Algorithms Visualiser in ReactJS

Final Year Project
B.Sc.(Hons) in Software Development

BY
Kevin Niland

April 3, 2020

Advised by Dr. Martin Kenirons
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract This project is intended to be an project that can be used in an educational environment. During the Data Structures and Algorithms module we had in 2nd year, short videos were shown to us showing various sorting algorithms being visualized. It is a web-based application written in ReactJS, a web framework for building user interfaces. The project allows users to visualize various sorting algorithms, which could be helpful in an educational context.

This project also allows users to register and sign in to an account. From here, the user can record their screen and upload these sorts to a database. These sorts are available to be viewed by all users once logged in.

Chapter 1

Introduction

This chapter serves as an introduction to the project. In it, the various aspects of the project are discussed and the main objectives of the project. Each of the chapters are briefly discussed as well, detailing what each contains and what is discussed in each.

1.1 Discussion of the dissertation

Throughout this dissertation, the following six chapters each covered a distinct aspect of the project:

- **Chapter 1 - Introduction** - Chapter 1 serves as the introduction to the project, discussing the aim of the project, an overview of sorting algorithms, the project itself, the objectives of the project, and the scope of the project.
- **Chapter 2 - Methodology** - Chapter 2 covers the various research and software methodologies employed throughout, meetings with the project supervisor. development tools used, source control, and types of testing carried out.
- **Chapter 3 - Technology Review** - Chapter 3 introduces each of the different technologies, tools, and languages used to develop the project itself.
- **Chapter 4 - System Design** - Chapter 4 discusses the design of the system and goes over each of the main components of the application.
- **Chapter 5 - System Evaluation** - Chapter 5 evaluates the system as a whole, analysing the various aspects of the project and further discusses the type of testing carried out.
- **Chapter 6** - Chapter 6 serves as the conclusion to the project, evaluating the project further relative to initial objectives set out and testing.

1.2 Aim of the project

The aim of this project is to create a web-based application that allows users to visualise various different sorting algorithms, allow user authentication and upload visualisations to a database from which other users can view and compare the performance. This project is intended to be used in an educational sense, providing an interactive application where user's can actively choose a sorting algorithm to visualise, save these visualisations to view them at a later point.

1.3 Sorting Algorithms

In computer science, a sorting algorithm is an algorithm that rearranges elements of a list in a certain order, according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure. The most frequently used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the efficiency of other algorithms (such as search and merge algorithms) that require input data to be in sorted lists. Sorting is also often useful for canonicalizing data and for producing human-readable output. More formally, the output of any sorting algorithm must satisfy two conditions:

1. The output is in non-decreasing order (each element is no smaller than the previous element according to the desired total order)

2. The output is a permutation (a reordering, yet retaining all of the original elements) of the input

Furthermore, the input data is often stored in an array, which allows random access, rather than a list, which only allows sequential access; though many algorithms can be applied to either type of data after suitable modification.

1.4 Discussion of the project

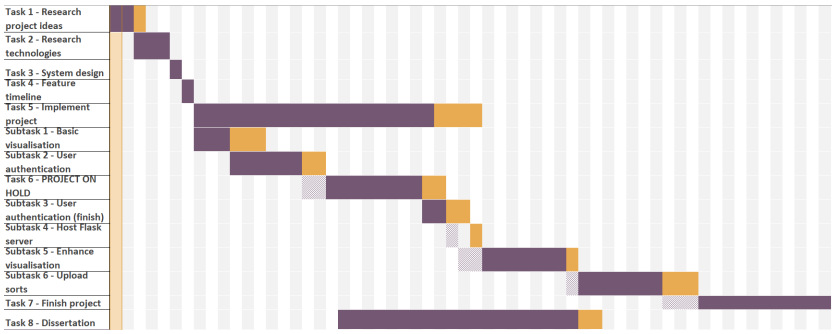
The project is sorting algorithm visualiser that allows users to choose a sorting algorithm to visualise. It also supports user authentication, where users can register and sign-in to save previous sorts. The project came about from viewing similar projects online and and an interest in finding out how such projects were developed. The project also came about from developing something that could be used in an educational sense. In 2nd year of our course, we were enrolled in a Data Structures and Algorithms module from which we learned about various different algorithms, how they were performed, space time complexity, etc.

1.5 Objectives of the project

There were several objectives I set out when first designing this project:

- Create an application that can be used in an educational sense, which can possibly be used for modules where students learn about data structures, algorithms, space time complexity, etc.
- Create a web application that allows users to visualise several different sorting algorithms.
- Allow users to register for an account and log in to save visualisations for later viewing.
- Allow users to upload their own 'data sets' to sort.

1.6 Scope of the project



Fill out...

1.6.1 Project Requirements

SORTING VISUALISER PROJECT TIMELINE

AT RISK	TASK NAME	DESCRIPTION	STATUS	ASSIGNED TO	START DATE	END DATE	DURATION
☐	Task 1 - Research project ideas	Research project ideas, with focus on project viability and scope	Complete	Kevin	09/08	09/25	17
☐	Task 2 - Research technologies	Research technologies that can be used to develop the project	Complete	Kevin	09/26	10/16	20
☐	Task 3 - Decide on system design	Decide on the system design, how each component will	Complete	Kevin	10/17	10/24	7
☐	Task 4 - Decide on feature timeline	Decide on what features will be developed first	Complete	Kevin	10/25	11/01	5
☐	Task 5 - Implement project	Work on project begins	Complete	Kevin	11/02	04/24	180
☐	Subtask 1 - Get basic visualisation working	Get basic visualisation working - implement a default sorting algorithm and visualise it	Complete	Kevin	11/03	11/11	8
☐	Subtask 2 - Start user authentication	Start on user authentication, which allows a user to register and sign in to an account	Complete	Kevin	11/12	11/20	8
☐	Task 6 - PROJECT ON HOLD	Project on hold for Christmas/Work on other projects	Complete	Kevin	11/21	12/28	37
☐	Subtask 3 - Finish user authentication	Finish user authentication	Complete	Kevin	12/29	01/10	12
☐	Subtask 4 - Host Flask server on PythonAnywhere	Set up Flask server on PythonAnywhere	Complete	Kevin	01/11	01/12	1
☐	Subtask 5 - Enhance visualization	Enhance visualisation - add more sorting algorithms, fix bugs, etc.	Complete	Kevin	01/13	02/13	14
☐	Subtask 6 - Implement uploading of past sorts	Implement uploading of past sorts to database	Complete	Kevin	02/13	02/21	8
☐	Task 7 - Wire everything together, bug fix, and test	Wire everything up - push to GitHub	Complete	Kevin	02/22	03/31	38
☐	Task 8 - Work on dissertation	Dissertation work	Complete	Kevin	12/29	04/24	117

There were several project requirements that I set out when designing this project. These can be seen in 1.6. The functions and features required were:

- Visualisation of sorting algorithms
- Support of various sorting algorithms
- Upload of user created data sets
- User authentication
- Database support

1.6.2 Project Limitations

As a whole, the application is primarily built to sort elements represented in a numerical format. The limitations of the project would be the inability to sort elements represented in a numerical format, special characters, files, etc. In the context of an algorithms visualiser, while being able to sort items like letters, special characters, files, etc. could very well be viable, visualising such things mightn't be as suitable as visualising the sorting of numbers.

Chapter 2

Methodology

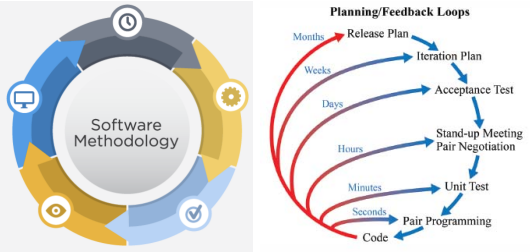
This chapter covers the various methodologies that were implemented in this project. It takes a look at the different types of research methodologies that were used such as Qualitative Research, Quantitative Research, and Mixed Methods Research and the different types of software development methodologies that were used such as Test Driven Development, Continuous Delivery, Extreme Programming, etc. and why each was chosen. Other areas also covered in this chapter include meetings, project management, development tools, source control, and testing.

2.1 Research Methodology

The research methodology that was used in this project was a Mixed Methods Research methodology, using a mix of both Qualitative Research and Quantitative Research. Qualitative research approaches are employed across many academic disciplines and is useful at an individual level. Qualitative data collection methods vary using unstructured or semi-structured techniques.

Fill out more...

2.2 Software Development Methodology



The software development methodology that was used in this project was Extreme Programming (XP). Extreme Programming is a software development methodology designed to improve the quality of software and its ability to properly adapt to the changing needs of the customer or client. While there was no customer or client for this project, this methodology was still applicable.

It is a form of Agile software development. The Agile methodology was developed as a response to growing frustrations with Waterfall and other highly structured, inflexible methodologies. This approach is designed to accommodate change and the need to produce software faster. Similar to other Agile methods of development, Extreme Programming aims to provide iterative and frequent small releases throughout the project, allowing both team members and customers to examine and review the project's progress throughout the entire SDLC (Software Development Life Cycle).

It became clear early on that the Agile methodology would be the most suitable methodology to use for this project as the Agile Methodology allows for incremental development, changing requirements, prototyping, and sustainable development. As there would be weekly meetings with the project supervisor, being able to show and discuss the progress of the project would be a bonus.

2.3 Meetings

Project meetings were held weekly for the duration of the project with the project supervisor. The majority of the meetings consisted of:

- Project progress updates.
- Feedback on progress.
- Planning of the next development iteration.
- Discussions on possible additional features that could be incorporated.
- QA on various project elements.

Initial project meetings were more focused on brainstorming possible projects that could be developed. The first two weeks comprised of research, whereby possible projects ideas, appropriate technologies, and a project timeline were developed.

2.4 Development Tools

The main Integrated Development Environment used throughout the project was Visual Studio Code. Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and mac OS. It includes support for debugging, embedded Git control and GitHub, syntax highlighting, intelligent code completion, snippets, and code refactoring.

Visual Studio Code is based on Electron, a framework which is used to develop Node.js applications for the desktop running on the Blink layout engine. Visual Studio Code is a source code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js and C++.

2.5 Source Control

Source control (or version control) is the practice of tracking and managing changes to code. GitHub provides hosting for software development version control using Git. Git is an open-source distributed source code management system.

Advantages

There are a number of advantages to using Git:

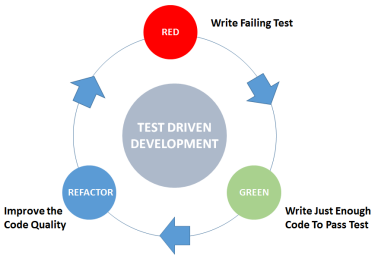
- **Distributed Development** - Git is a distributed version control system. Instead of a working copy, each developer gets their own local repository, complete with a full history of commits. Having a full local history makes Git fast, since it means you don't need a network connection to create commits, inspect previous versions of a file, or perform diffs between commits.
- **Faster Release Cycle** - As a result of feature branches, distributed development, pull requests, etc. is a faster release cycle. This facilitates an agile workflow, encouraging developers to share smaller changes more frequently. This results in changes get pushed down the development pipeline faster
- **Testing** - The main type of software testing development process used throughout the project was Test-driven development (TDD). Several different types of testing was used throughout in conjunction, such as Integration Testing, Unit Testing, System Testing, Performance Testing, etc. Several tools were used to aid in this, which included Postman, ... TDD allows for continuous testing to happen alongside the development of the project.

Disadvantages

Git is a very well-designed tool and, as such, has very few disadvantages:

- **Pricing** - Some of GitHub features, as well as features on other online repositories, are locked behind a SaaS paywall. If you have a large team, this can add up fast. Those who already have a dedicated IT team and their own internal servers are often better off using their own internal git for cost reasons, but for most, the cost isn't outrageous.
- **Security** - GitHub does offer private repositories, but this isn't necessarily perfect for many. For high value intellectual property, you're putting all of this in the hands of GitHub as well as anyone who has a login, which like many sites has had security breaches before and is targeted constantly. In addition, some clients/employers will only allow code on their own secure internal Git as a matter of policy.

The above mentioned testing processes suited this type of project as there were many different iterations of the project. Using testing processes such as Integration Testing, Unit Testing, System Testing, etc. ensured new and old code worked with each and every iteration of the project.



Chapter 3

Technology Review

This chapter discusses the different technologies used in throughout the project. It discusses the the advantages and disadvantages of each technology and why certain technologies were used over others.

3.1 Overview

This project is comprised of React as the frontend, Flask as the server which is hosted on PythonAnywhere, MongoDB as the database for user authentication, Firebase as the database to store sorts, and Docker to deploy/host the application. Throughout the project, the following technologies were also used and tested before the above was ultimately chosen:

- Angular/Ionic
- Django
- MySQL
- Amazon Web Services
- Heroku

3.2 Main Technologies

This section will discuss the main technologies currently in use in the web application. The preceding section will discuss other technologies tried but ultimately weren't incorporated.

3.2.1 React

React (also known as React.js or ReactJS) is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies.
React can be used as a base in the development of single-page or mobile applications. However React is only concerned with rendering data to the DOM and so creating React applications usually requires the use of additional libraries for state management and routing. Redux and React Router are respective examples of such libraries.

Advantages

React has many advantages, several of which apply to this project:

- **Wide React and Redux toolset** - Both React and Redux come with a decent set of related tools that make a developer's life easier. For instance, React Developer Tools extension for Chrome and a similar one for Firefox allow for examining component hierarchies in the virtual DOM and editing states and properties. Additionally, you can check React Sight that visualizes state and prop trees; Reselect DevTools that helps with debugging and visualizing Reselect, a selector library for Redux. Redux DevTools Profiler Monitor allows for profiling actions in well.
- **Faster Rendering** - Building a high-load application is essential to consider how the structure will impact the overall app performance. Even latest platforms and engines can't ensure the absence of annoying bottlenecks, because DOM (Document Object Model) is tree-structured and even small changes at the upper layer can cause awful ripples to the interface. To solve the issue, Facebook development team has introduced Virtual DOM – currently, one of the benefits of using React for heavy loaded

and dynamic software solutions. As the name suggests, it is a virtual representation of the document object model, so all the changes are applied to the virtual DOM first and then, using diff algorithm, the minimal scope of necessary DOM operations is calculated. Finally, the real DOM tree is updated accordingly, ensuring minimum time consumed. This method guarantees better user experience and higher app performance.

- **Guaranteed Stable Code** - To make sure that even small changes that take place in the child structures won't affect their parents, ReactJS uses only downward data flow. Changing an object, developers just modify its state, make changes, and, after that, only particular components will be updated. This structure of data binding ensures code stability and continuous app performance.

- **Strong Community** - Initially, React library was created for internal use and later shared with the entire world. Currently, it is supported by Facebook and Instagram engineering teams, plus external experts. For example, React GitHub repository numbers over 1100 contributors, while users can ask their questions on Stack Overflow, Discussion forum, Reactiflux Chart, Freenode IRC, social media platforms and many others.

Disadvantages

There are a few disadvantages to using React:

- **Poor Documentation** - The problem with documentation traces back to constant releases of new tools. Different and new libraries like Redux and Reflux are promising to accelerate the work of a library or improve the entire React ecosystem. At the end, developers struggle with integrating these tools with ReactJS. Some members of the community think that React technologies are updating and accelerating so fast that there is no time to write proper instruction. To solve this, developers write their own documentation for specific tools used by them in current projects.

- **JSX as a barrier** - ReactJS uses JSX. It's a syntax extension, which allows mixing HTML with JavaScript. JSX has its own benefits (for instance, protecting code from injections), but some members of the development community consider JSX to be a serious disadvantage. Developers and designers complain about JSX's complexity and consequent steep learning curve.

3.2.2 Flask

Flask is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Some of the main features of Flask are:

- Development server and debugger
- Integrated support for unit testing
- Support for secure cookies (client side sessions)
- 100% WSGI 1.0 compliant

For this project, Flask was used as the middle-man for the React application and MongoDB database. Certain functionality, such as enabling users to register, login, upload sorts, etc. are written in Flask which are then accessed by the React application when needed. The Flask application is hosted on PythonAnywhere.

Advantages

There are a number of advantages to using Flask:

- **Flexibility** - There are very few parts of Flask that cannot be easily and safely altered, due to its simplicity and minimality.
- **Modularity** - Modular code provides a huge number of benefits. With Flask, you have the ability to create multiple Flask applications or servers, distributed across a large network of servers, each with specific purposes. This creates more efficiency, better testability, and better performance.
- **Performance** - A micro framework can be thought of as being slightly more "low-level" than something like Django. There are fewer levels of abstraction between the user and the database, the requests, the cache, etc. so performance is inherently better from the start.

- **Scalable** - Flask can be argued to be more scalable than monoliths if using modern methods. Today, applications are often running in containers or using cloud computing with auto-scaling. Applications do not typically "scale" themselves. The infrastructure scales. With a smaller application, it's easier to deploy instances across thousands of server easily to handle increased traffic/load. For example, it's partly the reason why Pinterest needed to migrate from Django to Flask as they grew to support more of a micro-services pattern.
- **Simpler Development** - If one understands Python well, then you'll be able to move around and contribute to a Flask application easily. It's less opinionated so there are fewer standards to learn.

Disadvantages

There are a few disadvantages to using Flask:

- **Community** - The monoliths provide such a large toolset, focused on providing solutions for a larger set of use cases out of the box, that they typically have a larger community. This means more eyes on the code, more code reviews, and better-tested core code. This is a little bit of a generalization though.
- **Fewer tools** - You don't have a full toolset underneath you. So you may need to build more on your own or search out extensions/libraries from the community.
- **Not standardized** - While Flask is simple, it's not very opinionated. A Python developer without Flask experience will get adjusted to a Flask application quicker than a Python developer without Django experience would get adjusted to a Django application. But Django is building up a large group of talent who knows the framework very well. A Python developer with Django experience will get adjusted to a new Django app quicker than a Python developer with Flask experience would get adjusted to a large Flask application.

3.2.3 PythonAnywhere

PythonAnywhere is an online integrated development environment (IDE) and web hosting service (Platform as a service) based on the Python programming language. it provides in-browser access to server-based Python and Bash command-line interfaces, along with a code editor with syntax highlighting. Program files can be transferred to and from the service using the user's browser. Web applications hosted by the service can be written using any WSGI-based application framework. For this project, the user authentication side of things was initially handled entirely by Firebase. However, as suggested by my supervisor, I decided to implement registering a user, logging a user into the application etc. myself. The functionality for this was written in Python and could be accessed by the application by using a proxy to delegate any requests made by the application to the Flask server. However, to enable the functionality to be accessed from anywhere (and, in turn, remove the need to run the Flask server alongside the application everytime), I decided to use PythonAnywhere. EXPLAIN

Advantages

There are a number of advantages to using PythonAnywhere:

- **Always running** - Even on a free tier account, PythonAnywhere never sleeps (compared to something like Heroku). This means real time services are viable with PythonAnywhere.
- **Fully configured** - PythonAnywhere gives you a fully configured Python environment...
- **Free** - PythonAnywhere is free to use. On a free tier account, web applications stay running 24/7 for 3 months. After 3 months, it will shut down. However, one needs only to start it up again.

Disadvantages

There are a few disadvantages to using PythonAnywhere:

- **Python-only on the server side** - You are free to use JavaScript in your web pages and so on, but you can't use Rails or Node on the server side of things.

3.2.4 MongoDB

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema. Some of the main features of MongoDB are:

- Ad-hoc queries (MongoDB supports field, range query, and regular-expression searches)
- Indexing (Fields in a MongoDB document can be indexed with primary and secondary indices)
- File storage (MongoDB can be used as a file system, called GridFS, with load balancing and data replication features over multiple machines for storing files)

For this project, MongoDB was used as the main database. It stores user details and user sorts. To write to the database, a request is first made through the application. Using a proxy, these requests are delegated to a flask application. The flask application then processes the request and will then access the database to perform a certain action (such as writing user details to the database, accessing user details, storing user sorts, etc).

Advantages

There are a number of advantages to using MongoDB:

- **Schema-less NoSQL Database** - MongoDB is a schema-less NoSQL database, meaning there is no need to design the schema of the database when using MongoDB. The code defines the schema.
- **Performace** - Performance of MongoDB is much higher than compared to any relational database.
- **Internal Memory** - MongoDB uses internal memory for storage which enables faster access to the data.
- **No Joins** - MongoDB doesn't use complex joins. There is no relationship among data in MongoDB.
- **JSON** - MongoDB supports JSON. Because MongoDB uses JSON format to store data, it is very easy to store arrays and objects.

Disadvantages

There are a few disadvantages to using MongoDB:

- **High Memory** - MongoDB uses high memory for data storage.
- **Document Size Limit** - MongoDB has a limit to the size of documents it can store.
- **Lack Of Transaction Support** - MongoDB does not support transactions.

3.2.5 Firebase

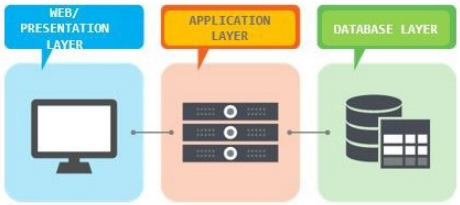
Firebase is a mobile and web application development platform developed by Firebase, Inc. in 2011, then acquired by Google in 2014.

Chapter 4

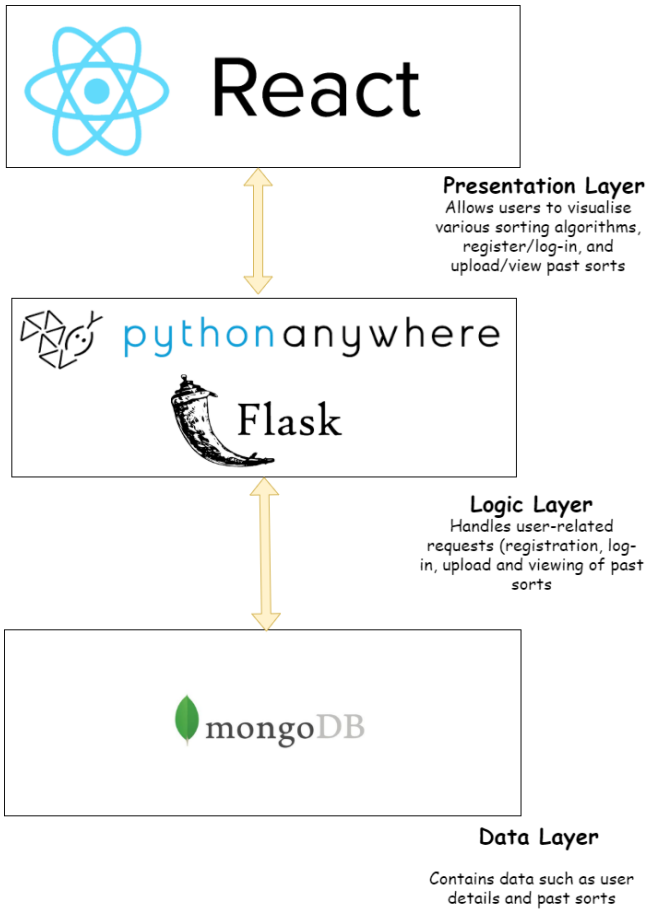
System Design

This chapter discusses the system design, analysing the various aspects of the project and purpose of each component such as the visualisation, the sorting algorithms, user authentication, the server, etc.

4.1 Overview



This project utilises the multi-tier architecture (often referred to as n-tier architecture) platform. Multi-tier architecture or multilayered architecture is a client–server architecture in which presentation, application processing, and data management functions are physically separated. The most widespread use of multi-tier architecture is the three-tier architecture. This project consists of ... main components, all working together: The web application, which represents the presentation layer, allows users to visualise various sorting algorithms, register and login, upload and view past sorts from other users. The Flask server, which represents the logic tier, handles requests such as account registration and log in, and uploading and retrieving of sorts by users.

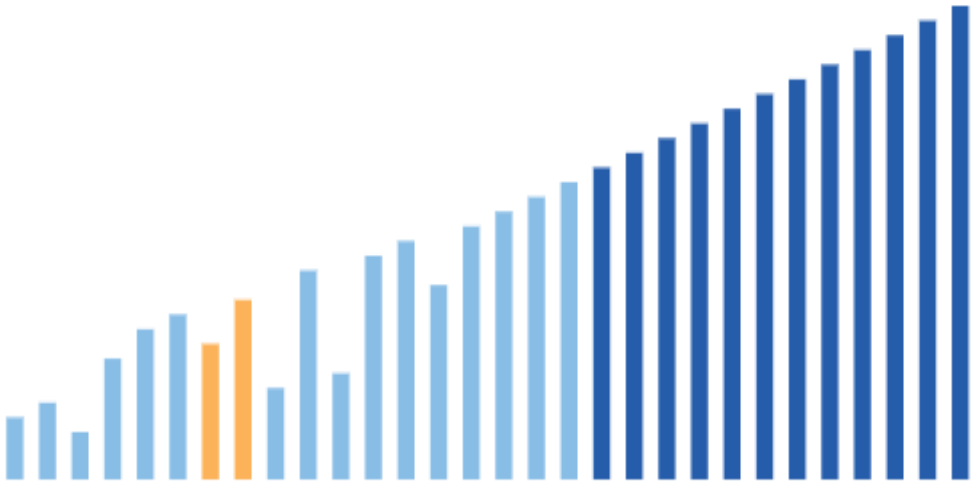

React
Presentation Layer
Allows users to visualise various sorting algorithms, register/log-in, and upload/view past sorts
pythonanywhere
Flask
Logic Layer
Handles user-related requests (registration, log-in, upload and viewing of past sorts
mongoDB
Data Layer
Contains data such as user details and past sorts

4.2 Web Application

The web application was designed and developed first as this is the main aspect of the project itself. The main objective of the web application was to allow a user to choose a sorting algorithm and be able to visualise it. Secondary objectives, which were developed at a later point, was to allow users to register and log in to an account, and have the ability to upload and view previous sorts. This was greatly aided by the decision to use ReactJS .... The web application consists of four main pages:

- **Sorting Page** - The sorting page, as shown in Figure 4.2.2, is the main page of the application. Allows a user to choose one of several sorting algorithms to visualise, generate a random array of elements to sort, and specify their own dataset to sort.
- **Login Page** - The login page handles user authentication and will login a user to the application if that user exists within the database. The user will then be redirected to the sorting page, with new functionality available such as ...
- **Register Page** - The register page handles user authentication and will register a user with the application. The user will then be redirected to the login page, where they can then attempt to login.
- **Sorts Page** - The sorts page displays all previous sorts recorded and saved by past users.

4.2.1 Sorting



There are currently seven algorithms that can be visualised within the application. They are as follows:

- Bubble Sort
- Heap Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Selection Sort
- Shell Sort

Bubble Sort

```
export default class BubbleSort {
    /**
     * Performs the bubble sort algorithm
     *
     * Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly
     * steps through the list, compares adjacent elements and swaps them if they are in the wrong order.
     * The pass through the list is repeated until the list is sorted. This algorithm is not suitable for large data
     * sets as its average and worst case complexities are of O(n2), where n is the number of items.
     *
     * @param {*} array - Array of items to be sorted
     * @param {*} sortHistory - Previous items that have been sorted
     * @param {*} highlightHistory - Previous items that have been...
     */
    static bubbleSort(array, sortHistory, highlightHistory) {
        this.clearArray(sortHistory);
        this.clearArray(highlightHistory);

        let counter = 0;

        for (let i = 0; i < array.length - 1; i++) {
            for (let j = 0; j < array.length - (i + 1); j++) {
                sortHistory.push(array.slice());
                highlightHistory.push([j + 1, j]);

                counter++;

                // If index j in the array is larger than index j + 1 in the array, swap them
                if (array[j] > array[j + 1]) {
                    this.swap(array, j, j + 1);
                }
            }
        }
    }
```

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. The algorithm, which is a comparison sort, is named for the way smaller or larger elements "bubble" to the top of the list.

This simple algorithm performs poorly in real world use and is used primarily as an educational tool. More efficient algorithms such as Tim Sort, or Merge Sort are used by the sorting libraries built into popular programming languages such as Python and Java.

Heap Sort

Insertion Sort

```
export default class InsertionSort {
    /**
     * Performs the insertion sort algorithm
     *
     * Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time.
     * It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
     * This algorithm is not suitable for large data sets as its average and worst case complexities are of O(n2), where n is the
     * number of items.
     *
     * @param {*} array - Array of items to be sorted
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static insertionSort(array, sortHistory, selectedHistory) {
        this.clearArray(sortHistory);
        this.clearArray(selectedHistory);

        let leftCol = -1;

        // Iterate through entire array
        for (let i = 0; i < array.length; i++) {
            leftCol = i;
            let temp = array[i];
            let j;

            for (j = i - 1; j >= 0 && array[j] > temp; j--) {
                array[j + 1] = array[j];

                sortHistory.push(array.slice());
                selectedHistory.push([j, leftCol]);
            }

            array[j + 1] = temp;
        }

        sortHistory.push(array.slice());
        selectedHistory.push([0, array.length - 1]);
    }
```

Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as Quick Sort, Heap Sort, or Merge Sort. However, insertion sort provides several advantages:

-

When people manually sort cards in a bridge hand, most use a method that is similar to insertion sort.

Merge Sort

```
export default class MergeSort {
    /**
     * Performs the merge sort algorithm on one half of array
     *
     * Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in
     * two halves, calls itself for the two halves and then merges the two sorted halves.
     *
     * @param {*} array - Array to be sorted
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static mergeSort(array, sortHistory, selectedHistory) {
        let step = 1;

        // While still able to step through the array
        while (step < array.length) {
            let left = 0;

            while (left + step < array.length) {
                this.mergeBottomUp(array, left, step, sortHistory, selectedHistory);

                left += step * 2;
            }

            step *= 2;
        }
    }
```

```
    /**
     * Performs merge sort on remaining half of array
     *
     * @param {*} array - Array to be sorted
     * @param {*} left - Starting index
     * @param {*} step - Amount to step forward in array
     * @param {*} sortHistory - Elements that have been sorted
     * @param {*} selectedHistory - Elements that have been previously selected for sorting
     */
    static mergeBottomUp(array, left, step, sortHistory, selectedHistory) {
        let right = left + step;
        let end = Math.min(left + step * 2 - 1, array.length - 1);
        let stepLeft = left;
        let stepRight = right;
        let temp = [];

        for (let i = left; i <= end; i++) {
            if ((array[stepLeft] <= array[stepRight] || stepRight > end) && stepLeft < right) {
                temp[i] = array[stepLeft];

                stepLeft++;
            } else {
                temp[i] = array[stepRight];

                stepRight++;
            }
        }

        for (let j = left; j <= end; j++) {
            array[j] = temp[j];
        }

        sortHistory.push(array.slice());
        selectedHistory.push([-1]);
    }
```

In computer science, merge sort (also commonly spelled mergesort) is an efficient, general-purpose, comparison-based sorting algorithm. Most implementations produce a stable sort, which means that the order of equal elements is the same in the input and output. Merge sort is a divide and conquer algorithm that was invented by John von Neumann in 1945.[2] A detailed description and analysis of bottom-up mergesort appeared in a report by Goldstine and von Neumann as early as 1948

Quick Sort

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm. Developed by British computer scientist Tony Hoare in 1959[1] and published in 1961,[2] it is still a commonly used algorithm for sorting. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort.[3][contradictory]
Quicksort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. This can be done in-place, requiring small additional amounts of memory to perform the sorting.
Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. Efficient implementations of Quicksort are not a stable sort, meaning that the relative order of equal sort items is not preserved.
Mathematical analysis of quicksort shows that, on average, the algorithm takes O(n log n) comparisons to sort n items. In the worst case, it makes O(n2) comparisons, though this behavior is rare.

Selection Sort

```
export default class SelectionSort {
    /**
     * Performs the selection sort algorithm
     *
     * Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based
     * algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted
     * part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list. The
     * smallest element is selected from the unsorted array and swapped with the leftmost element, and that element
     * becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to
     * the right. This algorithm is not suitable for large data sets as its average and worst case complexities
     * are of O(n2), where n is the number of items.
     *
     * @param {*} array
     * @param {*} sortHistory
     * @param {*} highlightHistory
     */
    static selectionSort(array, sortHistory, highlightHistory) {
        this.clearArray(sortHistory);
        this.clearArray(highlightHistory);

        let leftCol = -1;

        /**
         * Iterate through the array
         */
        for (let i = 0; i < array.length; i++) {
            let min = i;
            leftCol = min;

            // Iterate through the array, starting at the index immediately after index i
            for (let j = i + 1; j < array.length; j++) {

                // If array index j is less than the minimum, j is the new minimum
                if (array[j] < array[min]) {
                    min = j;
                }

                sortHistory.push(array.slice());
                highlightHistory.push([j, leftCol, min]);
            }

            // If i doesn't equal the minimum, swap i and min
            if (i !== min) {
                this.swap(array, i, min);
            }
        }

        sortHistory.push(array.slice());
        highlightHistory.push([-1, array.length - 1]);
    }
```

In computer science, selection sort is an in-place comparison sorting algorithm. It has an O(n2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity and has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

The time efficiency of selection sort is quadratic, so there are a number of sorting techniques which have better time complexity than selection sort. One thing which distinguishes selection sort from other sorting algorithms is that it makes the minimum possible number of swaps, n 1 in the worst case.

Shell Sort

4.2.2 Visualization

```
/**
 * Renders a bar - these bars are for visualisation and will be sorted based on heigh/value
 * @param {*} props
 */
function Bar(props) {
  const barStyling = {
    bar: {
      color: 'green',
      display: 'inline-block',
      width: 20,
      margin: 3,
      height: props.size * 8,
      backgroundColor: props.color
    },
    text: {
      display: 'inline-block',
    }
  }

  // Renders a bar for element in the array
  return (
    <div className='bar' style={barStyling.bar}>
      <h7 style={barStyling.text}>{ barStyling.bar.height / 8}</h7>
    </div>
  );
}
```

4.3 Flask Server

The Flask server, which is hosted on PythonAnywhere, is the middle-man of the entire application and allows the web application to communicate with the database. The server was developed and integrated into the web application secondly alongside the database. It handles requests, such as user login requests, user registration requests, uploading saved sorts, etc. made by the web application and performs the appropriate action. The database is then read and/or updated based on this.

```
42    },
43    "proxy": "http://kniland97.eu.pythonanywhere.com"
44  }
```

Flask server for Kevin Niland's Algorithms Visualizer

Owner: Kevin Niland

Running since: 02/03/2020

4.4 Database

Fill out....

Chapter 5

System Evaluation

This chapter evaluates the system, analysing the various aspects of the project to ensure the requirements of the project were met and discusses the various types of testing that was performed throughout the software development cycle to ensure each of the requirements was met and the software was of high quality.

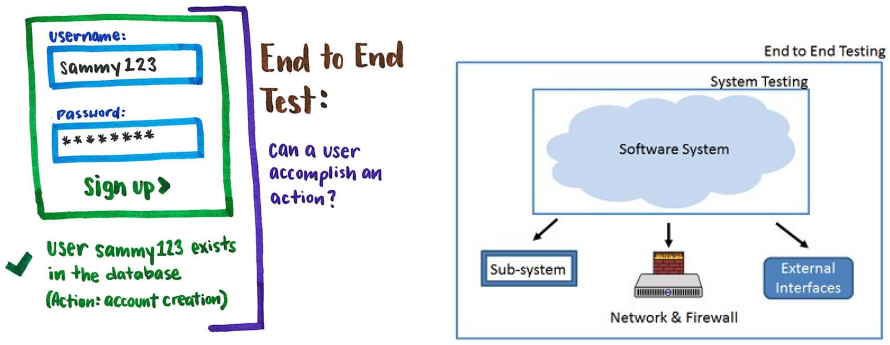
5.1 Overview

The system was designed and developed using the Test Driven Development (TDD) methodology. As each unit of code was written and before each new component was integrated into the system, appropriate tests were carried out and adjusted if needed. Because of this, a lot of bugs were caught early on, allowing them to be fixed with minimal disruption to the system as a whole. This also allowed the system to be redesigned early on when necessary.

The types of testing carried out were as follows:

- End-to-End Testing
- Exploratory Testing
- Functional Testing
- Graphical User Interface (GUI) Testing
- Integration Testing
- Unit Testing
- System Testing

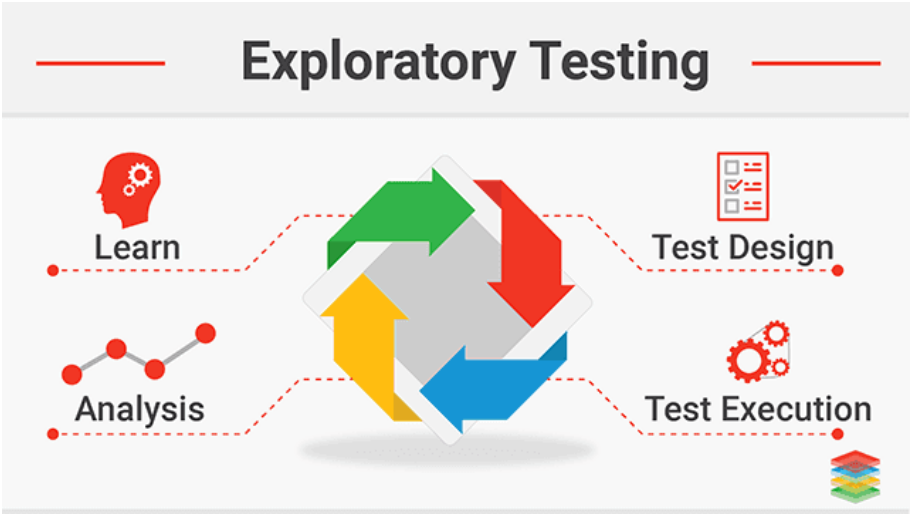
5.2 End-to-End Testing



End-to-end testing is a technique used to test whether the flow of an application right from start to finish is behaving as expected. The purpose of performing end-to-end testing is to identify system dependencies and to ensure that the data integrity is maintained between various system components and systems.

The entire application is tested for critical functionalities such as communicating with the other systems, interfaces, database, network, and other applications.

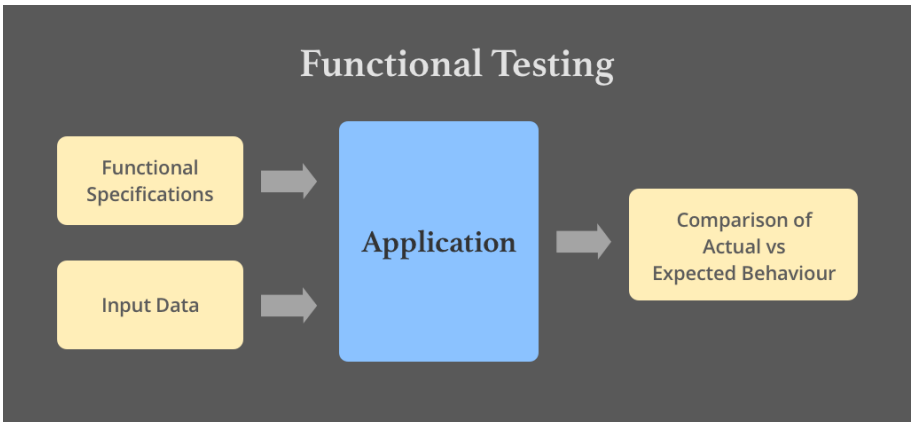
5.3 Exploratory Testing



Exploratory testing is a technique or a method whose aim is to discover the flaws during the process of software development.

This technique of testing is concerned about the qualitative assurance of the software. It is used to discover the anonymous issues during the process of development of software.

5.4 Functional Testing



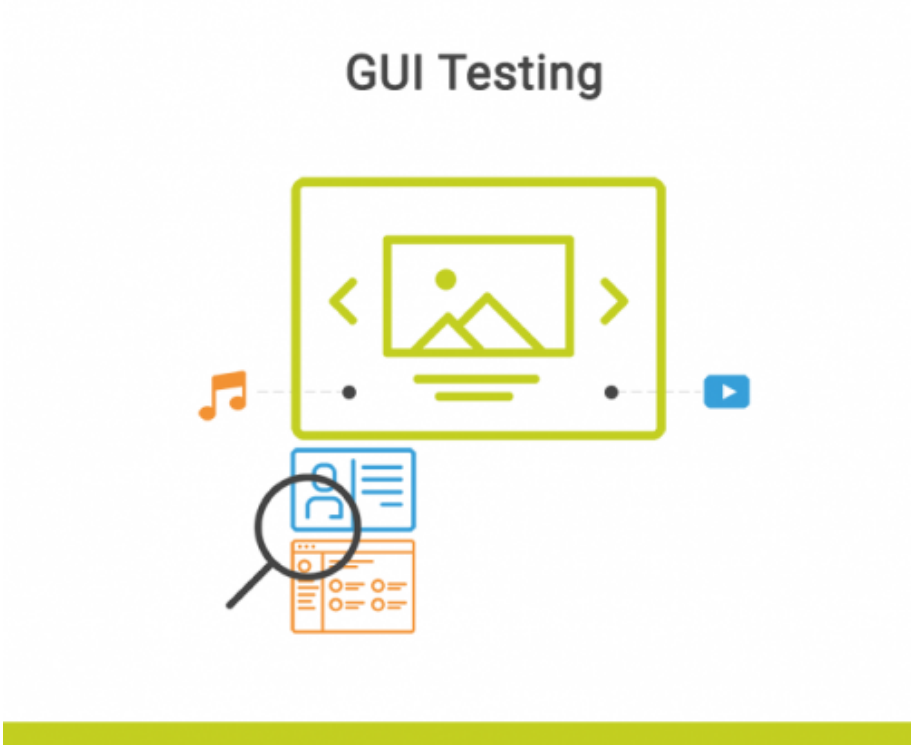
Functional Testing is a type of software testing whereby the system is tested against the functional requirements/specifications.

Functions (or features) are tested by feeding them input and examining the output. Functional testing ensures that the requirements are properly satisfied by the application. This type of testing is not concerned with how processing occurs, but rather, with the results of processing. It simulates actual system usage but does not make any system structure assumptions.

During functional testing, Black Box Testing technique is used in which the internal logic of the system being tested is not known to the tester.

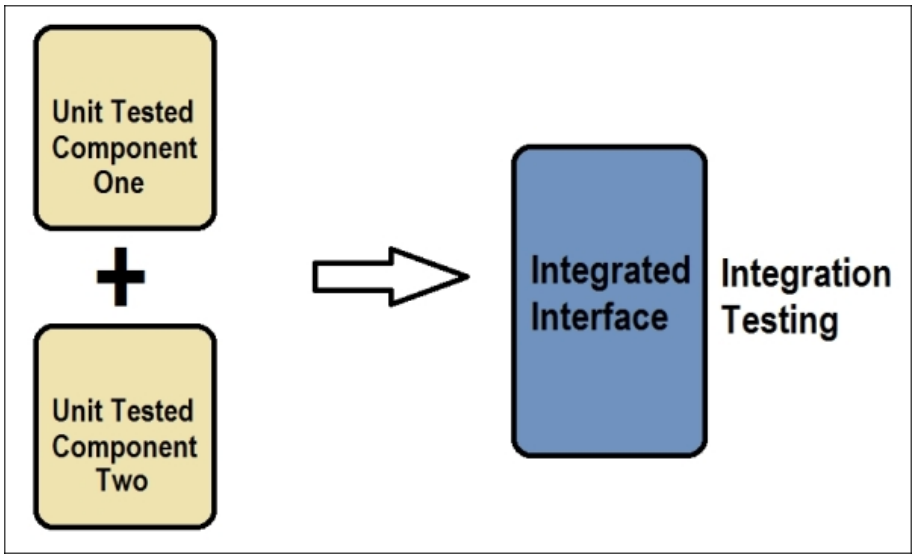
5.4.1 Web Application

5.5 Graphical User Interface (GUI) Testing



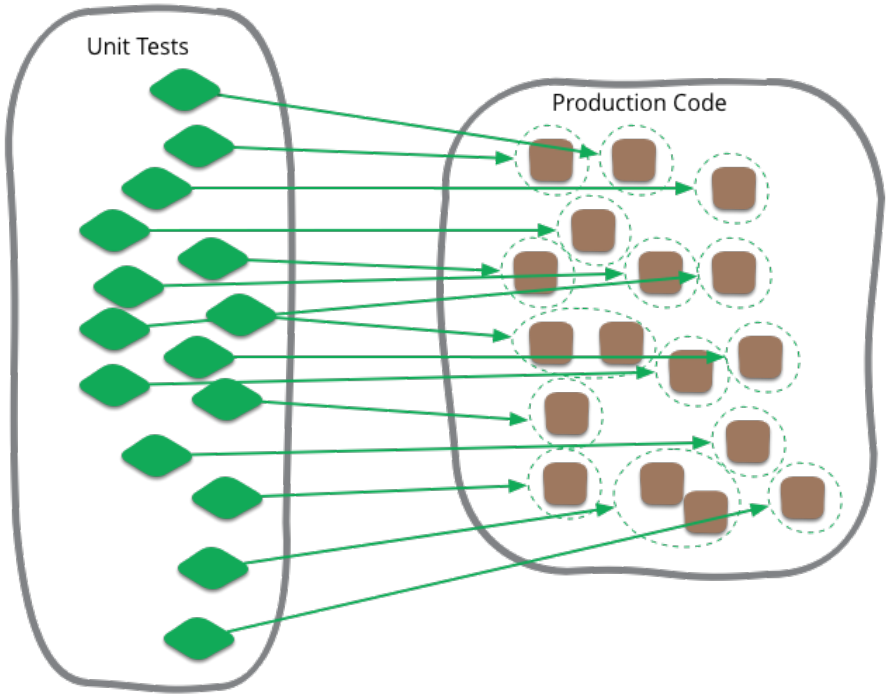
Graphical User Interface (GUI) Testing is a software testing type that checks the Graphical User Interface of the Application Under Test. GUI Testing involves checking the screens with the controls like menus, buttons, icons, and all types of bars - toolbar, menu bar, dialog boxes, and windows, etc. The purpose of GUI Testing is to ensure UI functionality works as per the specification.

5.6 Integration Testing



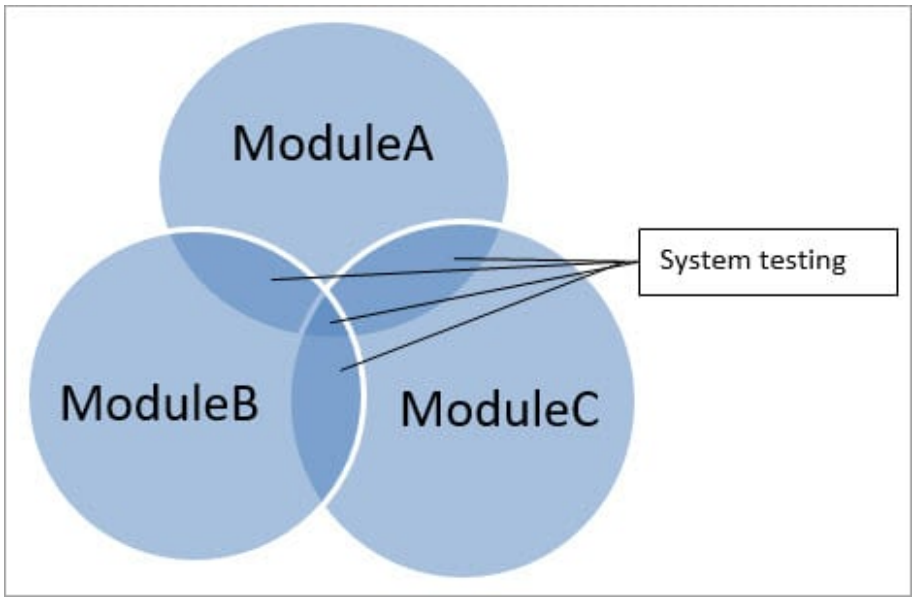
Integration testing (sometimes called integration and testing, abbreviated IT) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.

5.7 Unit Testing



Unit Testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

5.8 System Testing



System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems.

Chapter 6

Conclusion

This chapter serves as a conclusion to the project. In it, the objectives of the project are analysed again briefly to evaluate whether or not they were met. It also analyses any improvements and/or changes that could be added/made to the project if it was to be developed again.

Chapter 7

Appendices

[**example**]