Erzincan Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
2024, 17(3), 884-904
ISSN: 1307-9085, e-ISSN: 2149-4584
Araştırma Makalesi

Erzincan University
Journal of Science and Technology
2024, 17(3), 884-904
DOI: 10.18185/erzifbed.1581945
Research Article

A Mobile Secure Content Development Application for Children in the Software Development Life Cycle Framework

İsakhan KARABAŞ[1*], Fulya ASLAY[1], Elif ÖKSÜZ[1]

[1]Department of Artificial Intelligence and Robotics, Institute of Science and Technology, Erzincan Binali Yıldırım University, 24030, Erzincan, Turkiye.

Received: 08/11/2024, **Revised:** 16/12/2024, **Accepted:** 16/12/2024, **Published:** 31/12/2024

Abstract

This study focuses on the software development process of a mobile application that aims to support children's learning, development and entertainment processes while ensuring their safety in the online environment. The project, which was carried out within the framework of the Software Development Life Cycle (SDLC), adopted a systematic approach that included requirements analysis, design, development, testing and maintenance phases. This systematic structure enabled the project to proceed in a planned manner and effective use of resources. The project aims to minimize digital security risks, provide appropriate content for children and enable parents to safely control their children's internet use. Extreme Programming (XP) methodology was preferred in the SDLC development phase. The principles specific to this methodology such as pair programming, test-driven development (TDD) and continuous integration were effectively applied. The use of XP enabled user feedback to be processed quickly and the application to be continuously improved. Thus, the software adapted not only to technical requirements, but also to user experience and security needs. This process, which successfully completed security and performance tests, demonstrates that effective management of the entire process, not just coding, in software development projects directly contributes to project success. The study aims to contribute to the literature on children's digital safety and to provide a basis for future research in this field.

Keywords: Methodology, SDLC, Software Engineering, Software Testing, XP

Yazılım Geliştirme Yaşam Döngüsü Çerçevesinde Çocuklar için Bir Mobil Güvenli İçerik Geliştirme Uygulaması

Öz

Bu çalışma, çocukların çevrimiçi ortamda güvenliğini sağlarken, öğrenme, gelişim ve eğlence süreçlerini desteklemeyi amaçlayan bir mobil uygulamanın yazılım geliştirme sürecini ele almaktadır. Yazılım Geliştirme Yaşam Döngüsü (SDLC) çerçevesinde yürütülen proje; gereksinim analizi, tasarım, geliştirme, test ve bakım aşamalarını içeren sistematik bir yaklaşımı benimsemiştir. Bu sistematik yapı, projenin planlı bir şekilde ilerlemesini ve kaynakların etkili kullanımını mümkün kılmıştır. Proje, dijital güvenlik risklerini minimize ederek çocuklara uygun içerik sunmayı ve ebeveynlerin çocuklarının internet kullanımını güvenle kontrol edebilmesini sağlamayı hedeflemektedir. SDLC geliştirme aşamasında Ekstrem Programlama (XP) metodolojisi tercih edilmiştir. Bu metodolojiye özgü olan çift programlama, test odaklı geliştirme (TDD) ve sürekli entegrasyon gibi prensipler etkin bir şekilde uygulanmıştır. XP'nin kullanımı, kullanıcı geri bildirimlerinin hızlı bir şekilde işlenmesini ve uygulamanın sürekli iyileştirilmesini sağlamıştır. Böylece yazılım, yalnızca teknik gereksinimlere değil, aynı zamanda kullanıcı deneyimi ve güvenlik ihtiyaçlarına da uyum sağlamıştır. Güvenlik ve performans testlerini başarıyla tamamlayan bu süreç, yazılım geliştirme projelerinde yalnızca kodlama değil, tüm sürecin etkin yönetiminin proje başarısına doğrudan katkı sağladığını ortaya koymaktadır. Çalışma, çocukların dijital güvenliği alanındaki literatüre katkı sunmayı ve bu alanda gelecekte yapılacak araştırmalar için bir temel oluşturmayı hedeflemektedir.

Anahtar Kelimeler: Metodoloji, SDLC, Yazılım Mühendisliği, Yazılım Testi, XP

*Corresponding Author: isakhan.karabas@ogr.ebyu.edu.tr
İsakhan KARABAŞ, https://orcid.org/0009-0006-0852-1996
Fulya ASLAY, https://orcid.org/0000-0001-5212-6017
Elif ÖKSÜZ, https://orcid.org/0009-0002-8743-2676

1. Introduction

The process of developing a software product is considered as the software life cycle and consists of various phases such as planning, analysis, design, implementation, testing and maintenance [1,2]. These phases are implemented sequentially through a model. The software development model expresses the strategy for the realization of the software and this strategy includes a set of activities, objects, transformations and events [3]. The software process includes all activities involved in software development. All activities of specification, development, verification and evolution are part of all software processes. In addition to the functionality and performance that customers expect, a good software product is expected to be easily modifiable according to changing customer requirements. In other words, a good software product should be maintainable, reliable, efficient and acceptable. In order for a software product to be successful, all activities planned throughout the software lifecycle process must be realized within the planned time and cost scope.

Today, a wide variety of software such as operating systems, mobile applications, web applications, cloud-based applications are being developed. The developed software can appeal to very different audiences. The software development process should be implemented by choosing the right methodology according to the scope and scale of the software. In today's digital world, it is of great importance to ensure that children use the internet safely, to provide them with appropriate safe and educational content, and to enable parents to safely control their children's internet use. For this reason, it is necessary to have applications that contain safe content for children. At the same time, content producers should be guided in this area. Managing the software processes required for the realization of such applications is also of great importance in terms of both time and cost. For this reason, this study focuses on a medium-sized mobile software application that contains safe content for children. In today's digital world, it is of great importance to ensure that children use the internet safely, to provide them with appropriate safe and educational content, and to enable parents to safely control their children's internet use. For this reason, it is necessary to have applications that contain safe content for children. At the same time, content producers should be guided in this area. Managing the software processes required for the realization of such applications is also of great importance in terms of both time and cost.

The term digital security is often used interchangeably with internet security, cyber security, online security, information security and data security. While some researchers in the field of early childhood education see the benefits of digital technology in supporting children's learning and social interactions, others have concerns about its negative impacts on children's health and development. Children growing up in the digital age from an early age are exposed to digital technology and overshadowed by digital security threats. According to the OECD (2021), digital security risks in children can be divided into 4Cs (concerning contact, content, conduct, and contract risks). Children also have privacy risks, advanced technology risks, and health and well-being risks. This often causes parental concern and tends to discourage children's use of technology. There is an important need to raise parents' awareness about the

impact of digital devices on children's health and development and to ensure that they use the internet safely [4].

Cyberbullying is increasingly recognized as a threat to the mental health of children and young people. Children, young people and their families may not know how to stay safe online or how to respond to unsafe online experiences. Child and Adolescent Mental Health Physicians believe that new generations should be guided by education from a young age, but it is impossible to reach everyone [5].

In this study, the software of a mobile application that will support children's learning, development and entertainment processes while keeping them safe in the online environment was realized within the framework of the Software Development Life Cycle (SDLC). This process was carried out by progressing step by step with the principles of continuous improvement within the framework of the software development life cycle. Within the scope of the study, requirements determination, system design, development, testing and maintenance phases are included.

2. Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a systematic process that covers all phases of software projects from planning to deployment and maintenance. Careful and meticulous work at every stage of the software development lifecycle and the application of the right methodologies contribute significantly to the successful completion of the project by improving the quality of the software [6].

SDLC is used in the software industry to design, develop and produce high-quality, reliable, cost-effective and timely software products. It is also called the software development process model [6]. SDLC models form the backbone of software engineering practice by guiding a systematic and structured approach to creating high quality software products. As technology evolves and market demands become more dynamic, software development organizations are faced with the challenge of selecting the most appropriate SDLC model to meet project requirements efficiently and effectively. Developers, project managers, and quality engineers should have a thorough understanding of the advantages, disadvantages, and applicability of various SDLC models in the context of software quality engineering in order to make informed decisions [7].

The SDLC is a fundamental framework that guides the process of building software applications, including mobile applications. It consists of a series of well-defined phases designed to ensure the development of high-quality software that meets or exceeds customer expectations [7,8].

2.1 SDLC Methodology Phases

SDLC methodologies provide a structured framework that guides the process of designing, developing and implementing software solutions. These methodologies are diverse and each offers an approach tailored to a specific project needs, complexity and objectives. One of the key aspects of the SDLC is its ability to adapt to the unique requirements and constraints of

different software projects, including mobile applications [10,11]. A visualized description of the stages in the SDLC is presented in Figure 1 [9].

- Planning
  1. Define project scope, objectives and requirements
  2. Identify stakeholders and roles
  3. Create a project plan, including timelines and resources
- Analysis
  1. Collect and document detailed requirements from users and stakeholders
  2. Analyze the information collected to understand the functional and non-functional requirements of the system
  3. Develop use cases, user stories or functional specifications
- Design
  1. Create an architectural design that defines how software components interact
  2. Develop detailed technical specifications
  3. Designing the user interface
- Application (Coding)
  1. Writing the actual code based on design and specifications
  2. Perform unit tests to ensure that individual components function as intended
  3. Integrate code modules as needed
- Testing and Merging
  1. Conduct various tests, including integration testing, system testing and user testing
  2. Identify and correct defects and problems
  3. Verify that the software meets the completed requirements
- Maintenance and Sustainability
  1. Monitoring and maintenance of the software
  2. Addressing and correcting reported problems or defects
  3. Make necessary updates, enhancements and patches

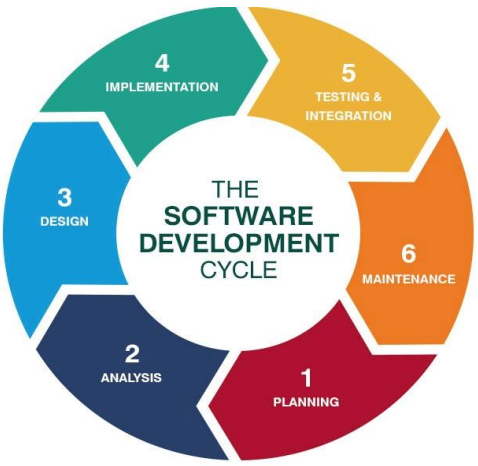

Figure 1. Software lifecycle.

2.2 Using Decision Support Matrix for Selection of SDLC Methodologies

In software development, selecting an appropriate SDLC methodology is a critical decision that significantly impacts project outcomes [10]. To effectively carry out this decision-making process, the use of a decision support matrix is invaluable. A decision support matrix provides a structured approach to evaluate and compare various SDLC methodologies based on their compatibility with project-specific factors, enabling stakeholders to make informed and strategic choices [11].

A decision support matrix considers several key factors that influence the suitability of an SDLC methodology for a given project:

Project Complexity: The complexity of a project's requirements, architecture and technology stack can influence the choice of SDLC methodology. Projects with high complexity can benefit from methodologies that allow iterative development and frequent adaptations [12].

Stability of Requirements: The stability of project requirements is a very important consideration. Projects with well-defined and stable requirements may tend towards a linear approach, while projects with evolving requirements may prefer Agile methodologies [6].

Customer Involvement: The extent to which customers or end users should be involved throughout the development process plays an important role. Agile methodologies emphasize regular customer feedback, while traditional methodologies may involve customers primarily in the requirements phase [12].

Flexibility and Adaptability: The project's capacity to accommodate changes and adapt to evolving needs is an important factor. Agile methodologies excel in providing flexibility, while traditional methodologies may struggle to accommodate late-stage changes [6].

Timeline and Predictability: The timeline requirements of the project and the organization's need for predictability in terms of project milestones and deadlines are considered. Traditional methodologies offer a relatively predictable timeline, while Agile methodologies embrace change and evolution [13].

Creation of Decision Support Matrix;

Factor Weighting: Assign relative weights to each factor based on its importance to the project [14]. For example, if customer engagement is very important, it may receive a higher weight.

Scoring Criteria: For each factor, scoring criteria can be defined, usually on a scale from low to high or inadequate to excellent [15,16].

Methodology Assessment: Evaluate each SDLC methodology against factors and criteria and points are awarded based on compliance [17].

Calculation of Weighted Scores: Multiply the assigned weights by the scores for each factor for each methodology. Sum these weighted scores for each methodology to obtain the total weighted score [18,19].

Selection: The methodology with the highest overall weighted score is the recommended choice [11].

The general structure of the decision support matrix is shown in Figure 2.

Decision Support Matrix		Methodologies	
Scoring Criterias	Factor Weights	Agile Methods	Traditional Methods
A1			
A2			
A3			
Decision			

Figure 2. Decision support matrix.

2.3 Software Development with SDLC Methodologies

2.3.1 Traditional Software Development Methodologies

It is an approach that generally follows the planning, design, development, testing and maintenance phases in a sequential and systematic way. It is used in complex projects where requirements are clear, large teams are working and complex projects. The most common traditional software development models include the cascade model, the V-process model and the spiral model.

The cascade model is one of the oldest methods used in software development processes. It is an approach in which project steps are sequential and progressive, one step is not completed until the next step is completed, and it is difficult to go back to the previous step.

The V process model is considered an improved version of the waterfall model. The software development process has a test phase corresponding to each development phase. Thus, each phase is verified. It is preferred in projects with high quality requirements.

The spiral model places great emphasis on risk analysis and risk management, which are ignored in the waterfall model. The project is divided into cycles and each cycle is detailed and its risks are assessed separately. Therefore, it is suitable for high-risk projects. It is also close to contemporary models.

2.3.2 Agile Software Development Methodologies

They are software development approaches that enable fast and effective response to customer requirements and can easily adapt to all kinds of changes [20]. It allows software development projects to be developed in short cycles by dividing them into small, manageable parts. It is used in situations where it is difficult to predict the detailed roadmaps and designs of projects. The most common agile software development methods include Scrum, Kanban, Extreme Programming (XP) and Lean software development [21].

The Scrum model is an agile software development model and its general characteristic is that it is observer, developer and iterative. The Scrum model is used when it is difficult and complex to plan a software project completely from the beginning. According to this method, a team is formed for the software development process and a scrum master responsible for the team is determined. This enables teams to work faster and more efficiently. Scrum teams ensure the progress of the project by holding sprint meetings at certain intervals [22].

Kanban is a visual management methodology that aims to make production processes and workflows more efficient. It is used in software development processes in the form of scheduling to control the phases [23].

XP is an agile software development methodology designed to improve software quality and responsiveness to changing customer requirements. XP is particularly effective in dynamic environments where requirements are likely to change because it encourages adaptability and communication. Team roles in XP vary, but key positions include developers, customers (or their representatives), coaches and monitors who collectively prioritize tasks, write user stories and evaluate progress. XP increases productivity and fosters innovation by fostering a culture of trust and shared responsibility, making it a valuable methodology for teams that want to deliver high-quality software in a flexible, customer-centric way [24].

Lean software development is adapted from lean manufacturing principles and aims to make software development processes more efficient and effective. It is a methodology used to minimize waste in software development processes, maximize customer value and ensure continuous improvement [25].

3. Mobile Application Project Management

Mobile application project management is the management of the processes and tools used in the development of mobile applications. This enables both organizing software development teams and achieving specific business goals. Successful mobile app project management ensures that the development process is completed on time and within budget, ensuring that the app meets the targeted quality and user experience standards.

In this project, a mobile application will be developed to create a clean platform for children that is easy to use, has a high level of security and at the same time does not contain inappropriate content. The application is designed to work on iOS and Android platforms. The stages of the application within the scope of SDLC are presented below;

3.1 Software Methodology Selection

XP stands out as one of the agile software development methodologies. Developed by Kent Beck in the 1990s, this methodology aims to maximize customer satisfaction, improve teamwork and adapt to changing requirements in software projects. XP is a particularly suitable approach for projects with frequently changing requirements and where fast deliveries are critical [24]. XP is built on a set of fundamental principles. These principles include small and frequent releases, test-driven development (TDD), pair programming, continuous integration, simple design, continuous feedback, code standards and the 40-hour work week. These

principles are designed to make the software development process more efficient and agile. For example, through small and frequent releases, software is delivered in such a way that customers can provide early feedback [26]. TDD ensures that tests are written before the coding process and bugs are detected early [27]. Pair programming improves code quality by encouraging developers to work together [28]. Continuous integration ensures that new code is continuously integrated into the main code base and that this process is tested [29]. Many advantages are gained through the use of XP. These include fast feedback, quality coding and flexibility to change. Frequent deliveries and customer involvement allow the software to be improved in the early stages [30].

Agile software development methodologies include Scrum, Kanban and Feature-Driven Development (FDD). A decision-making matrix was created to decide which software development method to use for the application planned to be made within the scope of this project, and as a result, it was deemed more appropriate to use the XP software development process in our software project due to the different sections it contains. XP was chosen especially because of its advantages of handling user feedback quickly, continuous improvement of the system through frequent releases and maintaining high quality standards. According to this software development process, each section is designed, coded and tested separately. After the testing phases of the sections are completed, all sections are merged. This user-oriented method contributed to the app providing an effective solution to ensure children's digital safety.

3.2 Software Requirements Analysis

Requirements analysis is critical to the success or failure of a software project. Requirements analysis should be performed in order to determine customer expectations in the initial phase of software projects and to minimize the errors that may occur in the later stages of the software project. The requirements determined in the early stages of the project may change as the project progresses. Therefore, they are dynamic. They are divided into two as functional and non-functional requirements. The steps of requirements analysis are as follows;

- Requirements Gathering: Gathering needs from users and stakeholders. This can be done through surveys, interviews and observations.
- Requirements Definition: Analyzing the information collected and transforming it into specific, measurable and achievable requirements.
- Requirements Documentation: Detailed documentation of requirements. These documents serve as a guide for the project team.
- Requirements Verification and Validation: Checking the accuracy and completeness of requirements and validation by stakeholders.
- Requirements Management: Monitoring requirements and managing changes throughout the project [31,32].

These steps ensure that the project achieves its goals and meets user needs. Some common techniques used in the requirements analysis process include flow diagrams and prototyping [33].

For the development of the application, requirements were first analyzed. For this, the functional and non-functional requirements that make up the software were identified. Then, a feasibility study was conducted for the analysis and it was decided whether the project should be carried out or not. Cost and time analysis were made within the scope of the feasibility study.

3.2.1 Functional Requirements

Requirements are expressed as services provided directly to the user in order to meet the user's needs and expectations. It defines what the system should do. Some functional requirements of the software developed in the study are presented below;

- Users should be able to register and login to the application with e-mail and password.
- Users should be able to create multiple child profiles and set age-appropriate content filters for each profile.
- The system should be opened according to the authorization level of the logged in user.
- Users should be able to search for content in the system using the content search feature.
- The user should be able to update their profile information (name, profile photo, etc.).
- Users should be able to access detailed information about each content such as author, publication date, age group.
- The user should be able to play the game of their choice within the app and track their achievements in the games.
- Users should be able to play online multiplayer games with other players.
- The user should be able to watch the cartoon of their choice within the application.
- The user should be able to read the books they have selected within the application or listen to them as audiobooks.
- The user should be able to add any content to their favorite list.
- Parents should be able to limit the usage time of the app and view viewing and reading history.
- If there is a problem with the password, a password renewal link should be sent to the e-mail address.

3.1.2 Non-Functional Requirements

It is a performance requirement that addresses the quality and correct operation of the software, not the service provided directly to the user. It defines how the system should perform. Non-functional requirements help to ensure that the user's needs are met, that it is reliable. It also helps to make it easier to use and maintain.

- The system must be secure against unauthorized access. It should ask for member login.
- The application should scale to serve 100000 users at the same time.
- Access to the application should be provided via the internet.
- The app's response time to user requests should not exceed 2 seconds.
- Loading and playback of books, videos and games in the app should not exceed a maximum of 3 seconds.
- Age verification mechanisms should be in place to control access to content.

- The user interface should be simple and colorful so that children can use it easily.
- There should be a feedback mechanism where users can easily report bugs in the application.
- The application should work flawlessly on all operating systems.
- The application should be suitable for commonly used web browsers.
- If there is a problem with the password, a password renewal link should be sent to the e-mail address.
- The app should be maintained on a monthly basis so that new content and updates can be easily added.
- The application should be tested weekly for vulnerabilities.

3.3 Software Feasibility Analysis

In software projects, a feasibility analysis should be performed to evaluate the feasibility of the project by taking into account all factors related to the project. This analysis aims to identify resources, risks and opportunities to increase the chances of success of the project. In feasibility analysis, evaluation is made from various angles and it is aimed to increase the efficiency of the project. Within the scope of this project, cost analysis and time analysis were conducted.

3.3.1 Cost Analysis

In general, cost estimation is one of the most difficult steps in project management. It is very important to accurately calculate the resources and schedule needed. The software costing process includes calculating the size of the software to be produced, calculating the effort required, developing the project schedule, and finally calculating the cost of the entire project [34]. There is a wide variety of cost estimation methods for evaluating projects, and there are quite a few studies that favor the COCOMO model for software projects. A study covering 115 different software projects highlighted that costs and timelines are often under- or overestimated, often due to a lack of structured estimation methods. By applying COCOMO, this study addressed common problems such as frequent changes made by users, missed tasks, and inadequate analysis, and helped to improve estimation accuracy by considering complexity, system size, and team capabilities [35]. In a study focused on creating an online bookstore, the COCOMO model was used to estimate development efforts. The project included 14 web pages written in HTML and JavaScript, and the application was categorized under the “organic” mode using both the basic and intermediate COCOMO sub-models. This approach assisted in effort estimation based on 2.9 KLOC (thousands of lines of code) and adjusted the predictions using cost drivers such as reliability, database size, and team experience [36]. The COCOMO model is used to estimate the cost, duration, and workforce requirements of a software project by considering its size and complexity. It was developed by Barry W. Boehm in 1981, based on the principle that the required effort is proportional to an exponential function of the program's size [12]. COCOMO estimates workload in software projects by considering developers' skills, software complexity, and the technologies used. The structure of the COCOMO model is presented in Figure 3 [37].

Figure 3. COCOMO model.

The COCOMO model has three fundamental types based on different project types and complexity levels: discrete, semi-detached, and embedded. The effort and duration formulas for these three models are shown in Table 1 [37,38]. Discrete projects are those developed by small, experienced teams, characterized by a well-understood scope and low complexity. Semi-detached projects refer to projects of medium size and complexity, typically managed by expert teams. Embedded projects are very large, complex projects that require specialized equipment or hardware.

Table 1. COCOMO model effort and duration formulas.

Mode	*Effort*	*Schedule*
Organic	$E=2.4*(KDSI)^{1.05}$	$TDEV=2.5*(E)^{0.38}$
Semidetached	$E=3.0*(KDSI)^{1.12}$	$TDEV=2.5*(E)^{0.35}$
Embedded	$E=3.6*(KDSI)^{1.20}$	$TDEV=2.5*(E)^{0.32}$

The cost multiplier in the model is obtained from the product of 15 cost drivers.

COCOMO equations; E=a×(KLOC)b [12]

E: Effort, **KLOC**: Lines of Code, **a ve b**: Constants Determined by Model Type

The cost multiplier for the mobile application developed in the study has been determined as shown in Table 2.

Table 2. Mobile application cost drivers

COST DRIVERS		RATINGS						PROJECT
		VERY LOW	LOW	NOMINAL	HIGH	VERY HIGH	EXTRA HIGH	
PRODUCT ATTRIBUTES	RELY : Required software reliability	0,75	0,88	1	1,15	1,4	-	1
	DATA : Size of application database	-	0,94	1	1,08	1,16	-	1,08
	CPLX : Complexity of the product	0,7	0,85	1	1,15	1,3	1,65	1
HARDWARE ATTRIBUTES	TIME : Run time performans constraints	-	-	1	1,11	1,3	1,66	1
	STOR : Memory constraints	-	-	1	1,06	1,21	1,56	1
	VIRT : Volatility of the virtual macine environment	-	0,87	1	1,15	1,3	-	1
	TURN : Required turnabout time	-	0,87	1	1,07	1,15	-	1,07
PERSONNEL ATTRIBUTES	ACAP : Analyst capability	1,46	1,19	1	0,86	0,71	-	1
	AEXP : Applications experience	1,29	1,13	1	0,91	0,82	-	1
	PCAP : Software engineer capability	1,42	1,17	1	0,86	0,7	-	1,17
	VEXP : Virtual machine experience	1,21	1,1	1	0,9	-	-	1
	LEXP : Programlama dili deneyimi	1,14	1,07	1	0,95	-	-	0,95
PROJECT ATTRIBUTES	MODP : Application of software engineering methods	1,24	1,1	1	0,91	0,82	-	0,91
	TOOL : Use of software tools	1,24	1,1	1	0,91	0,83	-	1,1
	SCED : Required development schedule	1,23	1,08	1	1,04	1,1	-	1

In this context, the effort coefficient for the software project has been calculated as follows;

C = C1*C2*C3*…*C15 = 1,28

The size of the software is approximately 15000 LOC=15 KLOC and it is evaluated as a semi-detached application.;

Effort = 3 * 15^1.28 = 96 man/month

Duration = 2.5* 96^0.35 = 12,3 month

N = 96/12,3 = 7,8 = 8

Accordingly, it has been estimated that the project can be developed by a team of 8 people in approximately 12 months.

3.3.2 Time Analysis

Time analysis is an evaluation that includes planning and monitoring the steps required to complete a software project within the targeted duration, the time allocated for these steps, and the business processes involved. Through time analysis, it helps determine when the project will start and finish, identify critical paths, and recognize potential delays, contributing to the timely and budget-compliant completion of the project. One of the most common methods for time analysis is the Gantt chart. A Gantt chart visualizes the start and end dates of specified time periods as horizontal bars. It is a simple and clear visual tool for tracking and managing project progress. Considering the duration obtained for the project within the framework of the COCOMO model, a time analysis has been conducted and is illustrated in Figure 4 using a Gantt chart. The chart shows the time analysis for the 12 months required to complete the project based on the planned tasks. Accordingly, customer meetings and target audience analysis were completed in the first month, followed by the determination of requirements. A cost analysis was performed based on the identified requirements, and immediately after that, the design phase commenced. After the design, the implementation phase—the longest phase of the project—began, during which testing was conducted for each unit as the application was developed. Performance and security testing started after the application was developed, and once these tests were completed, the software project was launched with a beta version. Subsequently, the project entered the maintenance phase with feedback from end users and requests for software updates, finalizing the software.

Gantt Chart	Month 1	Month 2	Month 3	Month 4	Month 5	Month 6	Month 7	Month 8	Month 9	Month 10	Month 11	Month 12
Customer meetings												
Target audience analysis												
Determination of the project team												
Requirements identification												
Cost analysis												
Database design												
User interface design												
Application development												
Unit tests												
User testing												
Performance and security testing												
Adding beta version												
Updates												
Adding new features												

Figure 4. Gantt diagram for the developed application.

3.4 Software Design

The software design phase is one of the most critical and important stages in the software development process, occurring before the coding phase. There are three different software design models: Architectural design, which involves designing program models using UML (Unified Modeling Language) graphic language and GUI (Graphical User Interface) design. Each model generates various sub-models. Architectural design is created by establishing the foundations of the software and connecting these building blocks to form the design. This involves determining how the software's modules and sub-modules relate to the database and other structures, thereby designing how the system will operate. Here, client-server architecture, layered architecture, and repository architecture can be employed [39].

The study plans to develop the software for mobile platforms. Mobile software development requires rapid updates and enhancements to meet the varying capacities of different customers in a mobile environment, which is subject to failures and a series of constraints. The evolving constraints also include existing limitations such as technology, resources, bandwidth, and coverage area in the future [40].

In object-oriented design, specialized design tools are used. UML is a standard modeling language commonly preferred for modeling systems where object-oriented programming techniques are planned to be used. UML is employed to provide standards for complex programs that require the consideration of both software and hardware, especially in situations where the code will be developed by multiple developers [41].

UML diagrams are graphical representations that illustrate different aspects and features of a system. When modeling a system, the appropriate diagram is selected and drawn based on the specific aspect that needs to be examined. Depending on the specialized needs of the developers, some programmers maintain a strict approach and take great care in using all the diagrams and definitions from the Object Management Group (OMG), potentially wasting time on unnecessary drawings. In contrast, others may settle for just using use case and class diagrams [41]. Use Case diagrams are a sub-module of UML diagrams, and during the software design development process, a use case diagram is used to model how a system interacts with its users. This diagram aims to visualize the functions offered by the system, the users who will utilize these functions, and the relationships between the functions. In software development processes, use case diagrams help simplify the visualization of complex system functionalities, enabling users to understand the system easily [42,43]. Accordingly, when creating the diagram, the functions that the system will offer and the users who will utilize the system are determined. Based on the functions provided by the system or the expectations of the users from the system, use cases are created, visualizing the relationships between the users and the use cases.

A general use case diagram illustrating the actors and their permissions for the mobile application developed in the study is displayed in Figure 5.

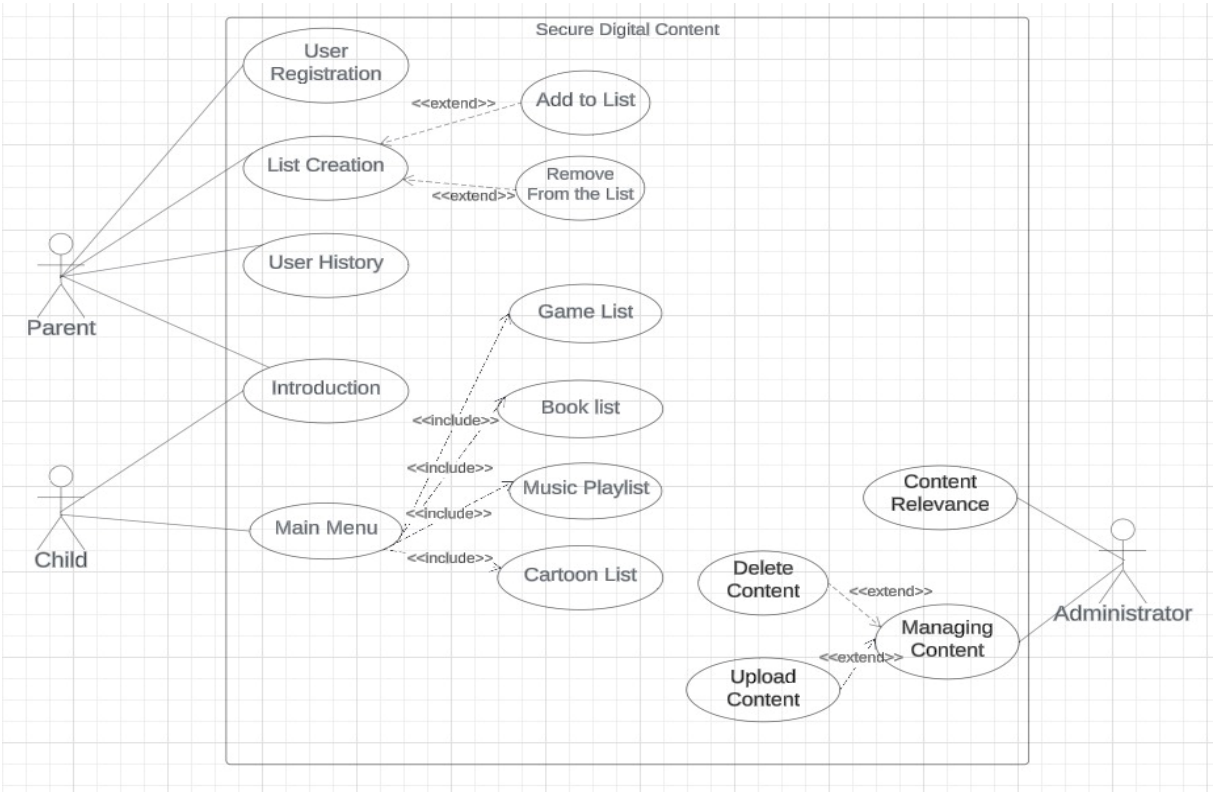

Figure 5. Use case diagram of the application.

Accordingly, a user with "Administrator" privileges can perform content compliance and addition/deletion operations. The registration and list creation processes are carried out by the "Parent" user, while the "Child" can access the created lists. This access helps prevent the child from navigating inappropriate content.

3.5 Software Testing

The testing phase, which is a part of the software development process, is a crucial stage for ensuring the functionality and quality of the software. The aim of the testing phase is to identify and rectify any potential errors in the software beforehand. Some of the tests conducted during the software development process include unit tests, integration tests, performance tests, user tests, security tests, and acceptance tests [44].

When the testing process begins, it is essential to plan in advance which tests will be conducted, how they will be performed, when they will take place, who will conduct them, and which tools will be used. Tests should be executed, results recorded, and necessary improvements made. Following these improvements, the tests should be repeated, and after all these testing stages, the success of the system should be evaluated.

Within the scope of the study, it is planned to test each module of the application separately for different content sections. For example, in the application's book list section, it will be tested whether the pages of the books displayed change individually and within a specified time frame. Based on the results, improvements will be made if necessary.

When the sections are integrated, it is essential to verify how the system works together and to check the data flow and compatibility between them. Additionally, since further additions to the created application are planned, it is intended to ensure the appropriateness of each content that is considered for inclusion.

It is planned to test the response times of each content section of the application, assess its performance under extreme conditions, identify any security vulnerabilities, and evaluate whether it is user-friendly.

Finally, it is planned to test whether the entire system operates according to its requirements across different devices, browsers, and operating systems. Each of these tests is critical to ensuring the application's functionality. Additionally, it is necessary to carefully analyze the results of these tests to manage the continuous improvement process with feedback at every stage.

3.6 Software Maintenance

In software development projects, the maintenance phases encompass activities aimed at ensuring the software operates effectively and reliably throughout its lifecycle after development. The maintenance phases are critical for the sustainability and longevity of a project. Specifically, regular maintenance activities lead to performance improvements and the resolution of security vulnerabilities [45,46]. Types of maintenance to be implemented in the study;

- Preventive Maintenance: Periodic security updates and system enhancements will be performed to improve the application's security and performance. Preventive maintenance is designed to prevent future issues with the software.
- Corrective Maintenance: Rapid identification and resolution of errors reported by users will be ensured. This type of maintenance is critical for enhancing the software's user experience. Additionally, regular updates will be planned based on user feedback.
- Performance Maintenance: Performance optimizations will be implemented to enhance the speed and efficiency of the application. The software's response times, resource usage, and other performance metrics will be regularly analyzed.
- Test Maintenance: This involves updating and maintaining software tests. Test scenarios need to be updated to accommodate new features or changes in the software.
- Training Maintenance: Training processes and materials will be developed to enable users to utilize the application most effectively. This aims to ensure that users are informed about the system and can troubleshoot potential issues.

In the study, rapid identification and resolution of errors reported by users will be ensured. Considering user feedback allows for updates to the system according to user needs, thereby continuously enhancing the application's speed and efficiency. Regular updates will be released to address security vulnerabilities and implement performance improvements. Conducting regular updates ensures the platform's resilience and security against technological advancements. Performance optimizations will be carried out to enhance the application's speed and efficiency. User feedback will be collected and evaluated regularly. This will ensure that

the application remains a long-lasting and secure platform, adapting to user needs and technological developments [45].

In a software lifecycle, the maintenance phase is continuous, meaning that every request (such as software bugs, new plugin requests, etc.) leads the lifecycle back to the beginning, and the software development process continues through the same stages. Therefore, this process takes the form of a cycle.

4. Conclusion & Discussion

In the study, the software development phases of a mobile application that ensures children's safety in the online environment while supporting their learning, development, and entertainment processes have been managed. Within the framework of the software development lifecycle, phases such as requirement identification, feasibility analysis, system design, development, testing, and maintenance have been prepared.

In the initial phase of the project, the needs of children and parents were analyzed in detail to determine requirements, leading to the design of a user-friendly, safe, and educational interface. By adopting the XP methodology, a lifecycle process aimed at continuous improvement has been implemented.

Security and performance tests of the application have been conducted to ensure smooth operation across all systems. During the maintenance phase, quick responses to user feedback have been provided to keep the application continuously updated and secure. Additionally, with the chosen methodology, the mobile application developed in the study will continue to be enhanced in line with the principles of continuous improvement, and updates will be planned to incorporate new features based on user feedback.

In software development projects, the code development phase is often considered the most important. However, the successful completion of a project relies on effective process management throughout all phases, from the project's inception to its final stages. Continuing the maintenance processes with the same steps after the project is completed enhances the project's success.

Life cycle models need to be rigorously applied to ensure that software projects meet user requirements [47]. He emphasized that digital security is a key element for developing more reliable apps, especially for vulnerable user groups such as children [48].

Studies on SDLC and secure digital content development projects for children prove the importance of security-focused SDLC methodologies. For example, threat modelling and security testing applied in Microsoft's Secure Development Lifecycle (SDL) model helps teams developing content for children to identify and eliminate security vulnerabilities at an early stage [49].

Research by Snyk shows that security assessments integrated into each phase of the secure SDLC improve the security of the software. As stated in Snyk's work on SDLC practices,

implementing security controls at the requirement and design stages significantly reduces the cost of changes to be made at later stages and increases security. In addition, they emphasize that the adaptation of SDLC in secure mobile applications for children is effective in protecting children's online safety and privacy rights [50].

Existing research shows that providing safe content for children is not only a software development issue, but also a critical issue for children's health and development. This study contributes to this literature and argues that analyzing user requirements, enabling parental control mechanisms and conducting digital security tests are necessary in the process of developing safe content for children [6,47,48,51].

In the literature, there are similar studies on digital security, software development processes and creating safe digital content for children. This study will contribute to the literature by combining digital security and user experience in the field of safe mobile content development for children. In the future, more comprehensive research on the personalization of applications for different age groups, the development of parental control tools, and the integration of digital safety tools with educational content can be recommended.

This study contributes to the literature by combining digital safety and user experience in the field of developing safe mobile content for children. However, there are some limitations. First, the study only addressed the development of a generic mobile application for children of a certain age group. It is thought that personalized solutions for different age groups may be more effective. Secondly, parental control mechanisms and digital safety tools could be expanded to analyze children's interactions with different digital environments in more detail. Furthermore, the methodology developed in this study has not been tested for its applicability to larger or different scale projects. For future research, it is recommended to conduct more comprehensive studies on customizing applications for different age groups, developing parental control tools, and integrating digital safety tools with educational content. In addition, it would be useful to conduct long-term studies to evaluate the impact of the developed applications on user satisfaction and safety in the long term. Such studies will contribute to the improvement of both software development processes and user experience.

Ethics in Publishing

There are no ethical issues regarding the publication of this study

Author Contributions

In the preparation of the publication, the authors have acted jointly in all parts.

References

[1] Curtis, B., Krasner, H., Iscoe, N., (1988). A field study of the software design process for large systems. Commun ACM; 31: 1268–1287.

[2] Rod S. BEGINNING Software Engineering, Second Edition, (2024). Beginning Software Engineering, Second Edition; 1–685.

[3] Setyantoro A. Process Models in Software Engineering. https://www.academia.edu/36272460/Process_Models_in_Software_Engineering.

[4] Sari, P.I., Rahmawati, I., Mariyana, R., Charmeida, N. (2024) The Correlation Between Parental Awareness and Concern to The Early Childhoods' Digital Safety. Kiddo: Jurnal Pendidikan Islam Anak Usia Dini, 5, 310–326.

[5] Lonergan, A., Moriarty, A., McNicholas, F., Byrne, T. (2023) Cyberbullying and internet safety: a survey of child and adolescent mental health practitioners. Ir J Psychol Med; 40: 43–50.

[6] Pressman, R.S., Maxim, B.R. (2015) The Software Engineer's Responsibility. Software Engineering: A Practitioner's Approach, 865–867.

[7] Software development life cycle methods | by Chathurika Dhananjani | Medium. https://chathurikadhananjani97.medium.com/software-development-life-cycle-methods-712be36f6aae.

[8] DevOps Puppet. In a simple word a Puppet is a… | by Sidhant Suryavansham | Medium. https://sidhant-suryavansham.medium.com/devops-puppet-78400f04696b.

[9] Software Development Life Cycle | Benefits, Phases, Process & Models. https://www.weetechsolution.com/blog/software-development-life-cycle.

[10] Comparative Analysis of Software Development Life Cycle Models https://www.researchgate.net/publication/286134213_Comparative_Analysis_of_Software_Development_Life_Cycle_Models.

[11] Wang, Y.M., Elhag, T.M.S. (2006) Fuzzy TOPSIS method based on alpha level sets with an application to bridge risk assessment. Expert Syst Appl, 31, 309–319.

[12] Boehm, B.W. (1988) A Spiral Model of Software Development and Enhancement. Computer (Long Beach Calif), 21,: 61–72.

[13] McConnell, S. R. (2010) Development: Taming Wild Software Schedules. Microsoft Press, 6, 680.

[14] Saaty, T.L. (1990) How to make a decision: The analytic hierarchy process. Eur J Oper Res, 48, 9–26.

[15] Liberatore, M.J., Nydick, R.L. (2008). The analytic hierarchy process in medical and health care decision making: A literature review. Eur J Oper Res, 189, 194–207.

[16] Aniley, D.B., Jalew, E.A., Agegnehu, G.A. (2024) Selection of Software Development Life Cycle Models using Machine Learning Approach. Int J Comput Appl, 186, 36–43.

[17] MIH (2023) Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. IJFMR - International Journal For Multidisciplinary Research; 5.

[18] Brans, J.P., Vincke, P. (1985). Note—A Preference Ranking Organisation Method. Manage Sci, 31, 647–656.

[19] Islam, A.K.M.Z., Ferworn, DrA. (2020). A Comparison between Agile and Traditional Software Development Methodologies. Global Journal of Computer Science and Technology; 20: 7–42.

[20] Keskinkılıç, M. & Kahveci, F. (2019) Yazılım Mühendisliğinde Çevik Yöntemler Üzerine Kavramsal Bir İnceleme ve Sınıflandırma. Atatürk Üniversitesi Sosyal Bilimler Enstitüsü Dergisi, 23(3), 1067-1091.

[21] Herdika, H.R., Budiardjo, E.K. (2020) Variability and Commonality Requirement Specification on Agile Software Development: Scrum, XP, Lean, and Kanban. International Conference on Computer and Informatics Engineering, 323–329.

[22] Schwaber, K., Sutherland, J. (2020) The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game.

[23] Ahmad, M.O., Markkula, J., Oivo, M. (2013) Kanban in software development: A systematic literature review. Proceedings- 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA, 9–16.

[24] Beck K. Praise for Extreme Programming Explained, Second Edition.

[25] Lean Software Development: An Agile Toolkit: Poppendieck, Mary, Poppendieck, Tom: 0785342150780: Amazon.com: Books. https://www.amazon.com/Lean-Software-Development-Agile-Toolkit/dp/0321150783.

[26] Fowler, M., Beck, K. (2018) Refactoring : Improving the Design of Existing Code, Second Edition Fowler, Martin.

[27] Test Driven Development: By Example: Beck, Kent: 8601400403228: Amazon.com: Books. https://www.amazon.com/Test-Driven-Development-Kent-Beck/dp/0321146530.

[28] Williams, L., Kessler, R.R., (2003) Pair Programming Illuminated, 265.

[29] Continuous Integration: Improving Software Quality and Reducing Risk: Paul M. Duvall, Steve Matyas, Andrew Glover: 9780321336385: Amazon.com: Books. https://www.amazon.com/Continuous-Integration-Improving-Software-Reducing/dp/0321336380.

[30] Cohn, M. (2009). What is a User Story? User Stories Applied, 4.

[31] Yazılım Gereksinim Analizi | AppMaster. https://appmaster.io/tr/blog/yazilim-gereksinimleri-analizi.

[32] Mobil Uygulama Geliştirme Rehberi | Vayes. https://www.vayes.com.tr/tr/blog/mobil-uygulama-gelistirme-rehberi.

[33] Gereksinim yönetimi - Vikipedi. https://tr.wikipedia.org/wiki/Gereksinim_y%C3%B6netimi.

[34] Albrecht, A.J., Gaffney, J.E., (1983) Software Function, Source Lines of Code, and Development Effort Prediction. IEEE Transactions on Software Engineering, SE-9, 639–648.

[35] Milicic, D. (2004) Applying COCOMO II- A case study.

[36] Albakri, M.M., Rizwan, M., Qureshi, J. Empirical Estimation of COCOMO I and COCOMO II Using a Case Study.

[37] Yazılımcılar Dünyası: COCOMO (Constructive Costing Model). https://www.yazilimcilardunyasi.com/2017/01/cocomo-constructive-costing-model.html.

[38] Rush, C., Roy, R. (2023) Analysis of cost estimating processes used within a concurrent engineering environment throughout a product life cycle. Advances in Concurrent Engineering; 58–67.

[39] Baresi, L., Pezzè, M. (2006) An Introduction to Software Testing. Electron Notes Theor Comput Sci, 148, 89–111.

[40] Corral, L., Sillitti, A., Succi, G. (2013) Agile Software Development Processes for Mobile Systems: Accomplishment, Evidence and Evolution. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8093 LNCS: 90–106.

[41] Dobing, B., Parsons, J. (2006) How UML is used. Commun ACM, 49109–113.

[42] Faitelson, D., Tyszberowicz, S. (2017) UML Diagram Refinement (Focusing on Class-And Use Case Diagrams). Proceedings - International Conference on Software Engineering; 735–745.

[43] Jayasiriwardene, S., Meedeniya, D. (2021) Architectural framework for an interactive learning toolkit. Proceedings - International Research Conference on Smart Computing and Systems Engineering; 14–21.

[44] Li, S., Lei, Y., Jia, Z., Boukhlif, M., Hanine, M., Kharmoum, N. (2023) A Decade of Intelligent Software Testing Research: A Bibliometric Analysis. Electronics, 12.

[45] Umudova, S. (2019) Analysis of Software Maintenance Phases. Noble International Journal of Scientific Research; 3: 62–66.

[46] The Unified Modeling Language Reference Manual, (2nd Edition): Rumbaugh, James, Jacobson, Ivar, Booch, Grady: 9780321718952: Amazon.com: Books. https://www.amazon.com/Unified-Modeling-Language-Reference-paperback/dp/032171895X.

[47] Curtis, B., Krasner, H., Iscoe, N. (1988) A field study of the software design process for large systems. Commun ACM, 31, 1268–1287.

[48] Setyantoro A. Process Models in Software Engineering. https://www.academia.edu/36272460/Process_Models_in_Software_Engineering.

[49] Children's online activities, risks and safety: A literature review by the UKCCIS Evidence Group - GOV.UK. https://www.gov.uk/government/publications/childrens-online-activities-risks-and-safety-a-literature-review-by-the-ukccis-evidence-group.

[50] Secure Development Lifecycle for App Security - AppSOC. https://www.appsoc.com/blog/understanding-the-secure-development-lifecycle-sdlc-for-app-security.

[51] Lonergan, A., Moriarty A, McNicholas F, Byrne T, (2023) Cyberbullying and internet safety: a survey of child and adolescent mental health practitioners. Ir J Psychol Med; 40: 43–50.