommended to do a security audit of larger systems, where professionals look for weak points in the software.

9.7 User Management

Identity and access management, simply user management, is the cornerstone of many applications. It contains the functions, processes, workflows, information about the program's users, access control, access rights, and data management. User management is also the leading security risk for many applications if a hostile party with excessive access rights gain access to the application. It is worth taking it seriously and, from an application perspective, ensuring that user data is checked and validated in all operations.

User credentials are also an essential and valuable asset, as they are usually backed by the person who is using the application. Different types of user information can help the application adapt to different needs and situations. For example, in a patient information system, information about the user's professional background is even critical for the functioning of the application, since only doctors have the right to prescribe medicine.

User management involves several different entities. Today, it is not necessarily worth building a complex and multi-level user management system yourself, but to use, for example, Microsoft's Active Directory or similar comprehensive user directory services. A dedicated application is integrated, e.g. for user authentication and access control, thus providing ready-to-use and tested services with less effort than doing it yourself. Even if you use off-the-shelf systems for user management, knowing the different components and most critical functions is essential.

User accounts contain all the information about users needed to run the application, such as name, identity, and access rights. User identification ensures that the users are who they claim to be. There are different options for user authentication, among which it is advisable to implement secure multi-factor authentication methods. The authenticated user's authorization, or access rights, determine what the user can see, do, and act on the service. The user management system gives an authenticated user a session that is only valid for a certain period and protected from other users. In terms of user interfaces, users should be offered a range of self-service options, from registration to log-in, log-out, resetting a forgotten password and managing their data.

9.8 Testing: It Compiles!

Testing systematically examines and evaluates software against its requirements and functions. Testing can be carried out on a single process or procedure or the operation of the software as a whole. The aim is to find and correct potential errors and problems before they reach production. In this way, the application works more reliably, more stable, more securely, and better in general, i.e., testing impacts the application's quality.

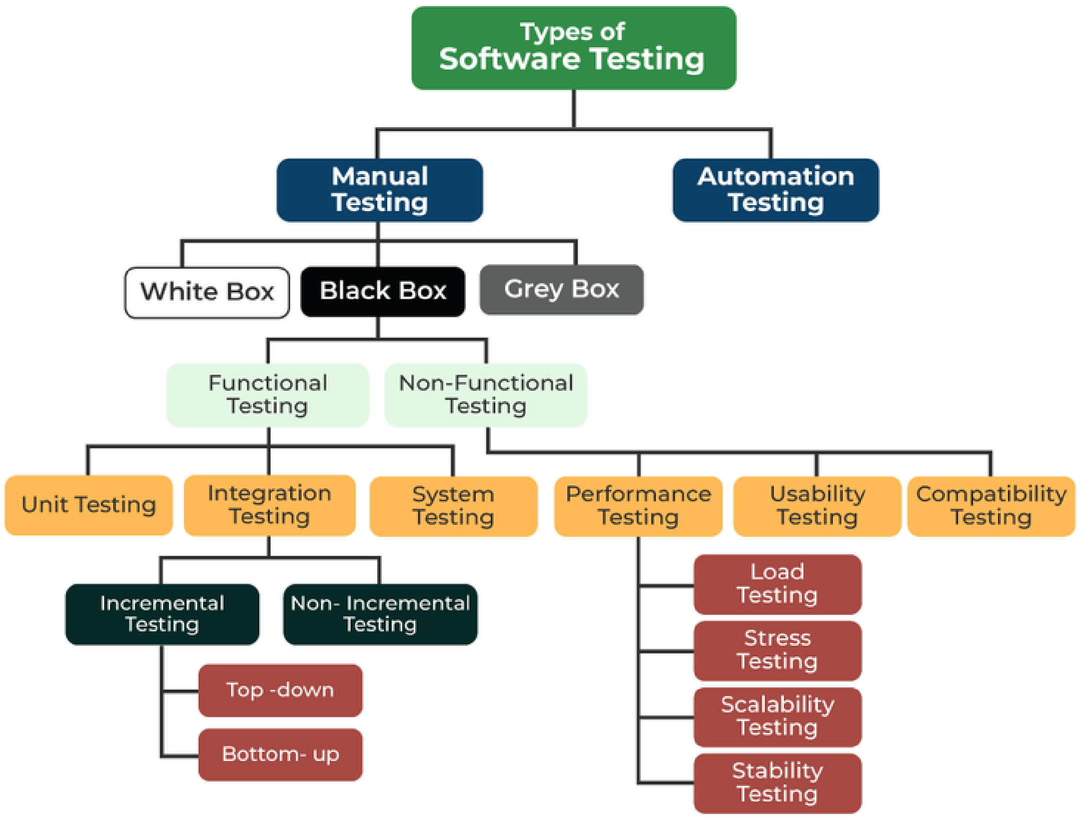

Figure 8. Various testing methods (Geeks For Geeks, 2017).

Testing is the next step after development in the software development life cycle. Despite the benefits of testing, software testing is a somewhat underestimated part of the development process. There are various testing methods, all focusing on different aspects of the application (See *Figure 8*).

From the programmer's point of view, the operational logic often goes through a positive 'happy flow', where the data and its processing are precise and error-free. It can create the illusion that the application always works as it should. Some problems are rooted in the definition and design phases, which is why testing should be done from the beginning and at all stages of the application development life cycle. For example, interface designs should be tested as early as possible to detect usability problems before a single line of code has been written.

The most challenging part of testing is determining what and how things should be tested. Functional testing is the most common way to test an application. Its purpose is to validate that the program functionality is as it should be in different modes, i.e. that it produces the right results, displays the desired data, saves changes to the data, and is smooth to use in the various user interfaces of the program.

Even if the program logic works as expected on the test bench, it is reasonable to consider non-functional properties. These things, such as performance, security, and usability, affect how the program works. A program running well on a developer's computer can crash when it has 50 concurrent users. The network openings required to make the program work are a threat to security. A complex and opaque user interface frustrates and creates friction in the user's tasks.

A dedicated testing environment is a good and safe place to go through the program's functionality without touching and changing the state of the production servers. However, the results are only valid if the test environment, the technical infrastructure, and the software's data are sufficiently similar to the production environment. Appropriate data is sometimes a challenge when testing a program. The correct data contains the right information in the right relationships and context.

On the other hand, using production data is a privacy risk. The correct data may only become available as the application is used, and errors may be rare even then. The recommended use of generated data is for testing system performance.

Continuous integration and delivery (CI/CD) processes aim to improve application development by improving the efficiency and speed of product-to-product transfer. From a testing point of view, this is a good thing because a smooth production transfer process only works when automated, which means that testing should also be automated. Developers have to think not only about the functional logic of the code but also about testing it, which usually positively affects the clarity of the program code. The test-driven development paradigm furthers this: tests are written before the code.

Unit tests are used to ensure the functionality of the different parts of the software. They are used to test small components, functions, unexpected inputs, and return values in an automated way. They ensure correct functionality at the detail level.

End-to-end testing runs through the application, as users run the program through a particular process, for example. Along the way, components, integrations, and other parts of the application are tested. It ensures the application works as expected and meets the users' goals and needs.

There are many different tools for automated testing, ranging from ensuring the functionality of interfaces to systems simulating massive loads. Testing systems should be selected according to their purpose and systematically deployed in application development and across projects.

Not all things can be tested automatically; manual testing is also needed. It depends on testing goals and how one wants to test the program. Usability testing involves monitoring the performance of a real user on various tasks in the program. If the user stalls, misunderstandings, or errors occur; these are usually usability problems. The usability of a program can also be assessed by an expert who is familiar with the program and knows several laws and good practices. It is a good idea for application owners to test and evaluate the usability of their product before it is released to the general public. Large companies may even have testing or quality assurance teams specialized in testing systems.

There is no point in testing if the results are not used. In the CD process, if testing is automated, the release can even be aborted in case of a fatal error. These are easy to react to and make corrections.

Using the results of manual testing is more challenging. Therefore, testing needs to be well planned, tested according to the plan and classified according to the criticality of the findings, e.g. for both users and the business. Critical problems are a barrier to using software or producing incorrect results, so they should be fixed immediately. Medium-level problems do not prevent use but make it more difficult. Low-level problems create friction in use or are otherwise cosmetic, albeit actual errors. By classifying findings, it is also possible to prioritize their correction.

Testing aims to verify that the software works as it should. In addition to detecting errors, it is essential to record which things work so that they do not need to be addressed, changed, or fixed from the outset. Especially if the code behind the program is high quality, it can live long without significant maintenance.

9.9 Summary

Implementation is the most time-consuming phase, where architects and developers work together to deliver robust and maintainable results. Different application development environments alongside the production environment facilitate testing and evaluation of the application. Different frameworks provide foundations, application interfaces, and development models for programming. Choosing a programming language and framework is a strategic choice with far-reaching implications. Therefore, continuity and skills should be considered when making these choices.

10 Deployment, Production, and Maintenance

10.1 Time to Go Live

After hard work of planning, designing, developing, and testing, it is time to go live. It usually causes some shaky fingers and sweat on the developers' foreheads. The release of a product is always challenging, especially if the old version is in active use and there are many changes to the new version. Or if the product is brand new and marketing wants to launch it at a big and important event. So, keep calm and carry on. Preparing carefully for migrating an application to production usage is a good idea. However, the certainty of its success is built in the earlier stages of the application development process – design, implementation, and testing.

Agile development and CI/CD automation enable rapid response and controlled production transition instead of complex, massive, and error-prone manual work. Usually, it is better to release minor updates to an application frequently rather than extensive updates infrequently. This way, the production migration is not a significant stress but part of a continuous improvement process. Suppose the release of an application requires some outage, user shutdown, or other visible operation. In that case, the production push can be scheduled when it is least disruptive to users, usually during the night hours.

Despite all the testing, few applications are flawless in the real world. When a program is modular so that its different parts can be updated independently, changes are easier to make, and errors are easier to trace and remove. Usually, the first question to ask in problem situations is: What has changed since the last time the program worked without errors? A controlled release process and the preservation of version history make it easier to correct mistakes that make it to release. A good production migration plan also includes how to revert to a previous, working version if necessary, and what is the timeframe after which there is no reversion to the old version?

Communicating changes and upgrades to the various parties will ensure everyone knows them. For example, customer service will be better able to respond to requests for support when they see what changes have been made recently. If the application includes new functions, processes, and other features, the instructions and user support should also be included in the release. Sales and marketing will also benefit from status updates.

10.2 Usage Monitoring

The application in production must be monitored and controlled. Are the organizational resources allocated sufficient? Is the application running smoothly? Is the application generating unexpected errors? Is the processed data stored correctly and with integrity? How many requests for support are there? Many things affect production usage, so various means must be used to monitor the health of the software.

Logging is one of the most useful monitoring tools. A program can record information about its activities in a time-stamped log, which records error conditions and key successful events. For example, logging what users do, like what search terms they use, what time of day they use the application, and when errors occur, provides information about the application's performance and user usage and activity. For example, log data can be used to determine which application features are used and which are not. Unused functions are either not needed or remain invisible to the users.

A good logbook entry tells you who did what and when. Especially in case of errors, all possible information helps solve the problem. For example, finding out the user's input can reveal why the error occurred. However, only the relevant information should be recorded in the log of regular events; otherwise, the log volume can grow out of control. Log events can also often be prioritized, and the level at which they are logged can be centrally controlled. For example, when the application is well-established, the usage and possible error conditions are known, and excessive logging can be detrimental to the detection of real problems.

When a log is generated, it must also be possible to analyze it. Different types of events must be distinguished and classified. The causes of recurring errors should be identified and corrected. Logs can also be used to generate usage statistics, which can be summarized to give a good quantitative picture of the use of the application.

Ad-hoc metrics such as the use of execution resources by the program, such as processing time, memory, disk space, and database usage, are also essential to monitor, with sudden spikes usually indicating symptoms and problems in the program's operation.

10.3 Documentation

If software testing is sometimes undervalued, documentation is even less considered in most application development processes. Documentation refers to a range of materials describing the software's operation, use, processes, and procedures so that users, developers, administrators, support services, etc., can understand how to use, modify, and maintain the software (Lutkevich, 2023).

Any documentation is better than no documentation. Therefore, the threshold for describing things should be kept low. Documentation can be done in many ways, such as written documents, code comments, pictures, diagrams, help files, etc. Documentation should be done during the whole application development process, making it easier to write things down as you go rather than afterward. Ideally, documentation is directly linked to the process, e.g., the use of version control and good change comments are documentation of what has been done. It is rare to be able to describe every issue with complete accuracy, but if the documented information can be applied, that is better than nothing.

Good documentation is reliable, up-to-date, and accurate. Definitions, instructions, and descriptions must be trusted to tell the story correctly, with sufficient scope and accuracy, and to have been kept up to date with changes. Maintaining documentation should also be part of the application development process. Accuracy in texts, instructions, illustrations, and diagrams helps to understand and clarify matters, for example, in further development, in the case of errors and the transfer of information from one author to another.

Finding the right level of documentation poses challenges. While a detailed description can help understand the program's logic, keeping up with the software is not easy. Therefore, keeping the documentation more general, where things are presented as more significant and interrelated entities, is often easier to approach program architecture. Detailed descriptions are appropriate for integrations where information is handled according to a precise process. But, in user interface documentation, presenting each field and component is often unnecessary.

10.4 Maintenance Mode

For an application to work well and safely, the whole environment in which it runs and is used must be kept in good condition. According to the SDLC, once the application goes into production, the maintenance phase begins, the purpose of which is to look after the well-being of the application, i.e., to update the infrastructure related to production, from the networks to the operating systems, from the components used by the application to the database patches.

Even when the application logic is functional and tested, many other system parts often undergo various updates as security, bug fixes, and new features are added. For example, dozens of major versions of node.js have been released in a few years, not to mention countless intermediate versions. The decision to not update can be a risk and a threat to the functionality of an application. Long intervals between updates are also challenging if the changes are so significant that your application no longer works, e.g., on newer operating systems.

In the cyclical model of agile development, the boundary between maintenance and new development becomes blurred, and the key is to ensure the continuity of the application life cycle. Most commonly, maintaining an application means fixing bugs, adding new features, and refactoring the code to make it more efficient, precise, and better. Feedback from users, new ideas, security holes, bugs in the logs, growth in usage, and performance bottlenecks are among the catalysts for new software versions.

10.5 Product End of Life

Many applications reach the end of their life cycle when they are no longer maintained, new technology overtakes the old, security is found to be inadequate, or users switch to other software.

Few software developers are interested in supporting the technology of a program that is years or decades old. Even companies trapped by outdated technology likely want to modernize their systems. Technology is changing so fast that things can and should be done more easily, quickly, and efficiently with new technology. The security of many legacy applications can be non-existent, which in today's networked infrastructure poses a significant risk to business continuity and also to business. And even if the authors think the old software is good and works, users may be attracted to another application and move on.

Whatever the reason, the end of life of an application should be handled well. Especially if it has paying customers and intends to get them to continue with a newer application. Communication plays a vital role in the demise of an application. Informing users, including the reasons for the shutdown, options for the future, and the possibilities for recovering existing data or moving to a new platform, should be described. Users should also be given sufficient time and support where possible. Clear deadlines for the termination of the service are part of this.

By carefully managing the end of the software's life cycle, users and customers retain confidence in the manufacturer. They are better able to migrate to newer software from the same vendor.

10.6 Summary

Going live with a new application can be challenging, but careful planning, agile development, and automated CI/CD processes can help. It's essential to monitor the application's usage in production and address any errors that arise quickly. Documentation is critical for different parties to understand how to use, modify, and maintain the software. The maintenance phase ensures its well-being and keeps it up-to-date with the latest updates and features. Neglecting updates can pose a risk to the functionality of the application. As many applications eventually reach their end of life when they are no longer maintained or when new technology overtakes them, proper communication and support are crucial when handling the end of the application's life cycle.

11 Discussion

In today's world, computers have become integral to various aspects of our lives. Their adaptability and potential to enhance, improve, develop, and entertain people's lives continue to grow. As hardware becomes more potent, software follows suit, gaining versatility and intelligence. Applications, algorithms, and data play a substantial role in our daily routines, turning software into a valuable digital asset. It's worth noting that, despite automation and artificial intelligence, humans are the architects behind every piece of software, even if assembled from off-the-shelf components.

The role of an architect involves understanding large entities, design methodologies, solution models, data storage options, and testing methods: all the things needed to build applications systematically and comprehensively. The work field of architects is varied, and different companies may have several specialized architects. An application developer produces code according to defined plans, designs, and architecture. The dialogue between architects and application developers must be practical and continuous.

This thesis aimed to look at software production from a practical systems architect's perspective, supporting the work of application developers. The aim was to answer the questions of what an effective application development process is, how architectures help application development, and what issues application developers need to consider in their implementation work.

As a result, efficient and effective software development is based on a systematic software development life cycle model divided into different phases. The production and use of an application are always based on the solutions and choices chosen, but only a well-thought-out and defined system architecture can be called a true architecture that supports and facilitates application development. Then, an exemplary architecture provides a suitable framework for application development. Despite the limitations of architectures, the application developer has a myriad of choices and possibilities for making the program. It also means challenges and solutions to many more or less complex issues.

11.1 The Software Development Life Cycle Process

The thesis focused on the software development life cycle model (SDLC). The aim was to determine an effective software development life cycle process. The SDLC provides a systematic and structured development model from idea to finished application. Dividing the software development process into different phases helps to achieve the final target because it allows the work to be focused on the right things at the right time. For example, programming an application with good design plans is easier and more efficient. A systematic life cycle model helps keep the overall software project under control, reducing the risks associated with production in terms of costs, resources, schedules, delivery, and, of course, the application itself. If applied in an agile and iterative way, the model will help the process through continuous feedback and improvement of the development.

According to the SDLC, the application development process is divided into definition, design, implementation, testing, production, and maintenance. Each company implements these steps to the best of its ability and capability. An application development process based on the life cycle model is as efficient and effective as the people who implement it want it to be. Therefore, it is more important to go deeper into the steps of the process and consider the possibilities of implementing them. User-centered design, also an iterative model, is one of the most valuable processes because it focuses on the user, their needs, and goals. Involving a cross-section of people in the initial brainstorming and design phase means that issues are considered in many ways, not, for example, just technology or sales. On the other hand, as the project progresses, it is good to use and rely on the talents of those who can deliver both design and implementation solutions.

11.2 Technical Architectures Supporting Application Development

Thoughtful and supportive technical architecture is crucial to the implementation of any service, as it lays the long-term foundations for the creation of applications. A good architecture supports applications from development to production and maintenance. The architecture must be just the right level of rigor and precision to define the framework and boundaries for application development but also flexible enough to allow the application to be creative and engaging. The good news is that this is what technologies are generally like.

Technical architecture must evolve with the rest of the technology. More and more applications are being made primarily for browser-based use. Browsers bring ease to installation, usage, maintenance, and updating. Browser technologies are practical in many ways, mainly as different JavaScript frameworks boost and enhance development and rich user experience. For some applications, such as games, mobile frameworks provide good services for building the application. The days of pure desktop applications are ongoing, too, when you are looking for the most efficient performance or access to resources that the browser cannot provide.

An architecture based on modules and components that can be used, modified, and updated independently also makes it easier to replace these parts when necessary. Transferring data through well-defined application interfaces helps to build services that can be extended in a controlled way. Good and documented application programming interfaces can be used to implement a variety of services beyond those for which they were originally intended.

A well-thought-out architecture also helps applications to scale. Increases in resources such as memory, processing power, disk space, and network traffic capacity should grow flexibly. Many things in an application are potential bottlenecks that can be tackled by anticipating them with good design and a correctly sized production environment.

Data security is one of the most essential things an architecture can offer if appropriately addressed and invested in. Protecting the application, its interfaces, database data, etc., from unauthorized access is a fundamental part of any software, but unfortunately, not all developers are familiar with secure application development. When the architecture provides ready-made or applicable models for security-aware development, they are better considered than various application-specific solutions. For example, systematically securing interfaces behind access session verification will reduce direct attempts to APIs.

One of the main contributions of architecture to application development is clarity and simplicity. The architecture provides a common platform that makes it easier to build applications and steers them in the right direction. Joint agreements on used development tools, programming languages, databases, and frameworks reduce the need to rethink the basics for each application. For example, if different user management activities were built separately in each application instead of a centralized service, the development time would be taken from the implementation of the core functions of the application itself, the user experience with different logins would be poor, and the security of the software would be questionable.

11.3 Key Considerations for Application Developers

The challenging job of a software developer is to implement software according to specifications and plans. Modern software, with all its functionalities, is usually a complex entity to build and maintain. Nowadays, software packages are created mainly by integrating different components. The application developer must manage several aspects of the development and maintenance processes.

Understanding and knowing the whole software life cycle is the key foundation for the rest of the knowledge. Phasing provides structure and planning in a field where going from an idea to something without a plan is straightforward. Especially from the perspective of a full-stack developer, understanding the big picture is the path to enlightenment. Even for a developer specializing in a particular area, it is essential to understand the whole production process to make the best possible decisions.

Mastering different methodologies is the next level. For example, agile development approaches, DevOps, version control concepts and functionality, data warehouse management, debugging, and testing practices are all part of the application developer's method stack. Communication, teamwork, and a willingness to learn are also good attributes of a successful developer.

Once one masters understanding the big picture and the various methods, the most crucial thing in terms of productivity is the programming work itself. How well can one produce solutions and functionality for the program you are implementing, whether it is a user interface form, a multi-source API, or the most efficient database query possible? Is the resulting code of high quality functional, clear, understandable, modular, documented, and therefore maintainable, testable, and, if necessary, extensible? Often, less is more, but for clarity, more is less may also be better for maintenance.

11.4 The Making of the Thesis

This thesis used an autoethnographic and diary-like model, focusing weekly on various application development topics. Their contents were based on the application development life cycle model (SDLC) and the author's decades of experience in software development.

The implementation method was well suited to the thesis, as it fitted in well with the author's day job and thus progressed smoothly. The daily work of a system architect includes reflection on application development processes and architectures, which was carried out during the thesis and also provided a structure and a timeframe. The work was more manageable to delimit without needing a worldview description.

In many diary-style works, the author describes the days' work chronologically. These may reveal the author's job description and tasks, but the content is often bland and secondary for the reader. Instead of a "Today I read and sent an email, then I coded" type of structure, the Notebook of a System Architect wanted to describe what and how a system architect thinks in his work rather than precisely describe how the job is done.

On the other hand, a diary's personal touch was wanted to be retained. Application development is solution-oriented brain work, and despite its systematic nature, it is not exact or fully definable in advance. Information technology's versatility gives many possibilities to implement software in different ways. It is also reflected in the implementation of this thesis.

A good example is the use of artificial intelligence, which has been at the forefront of the technology hype for about a year. Generative AI's language models produce natural and fluent text on almost any topic, although sometimes, they spew out completely wrong content without actual facts. The user holds responsibility for using the content for different purposes.

This work has made use of AI in three main ways:

- Ideation and idea expansion: in this respect, ChatGPT is like a colleague with whom you can discuss different topics.

- A translation tool: most of the work was written in Finnish first, so there was no need to get bogged down in linguistic issues when writing the raw text. The translations on DeepL have produced a reasonably smooth translation text, which has saved a lot of time.
- Language maintenance: correcting language, marking various punctuation errors, and assessing and improving the quality of the text have all been helped by both Microsoft Word's tool and Grammarly.

Artificial intelligence in its various forms is, therefore, a tool in the same way as other utilities. If they make work more efficient and improve the quality of work, they have a place in the future.

The thesis is half of the 60 ECTS credits of the degree programme. The thesis process itself is split into smaller modules. Rather than a massive whole, this structure has been functional and has facilitated the work. The initial consideration of the thesis topic during the application phase and the further development of ideas as soon as the studies have started have been helpful and valuable in keeping the threshold for starting the thesis relatively low. JAMK's good electronic services, such as the library, thesis guidelines, and ready-made document templates, have also made the work go smoothly.

While writing, the author read a lot of articles, blogs, and books on various topics. Usually, online searches included terms like software architecture, software development, SDLC, and architecture best practices. Many sites and information were consulted several times also in the course of the work. These useful and interesting resources for architects, developers, and designers are summarized in *Appendix 3*.

11.5 Future Research

This work is just the beginning of a systematic development of the application development process in the author's organization. It will provide a basis on which to design and build more detailed procedures, strategies, policies, and new technology use cases, among other things. Using the SDLC, a standard application development model for the organization can be described in more detail, which unifies and thus both improves and enhances software development efficiency.

In addition to the written part of the thesis, the results of a survey of the IT department, which provided information on the current and target state of knowledge and skills, were also used. In addition, many discussions with different people have added to the organization's understanding and valuable capital of technical competence. The visibility of the IT team members and their ability to contribute to the whole organization is therefore essential, and it is worthwhile to retain good talent.

Becoming an iron-fisted professional takes time, the right amount of challenge, and support from the rest of the community. By mastering technologies, tools, and processes, architects and programmers can focus on the primary objective of delivering the vision of the application.

References

Adams, T. et al. (2017). *Autoethnography. The International Encyclopedia of Communication Research Method.* https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118901731.iecrm0011

Agile Alliance. (2001). *Manifesto for Agile Software Development.* http://agilemanifesto.org/

Berners-Lee, T. & Cailliau, R. (1992). *Word-Wide Web*. C.E.R.N. https://cds.cern.ch/record/245440/files/p69.pdf

Brin, S., & Page, L. (1998). *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* https://storage.googleapis.com/gweb-research2023-media/pubtools/pdf/334.pdf

CHM (2020). *Timeline of Computer History.* https://www.computerhistory.org/timeline/computers

Dhiraj K. (2023, Aug 23). *Difference between Scrum, Kanban, and SAFe Agile Methodologies.* https://dhirajkagile.medium.com/difference-between-scrum-kanban-and-safe-agile-methodologies-4bbaeb026de7

Dribble (2023, January 26th). *Top Web Design Trends of 2023*. https://dribbble.com/resources/web-design-trends-2023

Eveleens, J.L., & Verhoef, C. (2008). *The rise and fall of the Chaos report figures.* https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=2b4b1f4bcd92ae588b8d5ca58d38a264ccb19996

Geeks for Geeks. (2017). *Types of Software Testing.* https://www.geeksforgeeks.org/types-software-testing/

GitHub Innovation Graph. (2024). *Programming Languages.* https://innovationgraph.github.com/global-metrics/programming-languages

Google (2000). *Google Code of Conduct.* https://web.archive.org/web/20180421105327/https://abc.xyz/investor/other/google-code-of-conduct.html (24.10.2023)

Gorton, I. (2011). Essential Software Architecture, 2nd ed. Springer. https://janet.finna.fi/Record/jamk.993721754606251?sid=4114452233

Hesterberg, K. (2023, March 7th). *The 29 Dominating Web Design Trends for 2023.* https://blog.hubspot.com/marketing/web-design-trends-2017

IEEE Computer Society. (1999). *Code of Ethics.* https://www.computer.org/education/code-of-ethics

IEEE Std. 610.12-1990. (1990). *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Computer Society. https://ieeexplore-ieee-org.ezproxy.jamk.fi:2443/document/159342

ISO. (2018). *ISO Usability standard. Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts.* https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

Jamk. (2023). *Diary-form thesis.* https://oppimateriaalit.jamk.fi/opinnaytetyo/en/thesis-implementation-methods-and-results/diary-form-thesis/

Khan, I.A. & Kumari, D. (2021). *The Role of Analysis Phase of SDLC for Small Scale Business Application – A Review.* International Journal of Humanities, Engineering, Science and Management, 2(1). https://journal.rkdfuniversity.org/index.php/ijhesm/article/download/82/68

Koikkalainen, M. (2023). *Factors leading to success in IT projects (Bachelor's Thesis)*. University of Jyväskylä, Jyväskylä.

Kolehmainen, A. (2022, October 25th). *Apotti-hankkeen kokonaishinta nousee yli 800 miljoonaan euroon.* https://www-tivi-fi.ezproxy.jamk.fi:2443/uutiset/apotti-hankkeen-kokonaishinta-nousee-yli-800-miljoonaan-euroon/cb69e9c5-4c2e-4a06-b40e-95e678db2a90

Laplante P. A., & Kassab M. (2022). *Requirements Engineering for Software and Systems (4th ed.).* New York: Auerbach Publications. https://doi.org/10.1201/9781003129509

Leon, A. (2015). *Software Configuration Management Handbook. Chapter 2.* https://janet.finna.fi/Record/jamk.993721468506251

Li, A. (2023, July 3rd). *Data is thew New Gold. The Data Revolition: Transforming Industries and Creating New Possibilities.* https://www.masterschool.com/magazine/data-is-the-new-gold/

Lutkevich, B. (2023). *Software Documentation.* TechTarget. https://www.tech-target.com/searchsoftwarequality/definition/documentation

Linux. (2023, October 12th). *Linux source code.* https://github.com/torvalds/linux

Node.js. (2023). *About Node.js*. https://nodejs.org/en/about

Novoseltseva, E. (2019). *User-Centered Design: An Introduction.* Usability Geek. https://usabil-itygeek.com/user-centered-design-introduction/

OpenAI. (2024). *ChatGPT.* https://openai.com/chatgpt

OWASP. (2021). *OWASP Top 10.* https://owasp.org/Top10/

Ponciano, J. (2023). *The World's Largest Technology Companies In 2023: A New Leader Emerges*. https://www.forbes.com/sites/jonathanponciano/2023/06/08/the-worlds-largest-techno-logy-companies-in-2023-a-new-leader-emerges

Satayabrata J. (2023). Difference between System Architecture and Software Architecture https://www.geeksforgeeks.org/difference-between-system-architecture-and-software-ar-chitecture

StackOverflow. (2023). *Most used programming languages among developers worldwide as of 2023* [Graph]. In Statista. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/

Standish Group. (2023). *Benchmarks and Assessments.* https://www.standishgroup.com/bench-mark

Statcounter. (2023, October 12th). *Browser Market Share Worldwide.* https://gs.statcounter.com/browser-market-share#monthly-200901-202308

Statista. (2022). *Number of Amazon.com employees from 2007 to 2022.* https://www-statista-com.ezproxy.jamk.fi:2443/statistics/234488/number-of-amazon-employees/

Tassey, G. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology.* https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf

Waddington, P. (1995). *Information as an asset: the invisible goldmine.* Business Information Review, 12(1), 26-36. https://doi-org.ezproxy.jamk.fi:2443/10.1177/0266382954235537

Widjaja M. (2022). *IT Architecture.* https://www.itarch.info

Appendices

Appendix 1. Data management Plan

1. **General description of data**

A table or list of data that you collect and produce or existing data and its properties (type, file format, rights to use the data, size):

- Current IT technical documentation of services, products, processes and tools, mostly various Office documents
- Survey, MS Forms query
- Interviews’ notes
- Other notes to support thesis writing.

Controlling the consistency and quality of data:

- Current documentation is not consistent, and the quality of data is fickle
- Cataloging information makes it more consistent and manageable
- Survey and interview data is collected more systematically

2. **Personal data, ethical principles, and legal compliance**

Does the data contain personal data (yes/no); measures related to their processing:

- There’s no personal data

Other legislative and ethical factors in the thesis and related actions:

- None

Rights to use, collect, or continue to use the data, possible confidentiality, and related measures:

- As a member of IT department and the author of many IT documents access is granted.

3. **Documentation and metadata**

The author of the thesis makes at least his own notes on how the data has been processed during the research. If the data is saved for further use, where and in what format is the information describing the data saved:

- Relevant data is saved for further use in IT department MS Teams Channels

4. **Storage and backup during the thesis project**

Storage and backup of the data:

- Collected data is stored on cloud storage, which is backed up daily.

Controlling access to your data:

- Most data is available to IT department, work-in-progress documents are on personal access only

5. **Archiving and opening, destroying or storing the data after the thesis project**

Possible archiving and opening of the data and descriptive metadata for reuse:

- Revised documentation and other files are published and saved to IT department MS Teams.

Data to be destroyed and method of implementation:

- Unneeded documents are deleted

Data to be stored for authors and / or the commissioner and storing location:

- Revised documentation and other files are published and saved to IT department MS Teams.

6. **Data management responsibilities and resources**

Responsibilities and possible resources available:

- Most documents are created by author, and other members of IT. The owner of the document is responsible for checking the current status of the documents.

Plan prepared (place and time):

- Helsinki, 28th October 2023

Appendix 2. Questionnaire to IT specialists.

Questionnaire was in Finnish; this is English translation of the questions:

1) Name
2) Education
3) IT experience in years
4) How interesting you find frontend application development?
5) How interesting you find backend application development?
6) Are you more frontend or backend developer? Scale 0-10. Justify your answer.
7) What do you think are the three most important qualities of a software developer?
8) How well do you think you know and know the following things related to system development? Scale 0-4: business goals, project management, teamwork, interaction skills, information management, technology architectures, system design, database design, usability and UX, information security, testing, documenting, integrations, user support, user training, acquisition of IT, agile methods, network solutions, cloud services…
9) How well do you feel you know the following technologies? html, css, javascript, node, databases (sql, pl/sql), angular, vue/nuxt, extjs, wordpress, git, nginx, mule, oauth2, api's, office-applications…
10) How important do you see the use of these [previous] technologies in the future?
11) Which technology would you like to get to know and why?
12) What would you like to learn or know from a colleague?
13) Any other comments regarding your skills or development?

Appendix 3. Interesting and Valuable Resources

Here is a selected list of interesting and valuable software development resources. The list contains resources related to design, programming, architecture, utilities, and learning.

A List Apart is an online magazine focusing on web design and development. It provides insightful articles, tutorials, and discussions on creating effective and accessible web content. https://alistapart.com

Caniuse is a web compatibility resource that allows developers to check the browser support for various web technologies. It provides a detailed database of features and their support across different browsers. https://caniuse.com/

ChatGPT is a natural language processing model developed by OpenAI. It leverages the GPT (Generative Pre-trained Transformer) architecture to understand and generate human-like text based on the input it receives. https://openai.com/chatgpt

Chrome Devtools is a set of web developer tools built into the Google Chrome browser. It allows developers to inspect, debug, and profile web applications, providing insights into the performance and behavior of web pages. https://developer.chrome.com/

CodeProject is an online community and resource hub for developers. It provides a platform to share code snippets, tutorials, and articles, fostering knowledge exchange and collaboration among software developers. CodeProject's mailing lists are precious. https://www.codeproject.com

CSS-Tricks is a web design and development blog that provides tutorials, articles, and tips related to CSS, HTML, and JavaScript. It serves as a valuable resource for front-end developers. https://css-tricks.com/

Dribbble is a platform for designers to showcase their work, share design insights, and connect with other creatives. It serves as a source of inspiration and collaboration within the design community. https://dribbble.com

GitHub is a widely used platform for version control and collaborative software development. It offers hosting for software projects, facilitates code collaboration, and serves as a platform for open-source contributions. https://github.com/

Google is best known for its search engine. It also offers a wide range of cloud-based products and services. Developers often use Google for information retrieval, documentation, and accessing various tools and platforms. https://www.google.com/

Hongkiat.com is a design and technology blog with articles, tutorials, and resources related to web development, design, and technology. It aims to inspire and educate designers and developers. https://www.hongkiat.com/

ITArch.info focuses on information technology architecture. It provides resources, articles, and insights to professionals interested in understanding and implementing effective IT architectures. https://www.itarch.info/

Joel on Software is a blog by Joel Spolsky, a software developer and entrepreneur. The blog covers software development, management, and business topics, offering insights and perspectives based on the author's experiences and expertise. https://www.joelonsoftware.com/

Mozilla Developer Network (MDN) is the official documentation resource for web developers provided by Mozilla. It offers comprehensive guides, tutorials, and documentation on web technologies. https://developer.mozilla.org/

Nielsen Norman Group is a renowned user experience research and consulting firm. Among others, it offers expert insights and research reports to help professionals enhance the user experience in their digital products. https://www.nngroup.com

Node Package Manager (NPM) is the package manager for JavaScript and Node.js. Developers use npm to discover, share, and distribute code packages and manage project dependencies. https://www.npmjs.com/

Node.js is an open-source, server-side JavaScript runtime enabling developers to build scalable, high-performance applications. It allows the execution of JavaScript code outside the web browser. https://nodejs.org/

Roadmap.sh is a website that offers learning roadmaps for various technologies and programming languages. It helps individuals navigate and plan their learning journey in the ever-evolving field of software development. https://roadmap.sh

Smashing Magazine is an online publication covering a wide range of web design and development topics. It offers articles, tutorials, and resources to help professionals stay updated with industry trends. https://smashingmagazine.com/

StackOverflow is a widely used online platform where developers can ask, answer, and discuss programming-related questions. It is a valuable resource for the global developer community seeking solutions to coding challenges. https://stackoverflow.com

Visual Studio Code (VS Code) is a popular, open-source code editor developed by Microsoft. It supports various programming languages and extensions, providing developers with a customizable and efficient coding environment. https://code.visualstudio.com/