

2018

The software development life cycle and its application

Gillian Lemke

Follow this and additional works at: <https://commons.emich.edu/honors>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Lemke, Gillian, "The software development life cycle and its application" (2018). *Senior Honors Theses & Projects*. 589.

<https://commons.emich.edu/honors/589>

This Open Access Senior Honors Thesis is brought to you for free and open access by the Honors College at DigitalCommons@EMU. It has been accepted for inclusion in Senior Honors Theses & Projects by an authorized administrator of DigitalCommons@EMU. For more information, please contact lib-ir@emich.edu.

The software development life cycle and its application

Abstract

The Software Development Life Cycle (SLDC) is a concept that is incredibly important to have a deep understanding of as a software engineer. With this project, my goal was to learn the complexities of each step conceptually and apply my skills to an actual application. The SDLC includes the following phases: planning and requirement analysis, design and development, implementation, testing, integration, and maintenance. In order to apply these concepts, I created a web application for users to schedule messages to be sent at a future time and date. The API is written in Ruby on Rails and the front end is written in ReactJS. There is also a PostgreSQL database that stores data such as saved messages and user information.

Degree Type

Open Access Senior Honors Thesis

Department

Computer Science

First Advisor

Dr. Krish Narayanan

Second Advisor

Dr. Augustine Ikeji

Keywords

Development, Software Engineering, Design, Agile, Extreme Programming, Computer Science

Subject Categories

Computer Sciences

THE SOFTWARE DEVELOPMENT LIFE CYCLE AND ITS APPLICATION

By

Gillian Lemke

A Senior Thesis Submitted to the

Eastern Michigan University

Honors College

in Partial Fulfillment of the Requirements for Graduation

with Honors in Computer Science

Approved at Ypsilanti, Michigan, on this date 8 May 2018

Supervising Instructor, Dr. Krish Narayanan

Honors Advisor, Dr. Krish Narayanan

Department Head, Dr. Augustine Ikeji

Honors Director

Table of Contents	2
Abstract	3
Keywords	3
1. Introduction	3
2. The Software Development Life Cycle	4
2.0 What is it?	4
2.1 The Phases	6
2.1.0 Initiation	6
2.1.1 System Concept Development	6
2.1.2 Planning	6
2.1.3 Requirements Analysis	7
2.1.4 Design	7
2.1.5 Development	7
2.1.6 Integration and Testing	8
2.1.7 Implementation	8
2.1.8 Operations and Maintenance	9
2.1.9 Disposition	9
2.2 Agile Development and Extreme Programming	9
2.2.0 Application of Agile and XP in this Application	11
3. My Application	11
3.0 Phases and their Results	11
3.0.0 Initiation	11
3.0.1 System Concept Development	12
3.0.2 Planning and Requirements Analysis	13
3.0.3 Design	14
3.0.4 Development	20
3.0.5 Integration and Testing	20
3.0.6 Repetition of Phases	21
3.0.7 Implementation	21
3.0.8 Operations and Maintenance	22
3.0.9 Disposition	22
3.1 Technologies Used	22
4. Conclusion	22
5. Future Work	23
6. Link to Source Code	24
Appendix A – Client Contract	25
Appendix B – Software Requirement Specification Document	26
Citations	31

Abstract

The Software Development Life Cycle (SDLC) is a concept that is incredibly important to have a deep understanding of as a software engineer. With this project, my goal was to learn the complexities of each step conceptually and apply my skills to an actual application. The SDLC includes the following phases: planning and requirement analysis, design and development, implementation, testing, integration, and maintenance. In order to apply these concepts, I created a web application for users to schedule messages to be sent at a future time and date. The API is written in Ruby on Rails and the front end is written in ReactJS. There is also a PostgreSQL database that stores data such as saved messages and user information.

Keywords

Software, Life Cycle, Application, Design, Development

1. Introduction

The Software Development Life Cycle (SDLC) has been widely adapted through most, if not all, technology based companies. With the advancement of technology, and the increasing number of companies that rely on their own custom applications, learning and understanding the complexities of the SDLC is increasingly important. Of course knowing how to code is the basis for any software engineering career, but understanding the SDLC brings a lot of new skill bases to the table, all of which are beneficial to the engineering process. These skills include planning, designing, and testing. The SDLC traditionally has ten phases which are initiation, system concept development, planning, requirements analysis, design, development, integration and testing, implementation, operations and maintenance, and disposition [2]. Despite being distinct phases, they are

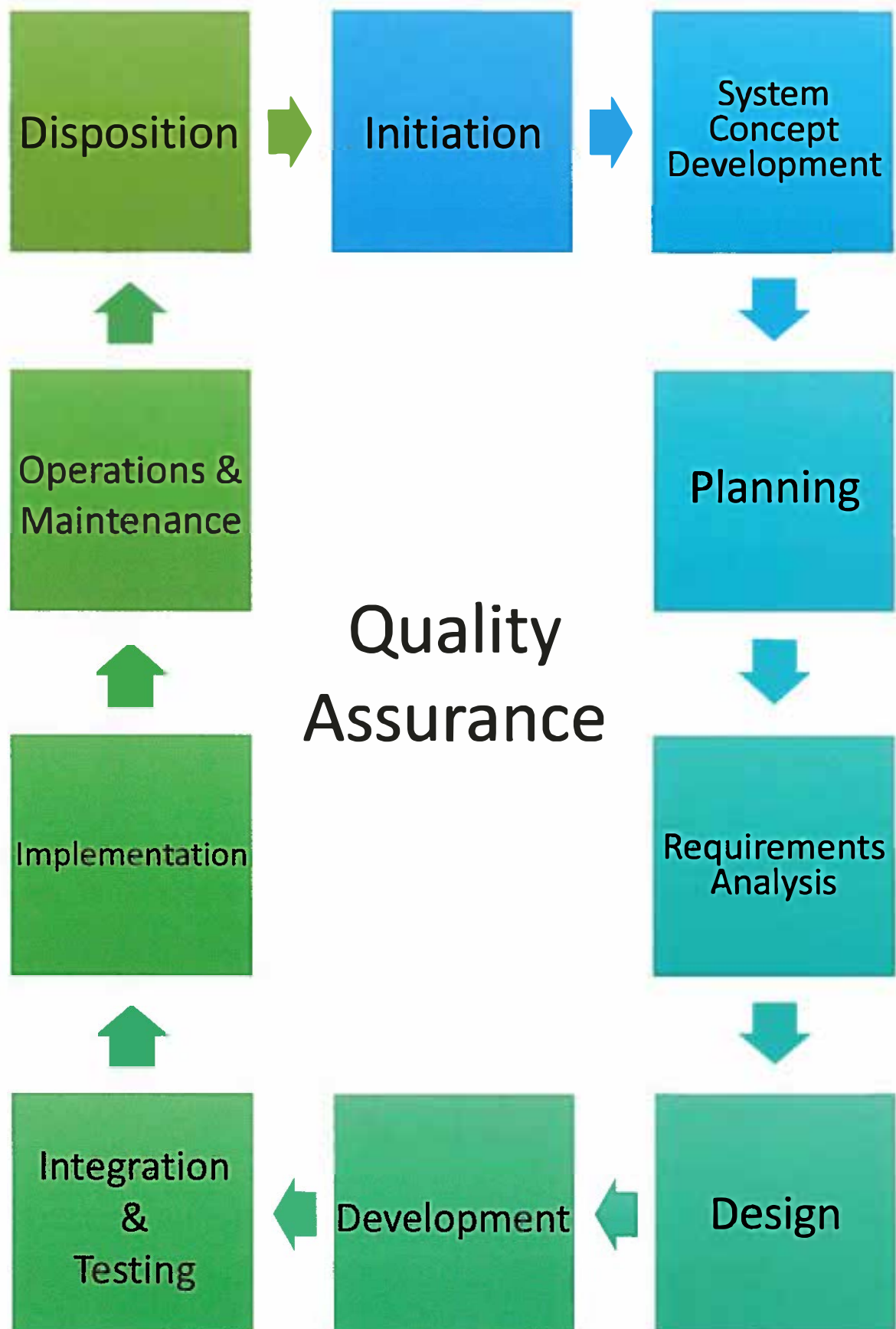
not all required in every situation and often are either skipped or combined together [2]. These phases are all addressed in more detail later. Additionally, variations on the SDLC, including increasingly important aspects devoted to security in software engineering, are addressed as well [reference].

Created and illustrated here is a web application made with the intention of learning and critiquing the phases demonstrated in the SDLC. The application serves the purpose to allow users to schedule messages to be sent to any person. It is written in ReactJS and Ruby on Rails, with a PostgreSQL database. Due to the importance of a client in the SDLC, a fellow student, studying similar concepts, acted as the client for this project. The intricacies of the application and important pieces of code are reviewed later.

2. The Software Development Life Cycle

2.0 What is it?

The Software Development Life Cycle is a basis or framework for structuring, planning, and executing tasks involved with developing an information system [5]. Due to the difference in requirements of systems along with numerous other differences, variations on the SDLC have been developed. Before beginning a project, it is important to determine which of these is the most suitable, especially with the recent advancement of web-based technologies [5]. The SDLC includes distinct phases designed to give software engineers a clearly defined goal for how to work [2]. It aims to aid developers and other project staff to create a system that meets all technical and user requirements as well as exceeds customer expectations [2]. These phases all revolve around a central theme of quality assurance. The major priority of the project team is to provide quality software to the users and client. A visual of this cycle is demonstrated below.



2.1 The Phases

Much of the information described here comes from resource [2].

2.1.0 Initiation

The Initiation phases identifies a problem to be solved through software engineering. A project sponsor must be chosen; this is a person of authority over the project and is in charge of mediating issues of scope and determining functional requirements. It is also stressed that this process be documented and a concept proposal is expected to be a deliverable. This document illustrates the needs defined in business terms and the overall mission of the system. The Concept Proposal is made by the program manager or sponsor and is presented to the executive board for approval.

2.1.1 System Concept Development

System Concept Development is used to define the scope of the system. This may include a high-level schedule, cost summary, and other plans that may be required for the specific type of system, usually determined by all members of the project team. Documents made during this phase may include the System Boundary Document, Cost Benefit Analysis, Feasibility Study, and Risk Management Plan. These documents are typically presented to stakeholders for review and approval.

2.1.2 Planning

Most of the documents needed during the other phases of the SDLC are made here. These may include an Acquisition Plan, Concept of Operations, and Project Management Plan, but there are many more that may be relevant to different projects. A summary is usually presented to stakeholders at this point, and a detailed plan is presented and reviewed.

2.1.3 Requirements Analysis

The Requirements Analysis phase starts after the documents prepared in the previous phases are reviewed and approved. Any document stating user needs in the system will serve as the basis for document created in this phase that illustrate user requirements and system requirements. Details such as system inputs, interfaces, and processes are described here. Typically, a Software Requirement Specification document or Functional Requirements document is created during this phase which includes information about how to meet user requirements. This may address issues with users with disabilities such as color-blindness, hearing loss, or cognitive disabilities, or mobility impairments. A Test Plan may also be created during this phase and may include details on unit test, integration test, and/or system tests. The portion of documentation is often created by the development team with the help of a Quality Assurance (QA) team whose purpose is to make sure different types of users can easily use the system.

2.1.4 Design

In the Design phase, detailed requirements are to be turned into detailed specifications that engineers will use during the Development phase. These specifications should address how functional, physical, interface, and data requirements are to be met in the system. This is usually done iteratively through the entire life cycle process. Designs may include database designs and user interface designs.

2.1.5 Development

The purpose of the Development phase is to convert designs from the Design phase into a functional system. This not only includes the software that should be written, but also the infrastructure that should be set in place. These infrastructures may include hardware,

software, and communication systems that are required for the functionality of the overall system. Besides the code that is written, some deliverables expected as a result of this phase include a Contingency Plan that directs the client what to do in case of emergency; a Software Development document which illustrates test cases and results, how the components work, and approvals; the test files and data; and an Integration Document which illustrates how the software and hardware components work together.

2.1.6 Integration and Testing

The Integration and Testing phase aims to determine if the requirements specified in the specifications document are met. Three different types of testing are ideal to include in this phase: integration testing of subsystems, security testing, user testing or acceptance testing, and unit testing. Each of these types of tests serve a different purpose and help all stakeholders determine flaws in the system prior to deployment. Unit testing, which tests small portions of code and is traditionally done during the development phase. Several analysis reports may be produced from these tests and should be presented to stakeholders before moving on to the implementation phase. During this meeting, details on how implementation should take place are imperative to address.

2.1.7 Implementation

During this phase, the system is deployed to the production environment. It is important to inform users of the potential system changes prior to deployment and to train any professionals on the new system if required. A post-implementation review is commonly completed to make sure all requirements are met and the system is functioning in production as expected. If everything is in order, a contract between the development

team and the client project manager is signed, stating that the system has been completed and delivered [2].

2.1.8 Operations and Maintenance

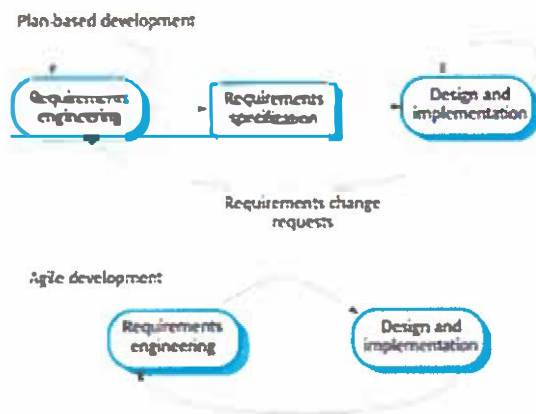
More than half the cost of the development life cycle is due to the Operations and Maintenance phase. This phase may include identifying system operations, maintaining data, identifying problems, and revising documentation that received additional analysis (e.g. security audit/analysis). A User-Satisfaction report may be developed during this phase to identify and potential user end problems to address in future development iterations.

2.1.9 Disposition

This phase is done only when a system has become obsolete or been replaced by another. Prior to shutting down the system, it is necessary to make sure all documentation and resources are in order. A plan for shutdown should be written and followed strictly, making sure to backup all resources and data from the system. It is also recommended that a security and contingency plan be made and stored with the system in case it has to be setup again.

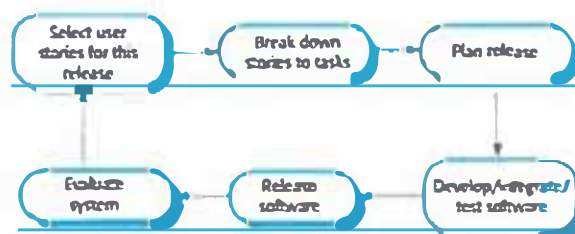
2.2 Agile Development and Extreme Programming

Agile development is an iterative process that aims to increase the speed of development by working incrementally. Agile development techniques include customer involvement, incremental delivery, flexibility for developers to make their own development plan, being open to constant change, and keeping code and overall system simple for future expansion [6], [7]. A comparison of agile and the more traditional plan-driven development is illustrated below [6].



Extreme Programming (XP) is an extreme approach to iterative development and stresses the importance of delivering small increments of the system (sometimes multiple increments each day) and testing for each build of the system increments. Some of the techniques used in extreme programming include incremental planning, small releases, simple design, test driven development, refactoring, pair programming, collective ownership, continuous integration, sustainable pace, and having an on-site customer. XP techniques are commonly used in agile development life cycles.

Below is a diagram illustrating a simple example of extreme programming [6].



Agile development and extreme programming both focus on decreasing delivery time and therefore, often lack the amount of documentation seen in other types of development such as waterfall or plan-based development [6].

2.2.0 Application of Agile and XP in this Application

Working in small increments, test driven development, and refactoring were all used during the construction of the application in this case.

3. REO: A Message Scheduling Application

3.0 Phases and their Results

3.0.0 Initiation

The purpose of the application, REO (short for reoccur), is illustrated below.

With the constant distractions of today's society, reminders are important.

These reminders may be for tasks that need to be completed, to laugh, or to wish someone a happy holiday. With the application "REO", short for reoccur, this is all possible. Users of the web-based application are able to schedule messages to be sent to themselves, a family member, or a project group. Either emails or text messages, users of REO can get helpful reminders to complete tasks, send their significant other funny images every day, or use for business purposes to make sure meeting attendees are present. The uses for REO are endless and there is a significant amount of potential for expansion.

The Concept Proposal in my case was created by myself and presented to the client for approval. The agreement is slightly different from the one illustrated by the DOJ [2] but includes relevant information and signatures confirming the initiation of this project.

Appendix A shows a copy of the agreement. A physical copy with signatures was also signed on the same date. Below is the agreement text.

“The client agrees to sponsor this system development and work with the developer to determine user requirements and technological restrictions and requirements.

The project manager agrees to oversee this project and work with the client project manager/sponsor to determine user requirements and technological restrictions and requirements.

Both project manager and client agree to meet every other week after project initiation to discuss details and progress on development.”

3.0.1 System Concept Development

Since this project had no cost (besides my own time), no Cost-Benefit Analysis was done. However, much of the other scope planning was done through the proposal submitted for a scholarship to do this project. This was done prior to the beginning of the semester. A high-level schedule was created to mark intended milestones. The milestones indicated below, are the components to be completed during each week and (in bold) each sprint.

Schedule

Jan 8 - Start of project

Jan 15 - planning

Jan 22 - project setup and login page

Feb 5 - home page

Feb 19 - new message page

March 5 - scheduled messages page/component

Not every week has a deliverable because some components may take longer than others, or additional phases needed be done during that time. This schedule was made in consultation with the client.

3.0.2 Planning and Requirements Analysis

The Planning and Requirements Analysis phases, in this case, were combined into one phase. Traditionally, there are many planning documents that need to be made and reviewed, especially for large systems with large teams. However, in this case, with the size of the development team and having only one person represent the client, these documents were not required. Instead, the planning took place as more of a conversation and the results were compiled into the Software Requirements Specifications document, traditionally created during the Requirements Analysis phase. This document specifies the purpose of the application, descriptions of user actions and components, interface requirements, and nontechnical requirements.

The document illustrates the high-level components to be included in the application, also specifying which components are of lower priority and are to be addressed in subsequent installments of development. Below is a list of the high-level components.

- Login page
- Sign-Up Page
- Dashboard/Homepage
- New Message Page

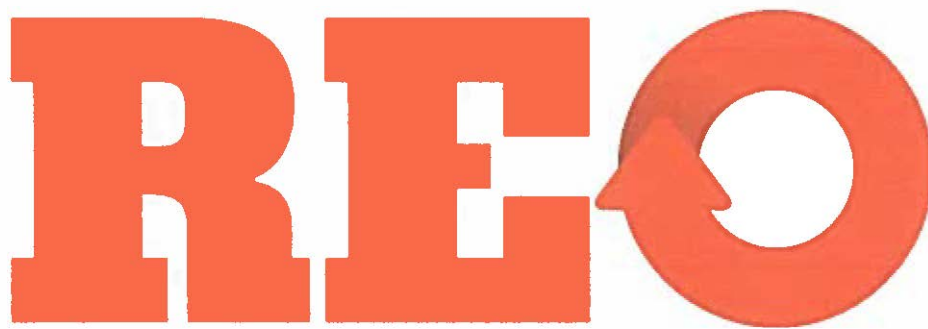
The full document, which shows the subcomponents of the pages listed above, can be referenced in Appendix B.

3.0.3 Design

The Design phase is often repeated through the development process when new designs are needed that were not originally anticipated or need to be edited to accommodate new or unforeseen circumstances. Most of the sprints of this project involved some edits or additions to designs.

Below are the user interface designs followed by the database designs for the system including the logo, signup and login page, home page and new message page. Links included in the designs including 'Settings', 'Friends', and 'REO Gold' are features to be implemented in future iterations of the project.

Logo

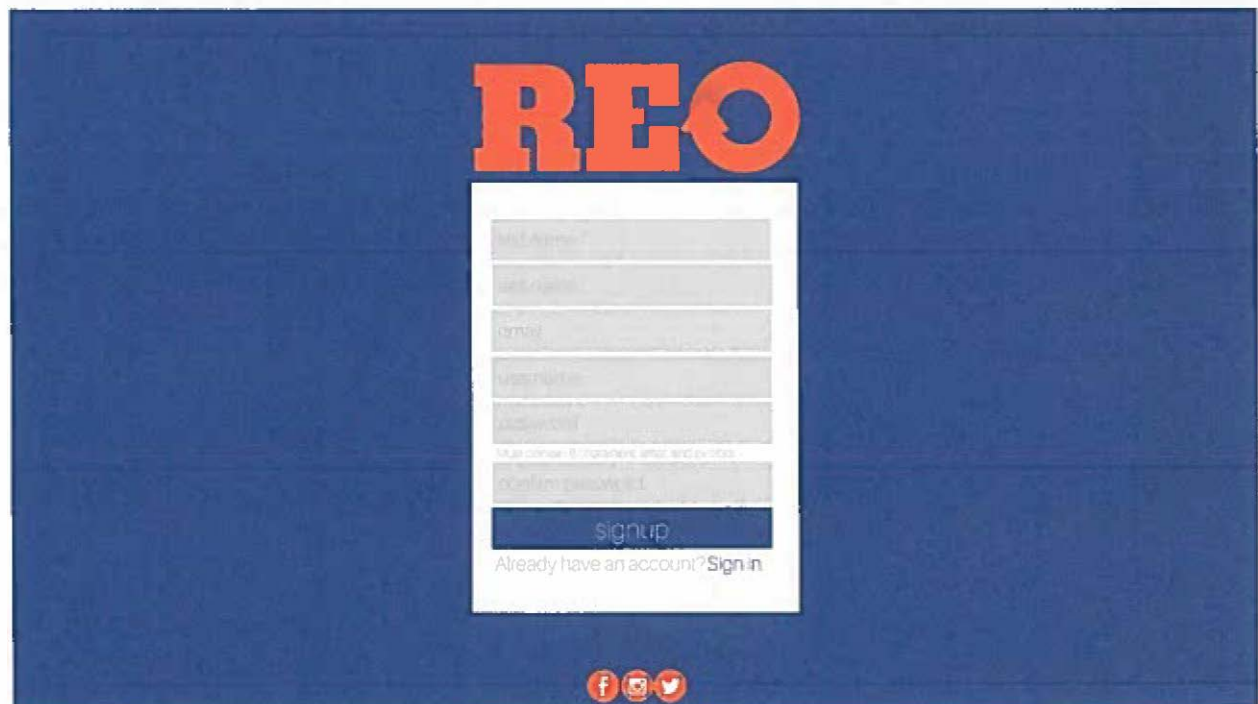


Login Page



The login page features a dark blue background with the 'REO' logo in large, bold, orange letters at the top center. Below the logo is a white rectangular form containing two input fields for 'Email or phone' and 'Password'. A blue 'login' button is positioned below the password field. Underneath the button, the text 'Not registered? [Signup here](#)' is displayed. At the bottom center of the page, there are three small, circular social media icons for Facebook, Instagram, and Twitter.

Sign-Up Page



The sign-up page has a dark blue background with the 'REO' logo in large, bold, orange letters at the top center. Below the logo is a white rectangular form with several input fields: 'First Name', 'Last Name', 'Email', 'Work phone', 'Address', and 'City'. A note below the address field states 'Must contain 8+ characters, upper and lowercase'. Below this is a 'confirm password' field. A blue 'signup' button is located at the bottom of the form. Below the button, the text 'Already have an account? [Sign in](#)' is shown. At the bottom center of the page, there are three small, circular social media icons for Facebook, Instagram, and Twitter.

Dashboard/Hompage



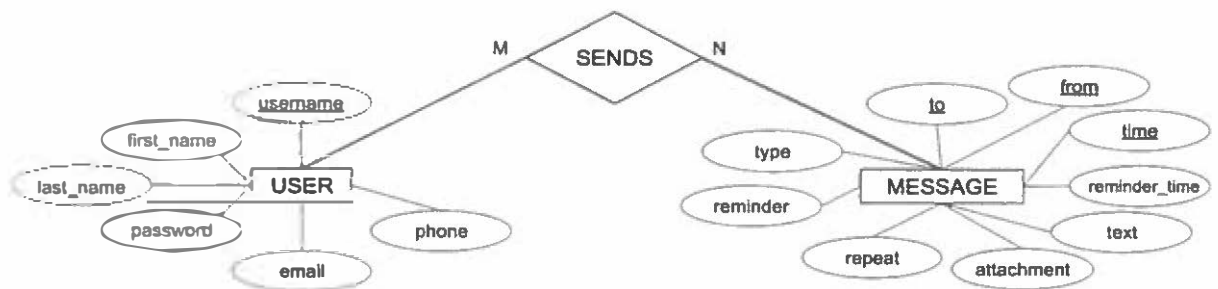
New Message Page



Database Design

Below is the Entity Relation (ER) diagram for the database system of the application.

In this case, there are only two tables, Users and Messages. Users have a unique username, password, first_name, last_name, email, and phone number. Messages have unique combination of to_whom [the message is sent], from_whom [the message is sent], and time. Messages also have a reminder_time, type (email or text), reminder (true or false), repeat, attachment, and text. The only relationship is that Users *send* Messages. This information is all represented in the ER diagram below.



A design of the database showing similar information to the above ER diagram is below.

It indicates the types for each attribute of the entities.

USER					
<u>username</u>	password	first_name	last_name	email	phone
varchar(20)	password	varchar(20)	varchar(20)	varchar(50)	char(10)

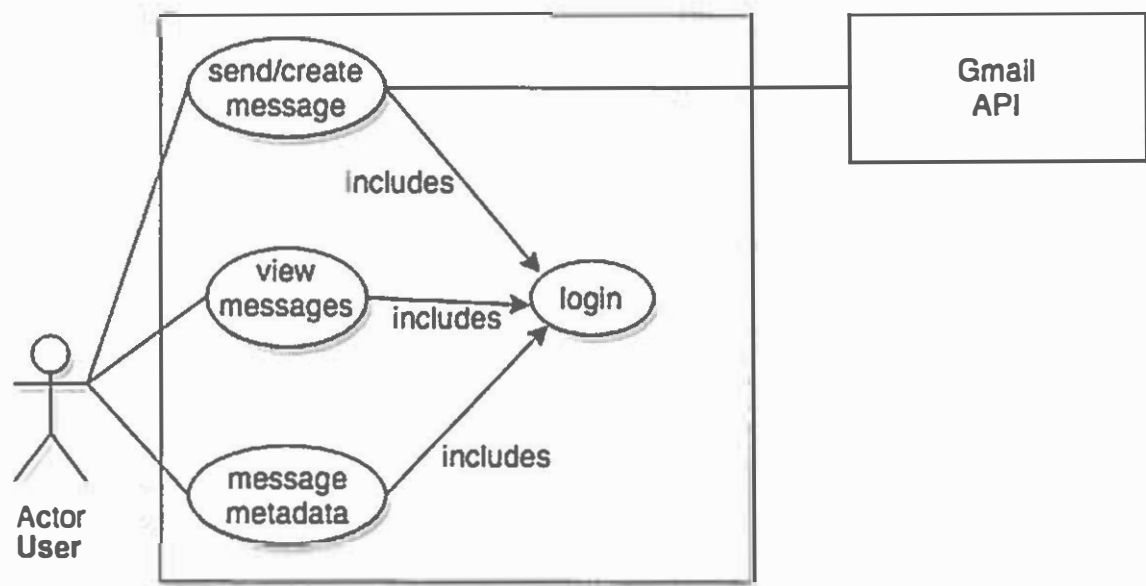
MESSAGE				
<u>to</u>	<u>from</u>	<u>time</u>	reminder_time	text
varchar(50)	varchar(20)	DateTime	DateTime	varchar(500)

	foreign key => USER.username		
--	------------------------------	--	--

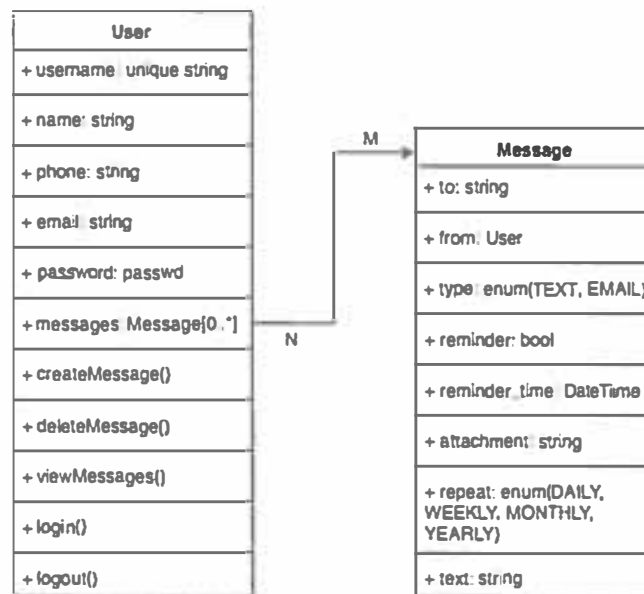
attachment repeat		reminder	type
file	enum(daily, weekly, monthly, yearly)	DateTime	enum(email, text)

Some additional UML diagrams developed during the design phase are included below.

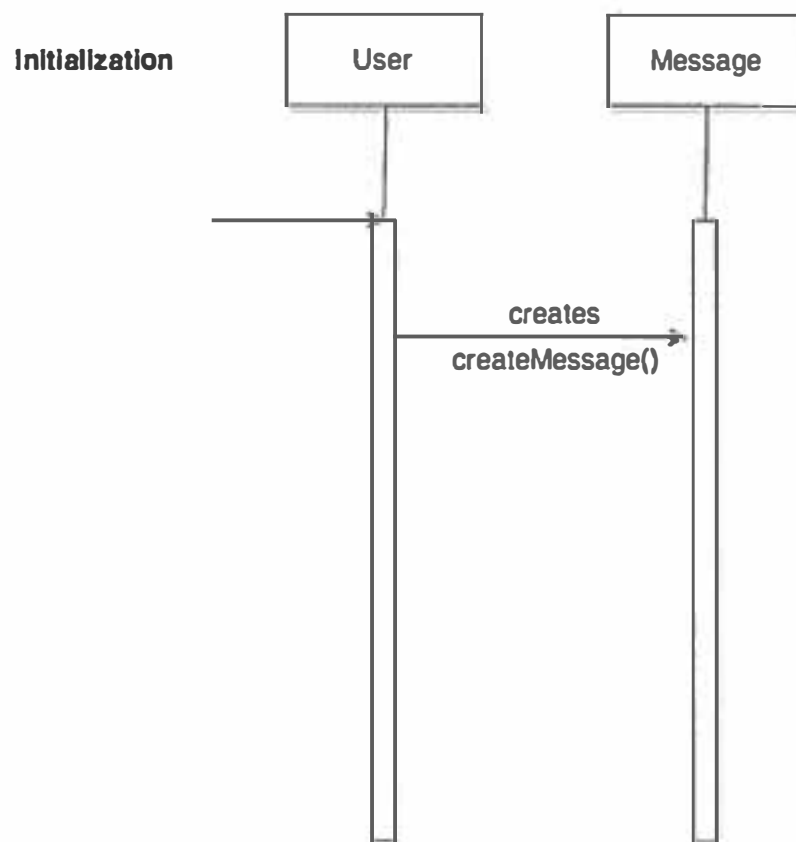
UML Use Case Diagram



UML Class Diagram



UML Interaction Diagram



3.0.4 Development

During this phase, the database design is turned into a database system. The database system in this case used PostgreSQL. Additionally, the user interface designs were converted into ReactJS components. Components in ReactJS are small parts of pages. For example, the header for each page is a component that can be used repeatedly on each page. This is convenient because there is a lot less repeat code than other frameworks or basic HTML. API endpoints, from where data can be collected, were created in the Rails portion of the application. The server listens for endpoint strings that match the API endpoints from Redux which calls the API. On a successful match, the API sends back the specified data. An example of this might be `'api/{user}/messages'` where user represents a variable (i.e. the user that is requesting the data). This API endpoint would return all the scheduled messages that the specified user has scheduled. Some of the development phase usually involves testing, primarily unit testing. In this case, test driven development was used throughout the development phase. Test driven development (TDD) is where the test is written before the functionality and the functions are written to make the tests pass. This is an Extreme Programming technique used frequently in Agile development. TDD is beneficial for many reasons including easier refactoring.

3.0.5 Integration and Testing

Much of the testing of the system was done during the development phase due to TDD. Since the developer team was only made up of one person and the client understands the easy-to-read tests, creating documentation of these tests and their results did not seem

necessary. Especially considering much of this life cycle was similar to agile methods, lack of documentation was not a problem.

3.0.6 Repetition of Phases

Similar to how Agile development functions, several phases during this life cycle process were repeated. Design, Development, and Testing phases were repeated for most of the sprints. Also common to Agile, is the use of sprints which is the name of short periods of time in which developers are to “sprint” to complete features. Much of the life cycle process was performed iteratively. Agile development stresses the importance of client communication and feedback which constitutes sprint review meetings: meetings held after each sprint with the client to discuss what has been done and what is to be completed next. I met with my client after every two-week sprint. Since the majority of the project was completed in accordance with the previously planned schedule, there was never much concern from the client. He did provide some design critiques however, which were implemented into the system.

3.0.7 Implementation

This application was not far enough along in the process to be deployed to a production server but was set up using Heroku in a development environment which could be used in the future to test production setup. Heroku is an easy-to-use deployment service based on Amazon Web Services and supports a wide variety of languages including ruby which was used here.

3.0.8 Operations and Maintenance

Due to the short time span dedicated to doing this project, this phase was not possible. As the project continues to be developed, maintenance will become increasingly important especially as exposure and the number of users increases.

3.0.9 Disposition

Since the system has not become obsolete, the Disposition phase was not required.

However, if the system is replaced or shutdown in the future, following the guidelines for the Disposition phase will become pertinent.

3.1 Technologies Used

This application is made using ReactJS, a Javascript framework, on the front-end (user interface), Redux to manage the state of the application, and Ruby on Rails which acts as the Application Programming Interface (API) and directly communicates with the PostgreSQL database system.

4. Conclusion

The Software Development Life Cycle (SDLC) is widely used in technology based companies and variations such as agile development with extreme programming have become increasingly popular as the importance of clean code and fast delivery rises. I have researched the ten phases of the SDLC and applied the techniques to creating my own web-based application. The application created is a message scheduler that aims to help people remember tasks, send reminders to others, or send encouraging messages to friends. The application uses ReactJS to create the user interface which communicates

through Redux, the state manager, to Ruby on Rails which is the application programming interface (API) and directly queries the PostgreSQL database.

5. Future Work

Security is also a topic that is becoming increasingly important when discussing development especially with the advancement of web technologies [3], [8]. Due to this innovation, including security in the phases of the SDLC is essential.

The Secure SDLC is a variation of the SDLC that stresses the importance of security [8]. Most security defects are due to known software defects [8]. Multiple frameworks have been made for detecting and fixing these issues before a product makes it to production. However, these are difficult to adapt into already functioning software development life cycles. Some tools used to address security throughout the software development life cycle include threat models, penetration testing, and ratings.

Difficulties in security are most commonly seen or are developed during the design phase [4] [1]. Creating security models including threat models, risk estimation, and identification of security goals [4] can be done during this phase to ensure more secure software.

The difficulties with using these new methods, especially in an agile based life cycle, is that it often consumes large costs, both time and money. Additionally, if developers are unfamiliar with security or the concepts recommended to make their code more secure, they are unlikely to implement the changes to the life cycle. One research study recommends training individuals on these techniques [4] which may solve the problem of knowledge but would increase costs.

Despite these cost issues, implementing security into the software life cycle is incredibly important. With the advancement of technology and the reliability users expect systems to have, any vulnerability puts the company at risk. With some initial training, developers would be able to have a better basis for how to fix problems they already know exist in their code base(s). Though there is a great time and financial cost for this training, it will outweigh the consequential risks of not putting security as a top priority during the development life cycle.

For this project, implementing these techniques in future iterations is a high priority, especially as deploying to production draws nearer.

6. Link to Source Code

<https://github.com/GillianLemke/MessageScheduler>

Appendix A – Client Contract**Concept Proposal and System Development Agreement for REO****Stakeholders**

Project Manager: Gillian Lemke

Client Project Manager/Sponsor: Jacob Rickerd

Description of System

With the constant distractions of today's society, reminders are important. These reminders may be for tasks the need to be completed, to laugh, or to wish someone a happy holiday. With the application "REO", short for reoccur, this is all possible. Users of the web-based application are able to schedule messages to be sent to themselves, a family member, or a project group. Either emails or text messages, users of REO can get helpful reminders to complete tasks, send their significant other funny images everyday, or use for business purposes to make sure meeting attendees are present. The uses for REO are endless and there is a significant amount of potential for expansion.

Agreement to Complete System and Collectively Determine Requirements

The client agrees to sponsor this system development and work with the developer to determine user requirements and technological restrictions and requirements.

The project manager agrees to oversee this project and work with the client project manager/sponsor to determine user requirements and technological restrictions and requirements.

Both project manager and client agree to meet every other week after project initiation to discuss details and progress on development

Client Signature

January 2, 2018
Date

Developer Signature

January 2, 2018
Date

Appendix B – Software Requirement Specification Document* *****Page numbers changed here for consistency with rest of document****** Original document contains database designs****Table of Contents**

Table of Contents	25
1. Introduction.....	26
1.1 Purpose.....	26
1.2 Document Conventions.....	26
1.3 Intended Audience and Reading Suggestions.....	26
1.4 Product Scope	26
1.5 References.....	27
2. Overall Description	27
2.1 Product Perspective.....	27
2.2 Product Functions	27
2.3 User Classes and Characteristics	27
2.4 Operating Environment.....	28
2.5 User Documentation	28
2.6 Assumptions and Dependencies	28
3. External Interface Requirements	28
3.1 User Interfaces	28
3.2 Hardware Interfaces	28
3.3 Software Interfaces	29
3.4 Communications Interfaces	29
4. Other Nonfunctional Requirements	29
4.1 Performance Requirements.....	29
4.2 Security Requirements	29
4.3 Software Quality Attributes	29

1. Introduction

1.1 Purpose

This web application is a service for scheduling messages, either email or text messages. Many audiences will be able to use this product. For example, teachers can use it to send reminders to students, students can send themselves reminders, parents can send their children chore lists or daily to dos, friends can send each other funny messages or pictures, significant others can send anniversary messages, family can wish each other happy holidays or birthdays, and many other situations. The service this application provides is relevant to all people of different professions, ages, and purposes. This is the first installment of the application and when it is released, will be version 1.0. The scope of this installment will cover basic login and authentication, a homepage/dashboard where users can view scheduled messages and can create new ones, a settings page. Version 1.0 will only allow for emails to be sent. Other features may be added as we come across their need, however, for the first release, we will try to keep it simple so we can deploy quickly.

1.2 Document Conventions

The basic requirements are illustrated in section 1.1. Those requirements are for the first installment of development and the application's first release. After that release, we will illustrate additional requirements. These additional requirements that could be developed during this installment but have a lower priority include adding Single Sign-On (SSO) through Google, uploading contacts to the application to more easily send messages, sending text messages, inviting friends to join the platform, and others. Sending text messages will be available only through a "gold" account. This upgrade will cost a set amount per month for unlimited text messages and a "silver" account will cost a small fee for each text message sent. The feature allowing users to invite friends, will give the user a week of free "gold" membership. Users will be able to sync their account with Facebook to invite friends that way. They can also invite friends through email. "Gold" and "silver" membership will be pushed to the next installment of the project, while SSO and other small features will be of low priority for the current installment.

1.3 Intended Audience and Reading Suggestions

This document is for the client project leader, software engineers, designers, security engineers, and testers. The rest of this document includes project scope, general descriptions, and interface requirements. It is recommended that project manager(s) read only the general descriptions while all engineers should read through the entirety of the document, especially the requirements sections.

1.4 Product Scope

The goal of this service is to simplify lives through use of reminders, make a fun environment for friends, and serve as a calendar for groups or one's self.

1.5 References

The Pivotal Tracker chart will serve as the developer's tool for keeping track of completed features. This will be the basis for sprint reviews and weekly updates on progress. This is also the place where any relevant development materials will be linked. For specifics on stories/chores/bugs, review the project on Pivotal Tracker.

Pivotal Tracker project: created and maintained by Gillian Lemke – tech lead

2. Overall Description

2.1 Product Perspective

This is a self-contained product. This specific release is the beginning of a large system with additional features partially illustrated in section 1.2.

2.2 Product Functions

*High level components include (items with * indicate low priority):*

- *Login page*
- *Signup page*
- *Dashboard/homepage*
- *New message page*
- *Settings page*
- *Signup/login with Google**

2.3 User Classes and Characteristics

*Components with characteristics include (items with * indicate low priority):*

- *Login page*
 - *Username*
 - *Password*
 - *Authentication*
 - *Login with Google**
- *Sign-up page*
 - *Username (unique)*
 - *Name*
 - *Password*
 - *Phone number*
 - *Signup with Google**
- *Dashboard/homepage*
 - *Preview of scheduled messages*
 - *Friends online (available if linked with Facebook)**
 - *Preview of received messages (email only)**
 - *Signup for membership**
- *New message page*
 - *Email option*
 - *Message to send*

- *Contact email to send to*
 - *Attach document/image**
 - *Search Google images/gifs**
 - *Text option**
 - *Select contact from contact book**
 - *Send to Facebook friend**
- *Settings page*
 - *Change password*
 - *Change email*
 - *Add/change avatar*
 - *Ability to be searched**
 - *Setup two-factor authentication**
 - *Block user**
 - *Change language**

2.4 Operating Environment

This is a web application that will be responsive for use on a mobile or tablet view but will primarily be used for desktop. It must be compatible with all web browsers including Google Chrome, Safari, Internet Explorer, and Mozilla Firefox. A low priority for this installment is that it will function with Google SSO. It must function on all operating systems and will be developed using MacOS X and will be tested on Ubuntu through continuous integration.

2.5 User Documentation

The first installment of the application will not include a user manual or online help option.

2.6 Assumptions and Dependencies

We assume that the technologies selected by the developers are reliable. If the feature involving Google SSO is reached during this installment, we assume the dependence is reliable.

3. External Interface Requirements

3.1 User Interfaces

Designs for UI components are not yet completed. The components will be built using ReactJS and will communicate with the API through Redux. The use of Material UI is still under consideration.

3.2 Hardware Interfaces

The application should be supported on all web platforms.

3.3 Software Interfaces

The system will function starting from the lowest level on a PostgreSQL database. The API will be built using Ruby on Rails and will communicate with the user interface, which is built using ReactJS, through Redux, a state management framework. Webpack will package the project before deploying to the server and will serve a single JavaScript file to users. The application will be deployed on Heroku or Amazon Web Services and will use TravisCI for continuous integration with a connection through GitHub.

3.4 Communications Interfaces

Emails will be sent from users using Google's SMTP Server. This will ensure proper encryption of messages, something that may be more difficult to guarantee if company server were to be used. The application itself will be served through HTTPS and will use TLS for proper security.

4. Other Nonfunctional Requirements

4.1 Performance Requirements

Since the project has not yet been deployed, performance requirements are not yet known. As the iterations are released, requirements may be added to this section and may include speed of queries for many users, speed of the application's server(s), and speed of the SMTP server.

4.2 Security Requirements

Security upon sign-in is incredibly important. The developers are in charge of determining the best technologies to use for user authentication and database encryption. Additionally, messages being sent and created must be kept secure. Since Google's SMTP server will be used for this, those security concerns are low. The application, as previously mentioned, must be run over HTTPS and use TLS. See Section 3.4 for more on secure communications.

4.3 Software Quality Attributes

The application must be available to users from across the globe. Since the application is being developed iteratively, adaptability and flexibility are important. There are many additional features that will be added in other versions so keeping adaptability in mind while developing is imperative for the application's success. In order to ensure usability, QA testing is advised before the deployment of the final iteration. This QA testing will make sure the application is easy to use by persons of any technical background. Additional unit, integration, and system testing is expected to be done during development to ensure clean and reliable code.

Citations

1. Dasanayake, Sandun, Jouni Markkula, and Markku Oivo. "Concerns in software development: a systematic mapping study." *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2014.
2. The Department of Justice Systems Development Life Cycle Guidance Document. (2003) US Department of Justice.
3. Pistoia, Marco, and Omer Tripp. "Integrating Security, Analytics and Application Management into the Mobile Development Lifecycle." *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle*. ACM, 2014.
4. Rindell, Kalle, Sami Hyrynsalmi, and Ville Leppänen. "Busting a myth: Review of agile security engineering methods." *Proceedings of the 12th International Conference on Availability, Reliability and Security*. ACM, 2017.
5. Selecting a development approach. (February 2005). Center for Medicare and Medicaid Services.
6. Sommerville, I. (2016). *Software engineering*.
7. What is Agile Software Development Life Cycle? (March, 2016). QuickScrum.
8. Yuan, Xiaohong, et al. "Retrieving relevant CAPEC attack patterns for secure software development." *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. ACM, 2014.