Functional Specification
Source Code Analyser Engine

Bernard O'Connor
Student Number: 14367821

Date: 24/11/2017

Supervisor: Dr David Sinclair

Functional Specification Contents

1. Introduction

1.1 Overview

The system is a Source Code Analyser Engine. The aim of the system is to create a tool that will help with the promotion of quality code writing, that other developers may use and integrate into their own code teaching websites, IDE plugins, or other such projects.

The system is a java library, which is written in Clojure, that will take in a piece of source code, analyse it to see what code style improvements may be made, and return these suggestions to the user.

A version of the library will also be available to use on a website that I will host through a public web API.

Initially the Source Code Analyser Engine will be written for Clojure source code, but will be designed to make the addition of other language support easier through a “language rulebook” system.

1.2 Business Context

I envision going open source with the project after Final Year. The Source Code Analyser Engine will be a freely available library for developers to use within their own projects such as code teaching web platforms and IDE plugins.

The open source nature of the final project will come in the form of open sourcing the additional support of more languages. I intend to make all language specific processing modular and standardised into separate sections that for now I will be calling “rulebooks”. For the Final Year Project I will focus on creating the Clojure Rulebook, then afterwards I will be open to accepting merge requests for the addition of rulebooks for other languages.

1.3 Glossary

- SCAE - Source Code Analyser Engine
- API - “a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.”
- Clojure(Script) - Clojure is a functional language that is built upon the JVM, and compiles down to Java. ClojureScript is a web-based version of Clojure that compiles down to JavaScript
- IDE plugin- An optional module that may be included with an IDE
- Library - “a collection of precompiled routines that a program can use”
- Rulebook - This is a combination of style rules (to be used in the style analysis phase) and token definitions (to be used in the lexical analysis phase).
- Style Rules - These rules are used to map between poor code style that has been detected to the suggested improvements to the poor code.
- Lexical Analyser - “lexical analysis, lexing or tokenization is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an assigned and thus identified meaning).”
- Multipart/form-data - A type of data encoding used in file uploads

2. General Description

2.1 Product / System Functions

Describes the general functionality of the system / product.

- Receive request to analyse source code through the library API
- Perform lexical analysis on source code to tokenise it
- Perform style analysis on the tokenised code
- Accept custom rulebooks to allow users to create their own style rules
- Host website where users may use the library online, download the library for their own use, or read up on the documentation for the library

2.2 User Characteristics and Objectives

- Open Source Community
  - Focus on expanding language and rule support
- Developers
  - Focus on using the engine in their own project
- Wishlist:
  - Well defined API
  - Clear Documentation

2.3 Operational Scenarios

Scenario ID	1
User Objective	User makes call to my website API
User Action	The user makes a call to the web API that is hosted on my website. In the API call the user must provide the code to be analysed and which rulebook should be used. The source code being submitted can be submitted as either a string or a multipart/form-data. The rulebook that is specified to use shall be supplied as a string. This string will be parsed to determine if it is a link to an external rulebook, if it is a full rulebook supplied as a string, or if it is a string enum pointing to one of my provided rulebooks.
Comments	The web API acts as a public facing wrapper for the version of the Source Code Analyser Engine that I will host on my website. The user calls the Web-API, which

	in turn calls the API of the library installed on the webserver. This feature is useful for developers who do not wish to download and use the library locally. This allows for quick use of the library.

Scenario ID	2
User Objective	User Downloads Library
User Action	The user goes onto my website. The user navigates to the download page (via the side nav-bar). The user selects which version of the SCAE Library they wish to download.
Comments	The library shall be packaged as a jar file.

Scenario ID	3
User Objective	User Incorporates Library in their project
User Action	In the intellij IDE they can follow this workflow (similar workflows exist for other IDE's) ● File > Project Structure ● Project Settings > Modules > Dependencies > "+" > JARS or directories ● Select the Source Code Analyser Engine jar file ● Click OK
Comments	After completing Scenario 2, the user must add the library as an external JAR or Library to their project. This can be done either manually or automatically through the use of an IDE. Steps for the intellij IDE outlined above

Scenario ID	4
User Objective	User makes call to local library
User Action	The user makes an API call directly to the local library they have installed on their system. They provide the same information as outlined in Scenario 1
Comments	Similar to scenario 1, though this time the user is making an API call directly to the version of the SCAE Library that they have installed.

Scenario ID	5
User Objective	User uses sample code submission page on my website
User Action	The user will be presented with the option of uploading a file of code as multipart/form-data or as a string through the use of a textbox. The user will also be able to select the rulebook to use by either entering a link into a textbox, entering the rulebook as a string, or by selecting one of my provided rulebooks through the use of a dropdown.
Comments	Allows users to either easily test out how the library works, to to use the library in a single instance to check their code quality without having to use the library as part of a separate application.

Scenario ID	6
User Objective	User makes their own rulebooks
User Action	User follows the documentation/ tutorials on my website in order to create their own rulebooks. The user can test the rulebook that they have made by supplying the rulebook as part of a call to the API. They can then verify that the rulebook is in the format that is expected and that it is behaving as expected. If not the call to the API will return an error.
Comments	

Scenario ID	7
User Objective	User submits their own rulebook
User Action	Following on from Scenario 6, once the user is happy that their custom rulebook is working, they may make a merge request to the main project. After the submitted the rulebook has been reviewed, it will either be accepted or rejected as being one of the pre-provided rulebooks of the project.
Comments	

Scenario ID	8
User Objective	User Views Documentation
User Action	The user goes onto the project website. The user then navigates to the documentation section via the side nav-bar.
Comments	

2.4 Constraints

- Should not be resource intensive if it is to be part of a larger IDE plugin
- Responsive
- Standardised Format & Stability Across Versions
- As other projects may link to the the web-API that I host, it is therefore my responsibility to ensure that no code breaking changes / changes to the format of the response of the API are ever released to the version of the project that the Web-API communicates to. The version used by the Web-API must remain stable.
- Website Uptime
- The Web-API should have a high uptime to prevent any projects that link to it from breaking if the website goes down.

3. Functional Requirements

Requirement ID	1
Description	Receive a request to parse code from the Library API. This request must include the code to analyse, and specify which rulebooks to use.
Criticality	This is critical as without a functioning API the project can not be used.
Technical Issues	Designing an easy to use API with a standard format, that can later be added upon without breaking any previous functionality.
Dependencies	

Requirement ID	2
Description	Tokenise the input code through the use of the Lexical Analyser
Criticality	This is one of the core processes of the Source Code Analyser Engine, as this step is ra prerequisite for analysing the style of the code.
Technical Issues	Must implement my own lexical analyser that can accept different token definitions (provided by the rulebook), with which it will perform it’s lexical analysis
Dependencies	This depends on requirement 1 as it needs the both the input code to analyse and to know which rulebook tokens to check against.

Requirement ID	3
Description	Analyse the tokenised code to see if it matches any styling rules, and return the appropriate suggestions
Criticality	Again this is of the utmost importance, as the response from this is the service that this library aims to provide, suggestions on how to improve code quality in accordance with the provided styling rules.

Technical Issues	This will be the most challenging aspect of the project. I must use a combination of code sequence building and then token pattern matching in order to be able to see which style rule's suggestion to return.
Dependencies	Depends on requirement 1 for the style rulebook to be used, and depends on requirement 2 for the tokenised code to be analysed.

Requirement ID	4
Description	Host a publically available Web-API to expose tan instance of the Source Code Analyser Engine running on my website, to developers wishing to link to it without downloading and running the library themselves
Criticality	Important but not critical. Requirements 1,2,3 are critical for the library to function. Requirement 4 is an implementation of the library in order to provide a useful service to potential users and current users of the system.
Technical Issues	Similarly to the Library API, the Web-API must be designed to be easy to use and to remain stable across different versions of the project.
Dependencies	Depends on requirements 1,2 and 3, as those requirements make up the the Source Code Analyser Engine Library that the Web-API will point to.

Requirement ID	5
Description	Accept external rulebooks. As described in User Scenario 1, users may link to an external rulebook. The external rulebook may be provided as a string or as a URL link. If a URl is provided, the Source Code Analyser Engine will attempt to make a GET request on the URL, in order to retrieve the rulebook.

Criticality	Important for developers wishing to use their own / other people's custom rulebooks. Non-critical for users wishing to use the inbuilt provided rulebooks.
Technical Issues	Must make a request to an external domain, and verify that the response is in the expected format.
Dependencies	Depends on requirement 6 to sanitise the rulebooks. Also depends on requirements 1 and 4 to provide the url to the rulebook.

Requirement ID	6
Description	Sanitise external rulebooks. As the rulebooks will be run by the library and may be user submitted, they must be sanitised to ensure security.
Criticality	To reduce any security vulnerabilities of the library, this is very important
Technical Issues	Must be able to safeguard against different kinds of potential malicious attacks. More research required on what standard attacks must be safeguarded against.
Dependencies	Depends on requirement 6 and its dependencies in order to retrieve the rulebook to be sanitised.

Requirement ID	7
Description	Allow user to enter code into a sample textbox on the sample code submission page. Also allow user to select which rulebook to use on this page.
Criticality	Moderate. As opposed to the Web-API, this functionality is intended as a simple demo tool or for single use users. Unlike the Web-API, this sample page will not be used within anyone else’s project.
Technical Issues	This page must be able to make a call to the Web-API.
Dependencies	Depends on requirements 1-6 and 8

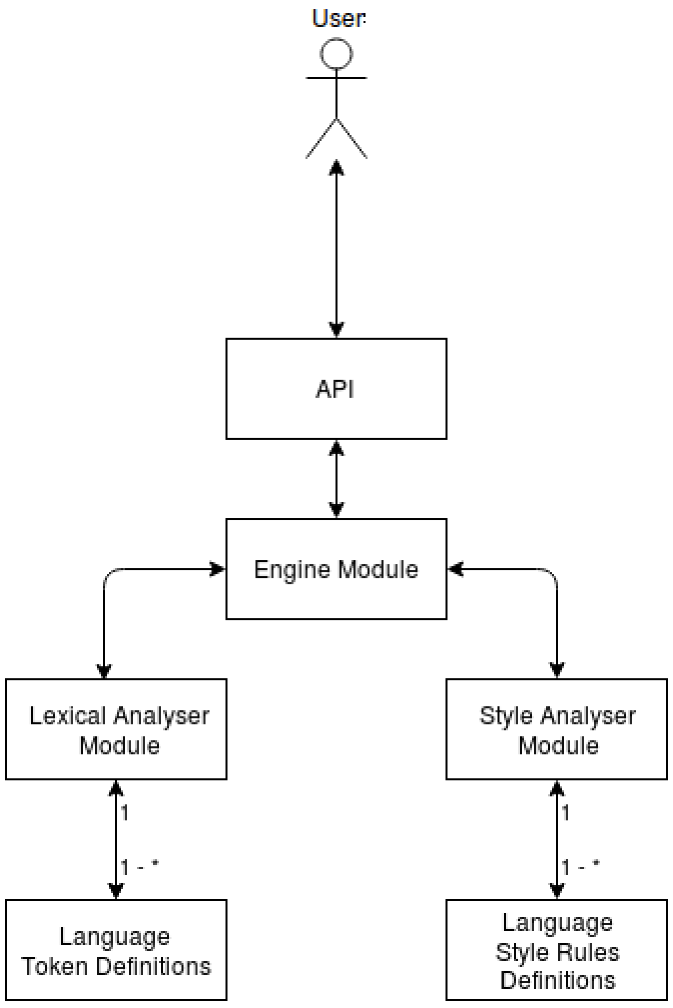
Requirement ID	8
Description	Sample Code Submission Page must display the result of its call to the Web-API in requirement 7 in an easy to understand fashion. This is to show off the usefulness of the project.
Criticality	Moderate.
Technical Issues	Must be able to receive and parse a response from the Web-API
Dependencies	Depends on requirement 7.

Requirement ID	9
Description	Provide download capabilities so that users may download the latest (or previous) versions of the library that is hosted on the website
Criticality	Non-Critical
Technical Issues	Standard file downloading from websites.
Dependencies	Depends on requirements 1-3 in order to have a working version of the project to download.

Requirement ID	10
Description	Should have easy to access and follow documentation hosted on the website
Criticality	Non-critical.
Technical Issues	N/A
Dependencies	N/A

4. System Architecture

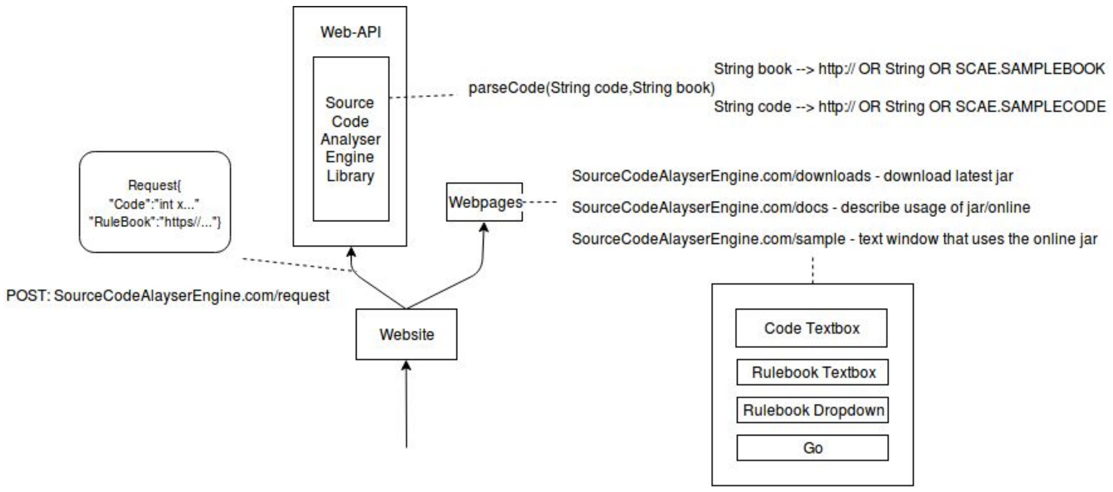
Source Code Analyser Engine Library System Diagram



In the diagram we can see the the System Diagram for the project library. The engine module is the component that keeps track of the progress of processing of the request, calls the other modules, and then return the result to the user.
The Lexical Analyser Module is a singular module that performs the Lexical Analysis of the submitted code. It uses the Token Definitions that were provided in the rulebook to perform this analysis. This way we only need one Lexical Analyser Module that can work for any language, so long as it is supplied the token definitions for that language.

The Style Analyser Module works in a similar fashion where it is language independent, and is fed in the language specific rules through the supplied rulebook.

Source Code Analyser Engine Website System Diagram


Web-API
Source Code Analyser Engine Library
parseCode(String code,String book)
String book --> http:// OR String OR SCAE.SAMPLEBOOK
String code --> http:// OR String OR SCAE.SAMPLECODE
Request{ "Code":"int x..." "RuleBook":"https//..."}
Webpages
SourceCodeAlayserEngine.com/downloads - download latest jar
SourceCodeAlayserEngine.com/docs - describe usage of jar/online
SourceCodeAlayserEngine.com/sample - text window that uses the online jar
POST: SourceCodeAlayserEngine.com/request
Website
Code Textbox
Rulebook Textbox
Rulebook Dropdown
Go

Note: anything linked to by a dotted gives a quick example of what the corresponding element will consist of / look like.

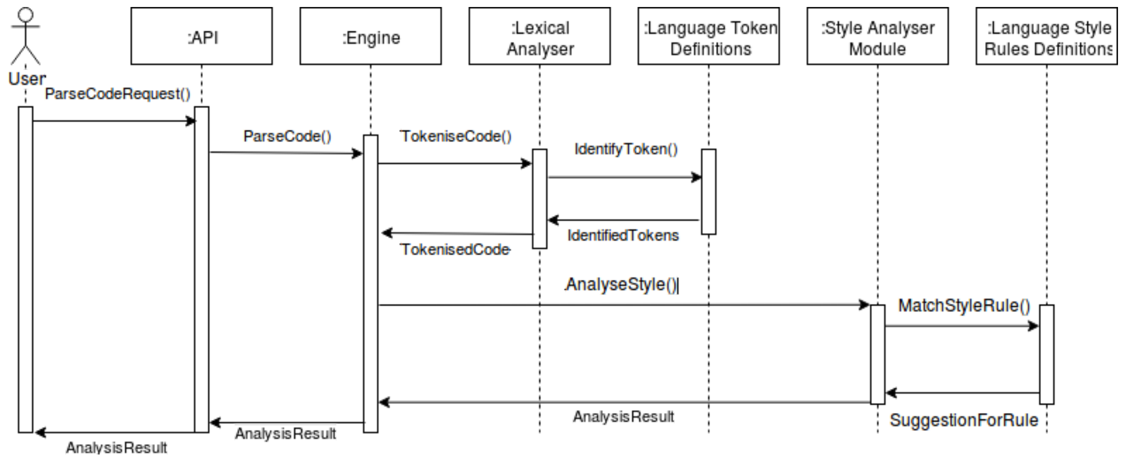
The above diagram aims to explain the architecture of the Source Code Analyser Engine Website.

On the right branch we have what a user can see when they go to the website. They can download the library, view documentation, or they can go onto the sample page. This page will allow the user the chance to see a quick demo of the system. They can input some code, select which rulebook to use, then see the result after the library has analysed the source code.

On the left branch we can see what happens when a request comes in to the Web-API. The Web API acts as a wrapper for the library. So the Web-API receives a request, it passes that request on to the Library API of the Source Code Analyser Engine, which works as outlined in the first diagram.

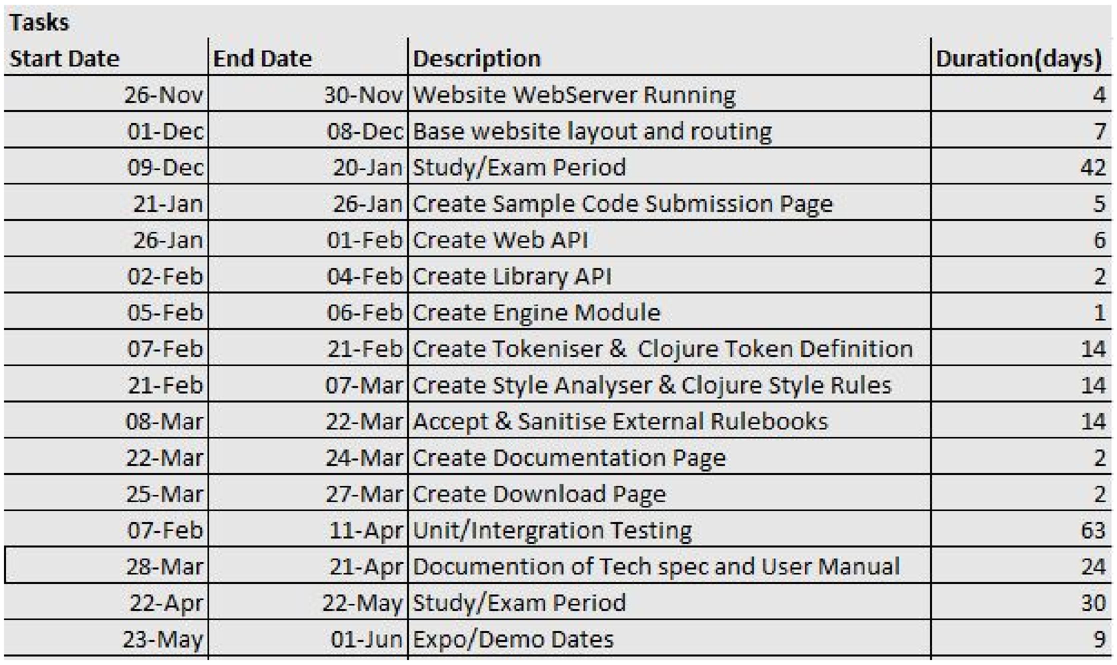
5. High-Level Design

Source Code Analyser Engine Library Sequence Diagram

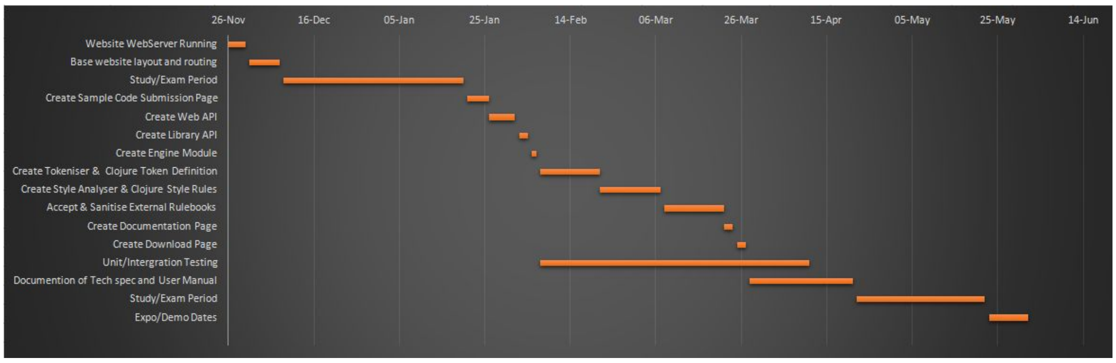


Here we have a Sequence Diagram for the Source Code Analyser Engine Library. We can see the workflow of a request coming in from a User to the Library API, how the Engine module receives that request and calls the other modules to process the request.

6. Preliminary Schedule



Tasks			
Start Date	End Date	Description	Duration(days)
26-Nov	30-Nov	Website WebServer Running	4
01-Dec	08-Dec	Base website layout and routing	7
09-Dec	20-Jan	Study/Exam Period	42
21-Jan	26-Jan	Create Sample Code Submission Page	5
26-Jan	01-Feb	Create Web API	6
02-Feb	04-Feb	Create Library API	2
05-Feb	06-Feb	Create Engine Module	1
07-Feb	21-Feb	Create Tokeniser & Clojure Token Definition	14
21-Feb	07-Mar	Create Style Analyser & Clojure Style Rules	14
08-Mar	22-Mar	Accept & Sanitise External Rulebooks	14
22-Mar	24-Mar	Create Documentation Page	2
25-Mar	27-Mar	Create Download Page	2
07-Feb	11-Apr	Unit/Intergration Testing	63
28-Mar	21-Apr	Documention of Tech spec and User Manual	24
22-Apr	22-May	Study/Exam Period	30
23-May	01-Jun	Expo/Demo Dates	9



7. Appendices

https://clojure.org/