Project Name: Yoshi
Design Description

Version 3.0

Revision History

Date	Version	Description	Author
2014-11-21	1.0	Design Draft of Yoshi Evolved	Rizwan Khalid
2014-12-01	2.0	Yoshi Vis	Rizwan Khalid
2014-12-28	3.0	Redone the structure of the document and the content of all parts, added diagrams. It is not based on the previous version.	Martin Anev

Table of Contents

1. INTRODUCTION 3
   - 1.1 PURPOSE OF THIS DOCUMENT 3
   - 1.2 DOCUMENT ORGANIZATION 3
   - 1.3 INTENDED AUDIENCE 3
   - 1.4 SCOPE 3
   - 1.5 DEFINITIONS AND ACRONYMS 3
     - *1.5.1 Definitions 3*
     - *1.5.2 Acronyms and abbreviations 4*
2. OVERALL DESCRIPTION 5
   - 2.1 PROJECT DESCRIPTION 5
     - *2.1.1 Interfaces 5*
   - 2.2 BACKGROUND DESCRIPTION 5
     - *2.2.1 Description of the "Yoshi" prototype 5*
   - 2.3 HIGH LEVEL DESCRIPTION OF THE DOMAIN 6
     - *2.3.1 Decisive Characteristics 7*
     - *2.3.2 Community Types and Yoshi 7*
     - *2.3.3 General functionalities 8*
   - 2.4 GOALS AND REQUIREMENTS 8
     - *2.4.1 Goals 8*
     - *2.4.2 Requirements 9*
3. SOFTWARE ARCHITECTURE & DESIGN 10
   - 3.1 OVERALL DECOMPOSITION 10
     - *3.1.1 Retrieval Component 10*
     - *3.1.2 Processing (Computation) Component 13*
     - *3.1.3 Visualization Component 14*
   - 3.2 SOFTWARE DESIGN 14
     - *3.2.1 Design technologies 14*
     - *3.2.2 Data flow Diagram 15*
     - *3.2.3 Exceptions and error handling 15*

REFERENCES: 16

1. Introduction

1.1 Purpose of this document

The purpose of this document is to expose the design decisions of the project delivered by Team Yoshi in the Distributed Software Development course done simultaneously in ‘Politecnico di Milano’ situated in Milan, Italy and ‘Mälardalen University’ situated in Västerås, Sweden.

1.2 Document organization

The document is organized as follows:

- Section 1, *Introduction*
- Section 2, *Overall Description*, describes different interfaces and high-level description of the domain
- Section 3, *Software Architecture and Design*

1.3 Intended Audience

The intended audience is:

- The customer of the project
- The supervisors of the project
- Yoshi Team
- All related stakeholders
- Any developer with interest to continue or improve the project

1.4 Scope

The document reveals what the description of the design, correlated to the Project Requirements Document. It contains information about the following aspects of the design of the project:

- Background and high-level description
- Software architecture, including conceptual design and system specification.
- External interfaces, including hardware, software and user interfaces.
- Detailed software design, including modules, data flows.

1.5 Definitions and acronyms

1.5.1 Definitions

Basic definitions of the document terms are defined in Table 1.

Definitions	
Keyword	**Definitions**
Community	Social unit of any size that shares common values.
Open Source Community	Community that develops open source software.
Compute a Community	The action of observing a community, measuring metrics for Social Communities (e.g. collaboration between community members, common projects, list of followers of the members, collaborators, contributors to projects). The outcome of the action is a decision – what type is the observed community.
Visualize a community	The action of observing the output of the decision "What type is a community?". A community can be visualized using text and images.
Software adaptor	In software engineering, the adapter pattern is a software design pattern that allows the interface of an existing class to be used from another interface.
Eclipse plug-in	Plugins are the smallest deployable and installable software components of 'Eclipse' software developing tool.
Community Decision Tree	Decision tree defining what type is an observed social community based on some characteristics. The Decision Tree is discussed in the document 'Uncovering Latent Social Communities in Software Development' [4].
Python	General-purpose, high-level programming language.
GitHub	Web-based repository hosting service, which offers all of the distributed revision control and source code management

Table 1. Definitions

1.5.2 Acronyms and abbreviations

Acronyms and abbreviations	
Acronym or abbreviation	**Definitions**
API	Application Programming Interface
G	Goal
FR	Functional Requirement
HTTPS	Hypertext Transfer Protocol Secure (communication protocol)
JSON	Java Script Object Notation (data interchange format)
OSS	Organizational Social Structures
CoP	Community of Practice
KC	Knowledge Communities
SC	Strategic Communities
IC	Informal Communities
LC	Learning Communities
PSC	Problem Solving Communities
NoP	Network of Practice
FN	Formal Networks
IN	Informal Networks
SN	Social Networks
WG	Work Groups
FG	Formal Groups
PT	Project Teams
HTML	HyperText Markup Language
PDF	Portable Document Format

Table 2. Acronyms and abbreviations

2. Overall description

This section describes the project, background description, high-level descriptions of the domain, goals and requirements of the project. These descriptions provide background knowledge for specifying the design and architecture of the software in the next sections of the document.

2.1 Project description

The project has two aims:

- Modify the existing “Yoshi” product.
- Develop missing components of the product.

A deliverable of the project is software product. Product’s name is “Yoshi Vis” and it is dependent on “Yoshi”[1] (Eclipse plug-in), providing a visualization and computation layer on top of “Yoshi”. In order to provide the adaptation, proper adaptors should be assigned to the output of “Yoshi”. Design and implementation of these adaptors will also be covered during this project.

2.1.1 Interfaces

- External interfaces – The software product does not provide any external interfaces.
- User interfaces
  - Input - the software product should have graphical user interface for the input from the user. The user is expected to specify the community that should be observed.
  - Output - the software product should provide visual output in HTML page form of text and images. The user should be able to visually understand and recon characteristics of observed communities.
- Hardware interfaces – The software product does not provide any hardware interfaces.
- Software interfaces – The software product should get as input the output of “Yoshi” (which is data.out file).

2.2 Background description

Damian A. Tamburri is a Ph.D. researcher at Vrije Universiteit - Amsterdam. Damian is the customer of Yoshi project. The customer wants a software for supporting social community awareness in open-source. He provided the source code of “Yoshi”. “Yoshi” has been built by Alexandra Leta, student in Information Sciences, Vrije Universiteit - Amsterdam for her graduation thesis project. The code was sent without any proper documentation. The team received only the thesis work of Alexandra Leta[5]. Yoshi, is an analytic software for open-source communities, helping different users better understanding the open-source community and getting a good understanding for research and for practice. The team for the project did not succeed in running the prototype. A meeting with the customer was scheduled, but the prototype was still not being able to run. According to the output files, the Yoshi prototype has worked before and it had already made some output in a text form (i.e. ‘data.out’ file).

2.2.1 Description of the “Yoshi” prototype

Based on observation of the code, “Yoshi” uncovers 6 characteristics to typify an open source-community: Structure, Situatedness, Dispersion, Informality, Formality, and Engagement, described in **part 3.2.1 Decisive characteristics**. The values should determine the community’s type and represent key characteristics that describe how the community is active and carrying out its work.

To evaluate these characteristics “Yoshi” should proceed as follows:

1) Establish the value of several indicators for said characteristics.
2) Ascertains that reference thresholds are passed, thus making the characteristics explicit.
3) Based on the characteristics and following an algorithm (from [2]) “Yoshi” should determine community type for an observed community.

By team’s observation and interviews with the customer, both parts deduced that the provided prototype was able to perform only part 1) in previous stable versions. From the team is not required to update the “Yoshi” prototype to stable version, but to develop parts 2) and 3) and provide output, based on the data that was computed before from stable versions of part 1). Figure 1 reveals Yoshi's software architecture using input-output control flow diagram.

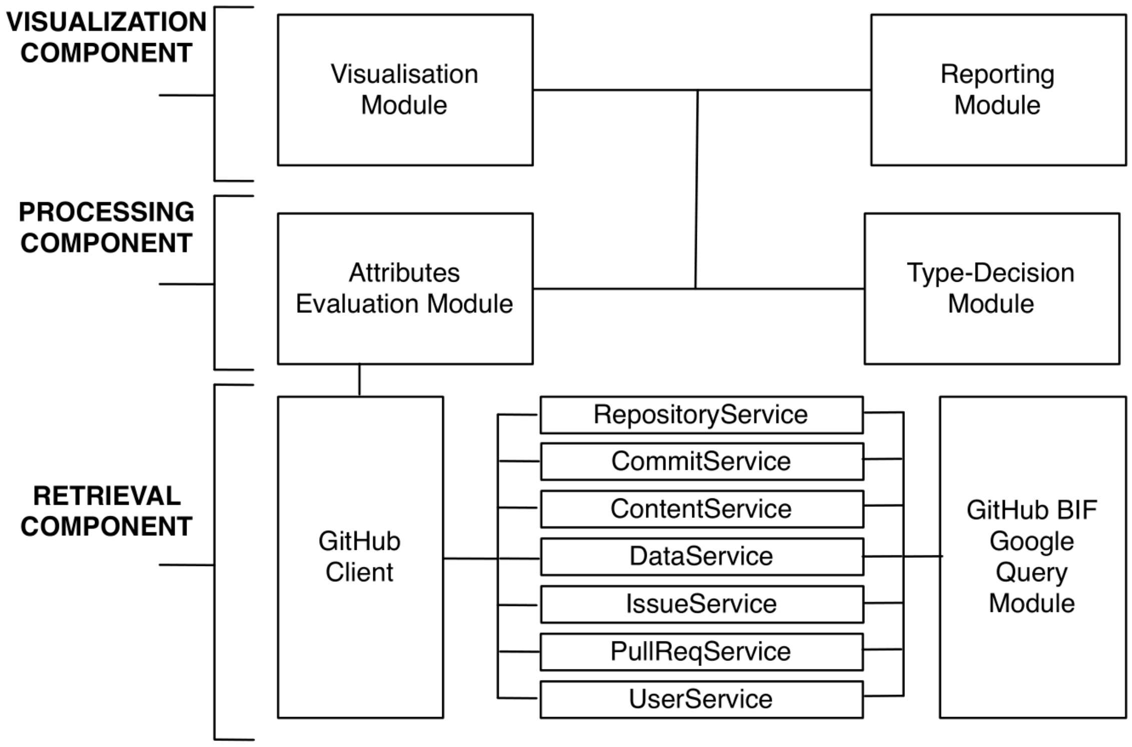


Figure 1. Yoshi high-level architecture

First, an information retrieval component is responsible for retrieving data with which YOSHI can perform its functions. This component retrieves data from two data sources: source code management systems and issue trackers from GitHub. The service classes (see middle part of Fig. 1)[1] provided by this API are used for retrieving information about the file structure of the repository, the members contributing to the project, issues associated to a repositories and their current status.
On top of the first layer the customer requires to be build the processing and visualization component. In order to achieve this, proper formatting of the current output should be done.

2.3 High level description of the domain

The software product that will be delivered is "Yoshi Vis". "Yoshi vis" is containing the Processing and Visualization layer of the "Yoshi" project. "Yoshi" and "Yoshi Vis" mutual coexistence intend to help people to measure, compute and study social community types by visualizing metrics and characteristics of the observed community. For instance, it will allow users to measure social and organizational characteristics of open-source communities (e.g. structure, formality, informality) measured by different metrics (e.g. collaboration between community members, common projects, list of followers of the members, collaborators, contributors to projects). The possible types of communities are: Community of Practice (CoP); Knowledge Communities (KC); Strategic Communities (SC); Informal Communities (IC); Learning Communities (LC); Problem Solving Communities (PSC); Network Of Practice (NoP); Formal Networks (FN); Informal Networks (IN); Social Networks (SN); Work Groups (WG); Formal Groups (FG); Project Teams (PT). [3] For more information on the domain, refer to the Project Requirements Document [6] and the references [2][3][4].

2.3.1 Decisive Characteristics

The followings are six key organisational, operational and socio-technical characteristics for open-source communities: **Structure** (Is there a non-trivial structure within the observed people?), **Situatedness** (Do all community members exhibit the same location?), **Dispersion** (Do all community members exhibit a different location?), **Informality** (Do community members require informal status evaluation?), **Formality** (Do all community members require formal status evaluation? – mutually exclusive with Informality), **Engagement** (Does the community effectiveness depend on people' engagement?). How the characteristics are represented by the observed metrics is an issue to be resolved. A Community Decision Tree has been defined in [4] and is represented on Figure 2. For Open-source communities (and respectfully for the project) is related only the right-hand side of the tree, divided by Dispersion (i.e. CoP, IN,FN, NoP, IC).

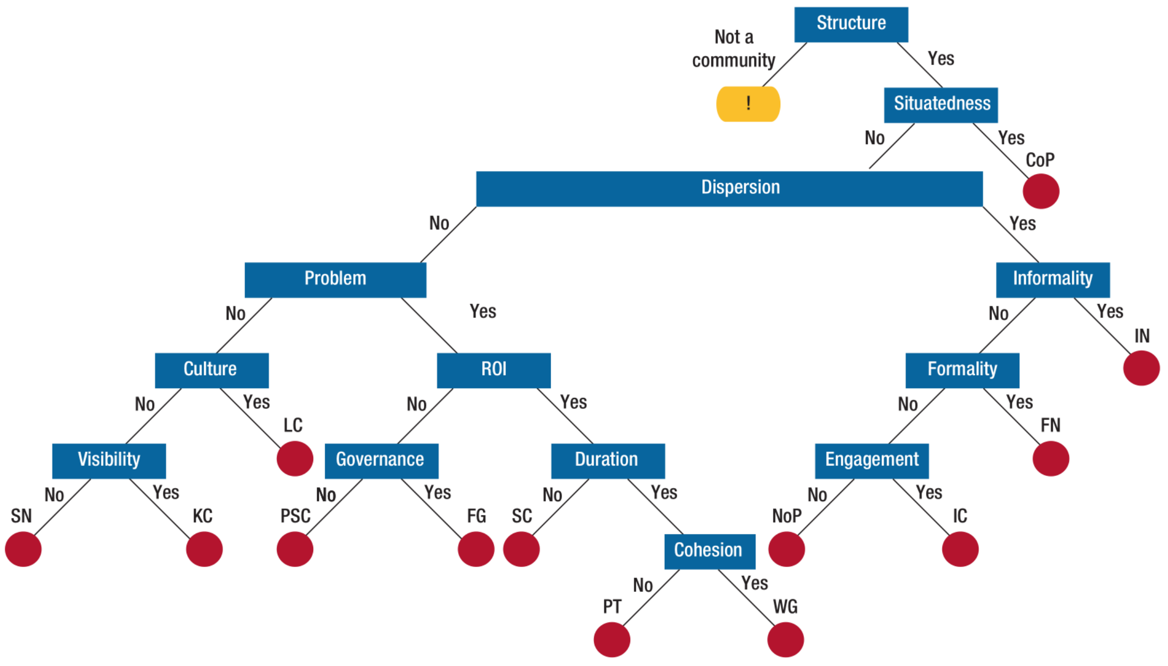


Figure 2. Community Decision Tree

2.3.2 Community Types and Yoshi

The "Yoshi" prototype has no documentation that could be appended to this document. Ideas of how it works and which communities it could measure should be discovered in the process of work, by its code and behavior. Up to date neither the team nor the client succeed in running the "Yoshi" prototype, "Yoshi Vis" is extending the output of previous stable versions of "Yoshi" as well as future versions that could provide the same output.
From theoretical point of view every community shall be uniquely defined by the algorithm (representing the Community Decision Tree – Figure 2) [2], but in practice the customer said "It is normal that a structured set of people (i.e. community) could blend in more than one types. To establish this, mind the precedence of the characteristics." The algorithm used for the project, presented in **part 3.1.2**, has been done together with the customer and it defines only the right part of the Community Decision Tree divided by the attribute Dispersion. The reason for this limitation is the data provided from "Yoshi" prototype. "Yoshi" and "Yoshi Vis" can establish the following types: CoP, IN,FN, NoP, IC.

2.3.3 General functionalities

"Yoshi" and "Yoshi Vis" should provide general functionalities for managing:

- Retrieval
  "Yoshi" is responsible for connecting to social communities and to gather metrics from their repositories. This component retrieves data from two data sources: source code management systems and issue trackers.
- Processing
  "Yoshi Vis" is responsible for evaluating metrics using data available from the retrieval component and compute community typification [3] by matching the type with the decisive attributes.
- Visualization
  "Yoshi Vis" is responsible for visualizing the metrics (using a text form) and the community type on a decision tree. [4]

2.4 Goals and Requirements

This part describes the goals and functional requirements that affect the software and their correlation.

2.4.1 Goals

In this part are discussed the general goals of the project, the goals that the project has for modifying 'Yoshi' and the goals that the project has for building 'Yoshi Vis'.

2.4.1.1 General

The following are the general goals of both "Yoshi" and "Yoshi Vis" components.
[G0] Allow users to observe community by gathering organizational characteristics.
[G1] Allow users to measure community social and organizational characteristics.
[G2] Allow users to study type and characteristics against performance metrics.
[G3] Allow users to understand the characteristics of an observed community.

2.4.1.2 Yoshi

The following are the project's goals for modifying "Yoshi".
[G4] Format the current output.

2.4.1.3 Yoshi Vis

The following are the project's goals for building "Yoshi Vis".
[G5] Define the type of the observed community based on the metrics.
[G6] Provide output with the defined social types and motivate the decision.

2.4.1.4 Description of the Goals

- [G0] Allow users to observe community by gathering organizational characteristics. This is a general goal for "Yoshi" and "Yoshi Vis". As "Yoshi" is not in a stable version, it cannot achieve the goal within the domain of the project. The project's aim is not to achieve [G0], but to fulfil the functional requirements, which will lead to fulfilments of the other Goals (G[1],...,G[6]). The team is not required to update "Yoshi" to runnable version.

- **[G1]** Allow users to measure community social and organizational characteristics. This goal is one of the main purposes of the project. The users should be able to measure the organizational characteristics gathered from the retrieval component.

- **[G2]** Allow users to study type and characteristics against performance metrics. The users should be able to understand what are the correlations between the type of the observed community and the metrics gathered.

- **[G3]** Allow users to understand the characteristics of an observed community.

The characteristics of the observed community should be expressed in a comprehensible way, every value should be mattered to its metric.

- **[G4]** Format the current output.
  As the current output is in violation with Goal 3, the customer requires that the output should be changed, in order to be able to understand what the "Yoshi" prototype is giving as output.

- **[G5]** Define the type of the observed community based on the metrics.
  This is the essential part of the system. This goal defines the purpose of the system.

2.4.2 Requirements

The following part discusses the functional requirements needed to satisfy the goals.

2.4.2.1 Functional requirements for the project related to "Yoshi"

[FR1] The current output of Yoshi should be prepared for adaptation.

2.4.2.2 Functional requirements for the project related to "Yoshi Vis"

[FR2] The product should be able to provide proper adapter for the formatted output of Yoshi.
[FR3] The product should be able to take decision what type is an observed social community based on the received metrics from "Yoshi".
[FR4] The product should be able to visualize as output the formatted output of Yoshi.
[FR5] The product should be able to visualize as output the evaluated decision of the community type in text form.
[FR6] The product should be able to visualize as output motivation of the taken decision.
[FR7] The product should be able to visualize as output the community type on the Community Decision Tree in graphic form.

2.4.2.3 Description of the Functional Requirements

- **[FR1]** The current output of Yoshi should be prepared for adaptation.

This FR should be fulfilled, in order to satisfy G4. The output should be re-formatted in JSON standards format.

- **[FR2]** The product should be able to provide proper adapter for the formatted output of Yoshi.

This FR should be fulfilled, in order to satisfy G4, G5. The input of the "Yoshi Vis" should follow the JSON standards, in order to be able to connect to the output of "Yoshi"

- **[FR3]** The product should be able to take decision what type is an observed social community based on the received metrics from "Yoshi".

This FR should be fulfilled, in order to satisfy G5. The type should be at least one from part 2.3.1 Community Types.

- **[FR4]** The product should be able to visualize as output the formatted output of Yoshi.

This FR should be fulfilled, in order to satisfy G1,G2, G3.

- **[FR5]** The product should be able to visualize as output the evaluated decision of the type in text form.

This FR should be fulfilled, in order to satisfy G3, G5.

- **[FR6]** The product should be able to visualize as output motivation of the taken decision.

This FR should be fulfilled, in order to satisfy G2.

- **[FR7]** The product should be able to visualize as output the type on the Community Decision Tree in graphic form.

This FR should be fulfilled, in order to satisfy G3, G5.

Software Architecture & Design

This section describes the overall decomposition of the components with their specific designs; diagrams of the components and data flow; error and exceptions handling.

2.5 Overall Decomposition

As described in part *2.2.1 Description of the "Yoshi" prototype* and on Figure 1, the system consists of three main components Retrieval, Processing and Visualization. These three components are abstract layers that are laid one above other in the same order. An abstract view of the layers is represented in Figure 3. The provided prototype "Yoshi" from the customer is representing the Retrieval Component, the upper layers are parts to be developed for the purposes of this project as "Yoshi Vis". In order to be able to implement the upper layers – proper adaption is needed as currently the Retrieval Component's output is a set of random numbers, which is not suitable for input for the upper layers. Modification of the output should be done, in order to deploy the upper layers on top.

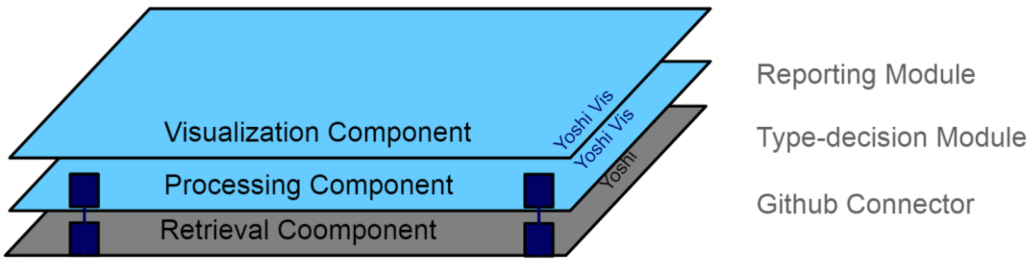


Figure 3. Abstract Layers decomposition

2.5.1 Retrieval Component

The Retrieval component (also known as the prototype "Yoshi") is the first (lowest) abstraction layer of the system. The component has the following characteristics [2]: "First, an information retrieval component is responsible for retrieving data with which YOSHI can perform its functions. The retrieval component automates the retrieval of data from public repositories of projects hosted on Github, using GET requests form GitHub Archive to obtain publicly available data. This component retrieves data from two data sources: source code management systems and issue trackers. GitHub Archive is a project to record the public GitHub timeline, archive it, and make it easily accessible for further analysis and this archive dataset can be accessed via Google BigQuery. This data is used to compute attributes values related to the software development process and study targeted open-source software development communities. GitHub Java API is the library used for communicating with the GitHub API, supporting the GitHub v3 API. The client package contains classes that communicate with the GitHub API over HTTPS and the client is responsible for converting JSON responses to appropriate Java model classes. The package contains the classes that invoke API calls and return model classes representing resources that were created, read, updated, or deleted. The service classes (see middle part of Fig. 1) provided by this API are used for retrieving information about the file structure of the repository, the members contributing to the project, issues associated to a repositories and their current status." The output of the Retrieval component is the measured metrics. The output is given only in text form in the data.out file. Up to date this file contains only values without explanation of what metrics are these values represented on Figure 4.

```
// Qualities //
Khan/khan-exercises 1197.738 0.003 0.444 true 24.137 2.773 4 0.170
0.011 0.356 9.341 0.395 0.508 10.147 0.326 2.303
SignalR/SignalR 247.405 0.432 0.032 true 105.0 1.839 1 0.554 0.021 0.25
64.711 0.461 0.381 6.653 0.286 1.990
…
```

Figure 4. Output of Retrieval Component (Data.out)

The team has retrieved the metrics of the values and they are presented on Figure 5 in JSON format. Not all of the metrics are needed for the upper layers, some of them were used by the initial developer of the prototype (Alexandra Leta) for defining whether an observed set of people has the attribute Structure (see Figure 2) as the lack of this attribute would mean no further evaluations are needed as the observed set of people cannot be defined as community. This, on the other hand, assures that the instances of observed communities that are forwarded to the upper layers are for sure with observed structure (i.e. having the characteristic Structure). The formatted output of the Retrieval component is used for input of the upper layer (i.e. Processing Component).

```
{
        Data Quality:[
              {
                    “RepoOwner”: String,
                    “RepoName”: String,
                    “monthlyEventsStdDev”: Float,
                    “ReopenedIssuesPercentage”: Float,
                    “milestonesPerDay”: Float,
                    “hasWiki”: Boolean,
                    “avgeUserCommits”: Float,
                    “avgeFileContributors”: Float,
                    “activeMembers”: Integer,
                    “avgUserCollaborationFiles”: Float,
                    “avgCCFollowers”: Float,
                    “avgCommiterLongivity”: Float,
                    “avgSubscriptions”: Float,
                    “avgBloggers”: Float,
                    “avgClusteringCoefficient”: Float,
                    “avgDegree”: Float,
                    “modularity”: Float,
                    “closenessCentrality”: Float
                    },
                    {Instance 2},
                    {Instance 3},
                    …
                    {Instance n}
              ]
        }
        {
              Data Community:[
                    {
                          “RepoOwner”: String,
                          “RepoName”: String,
                          “avgMilestonesPeriod”: Float,
                          “hierarchyDegree”: Float,
                          “avgCollabProject”s: Float,
                          “inPercentageContributorCompanies”: Float,
                          “lackOfGovernance”: Float,
                          “avgDistance”: Float,
                          “standardDevDistance”: Float,
                          “avgCulturalDistance”: Float,
                          “selfSimilarity”: Float,
                          “activeMembers”: Integer,
                          “uniqueCommenterExists”: Boolean,
                          “highEngagement”: Boolean,
                          “avgDistance”: Float,
                          “avgCulturalDistance”: Float
                    },
                    {Instance 2},
                    {Instance 3},
                    …
                    {Instance n}
              ]
}
```

Figure 5. Formatted Data.out file output

2.5.2 Processing (Computation) Component

The Processing or Computation component is the second abstract layer of the system. It takes as input the output of the Retrieval component. This layer is responsible for matching of the provided data compared to pre-defined thresholds, provided from the customer. The algorithm is presented with a block-diagram in Figure 6. The thresholds of the algorithm are shown in Figure 7. Only the mentioned metrics are needed for defining a community type, while the others that are used in the retrieval component serve for understanding whether there is structure (i.e. whether there is structured community). The algorithm's output is the type(s) of the observed community. The output of the Processing component is used for input of the upper layer (i.e. Visualization Component).

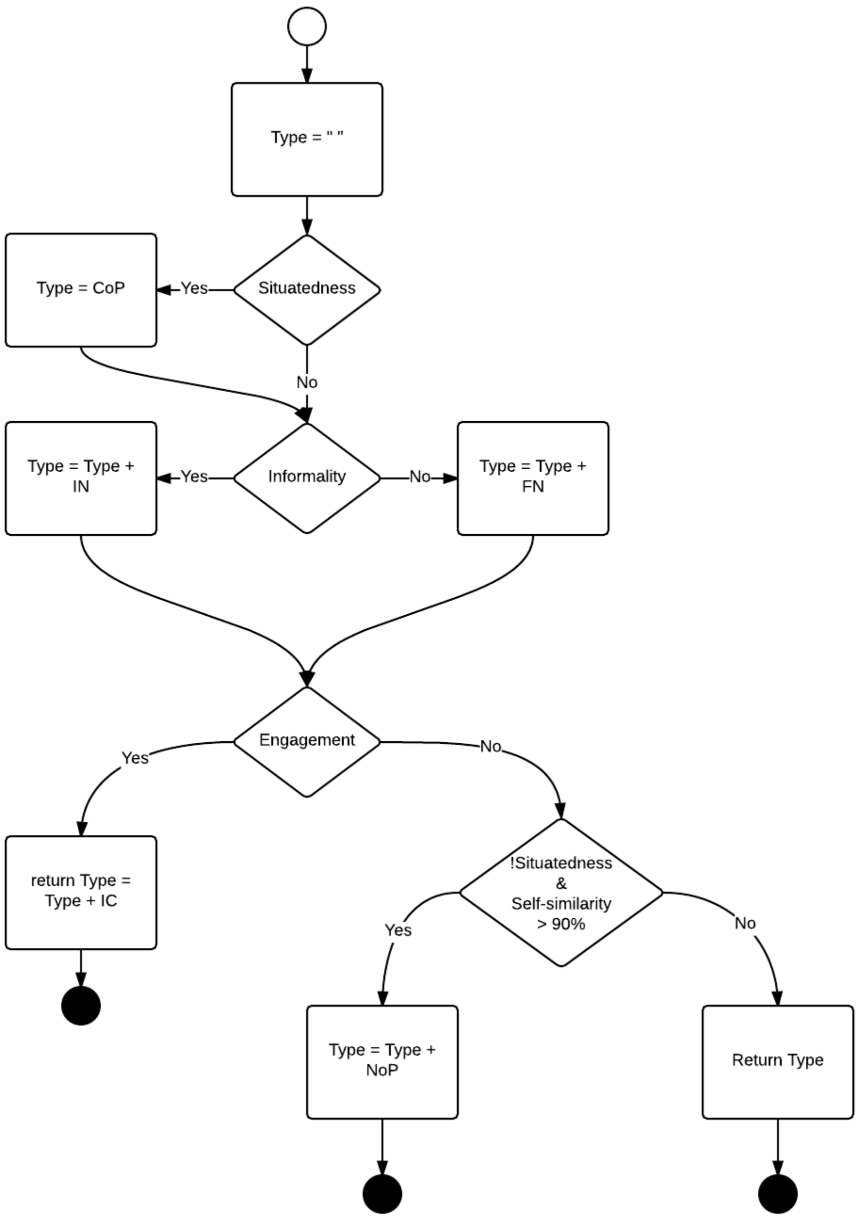


Figure 6. Algorithm Block Diagram

```
Geographical Distance = 4000
Cultural Distance = 15.0
Average Milestones = 0.1
Hierarchical Degree = 0.8
Self Similarity = 0.9
Average Subscription = 9.794
Average User Commits = 14.22
Average User Collaboration Files = 0.200
Average File Contributor = 1.220
```

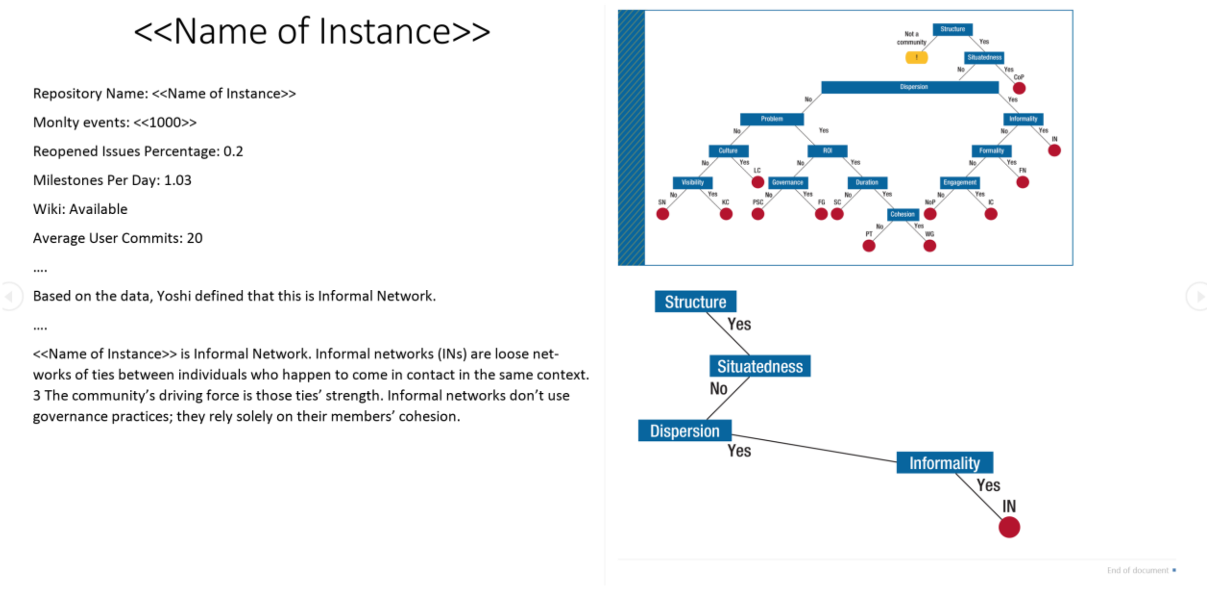
Figure 7. Algorithm Block Diagram

2.5.3 Visualization Component

The Visualization component is the third abstract layer of the system. It takes as input the output of the Processing component. This layer is responsible for visualizing to the user:

- In text form:
  - The decision of the type of the observed community taken in the Processing component
  - Motivation for the taken decision based on the metrics
  - Relevant data about such types of communities
- Providing in graphical form
  - The Community Decision Tree
  - The paths taken on the tree (as cut-offs) in order to deduce the type of the community.

A mock-up prototype of the output of the Visualization part is presented on Figure 8. The realization of the output is done in HTML page, which could be manipulated to other forms of representation (e.g. PDF).



<<Name of Instance>>

Repository Name: <<Name of Instance>>

Monlty events: <<1000>>

Reopened Issues Percentage: 0.2

Milestones Per Day: 1.03

Wiki: Available

Average User Commits: 20

....

Based on the data, Yoshi defined that this is Informal Network.

....

<<Name of Instance>> is Informal Network. Informal networks (INs) are loose net-works of ties between individuals who happen to come in contact in the same context. 3 The community's driving force is those ties' strength. Informal networks don't use governance practices; they rely solely on their members' cohesion.

Figure 8. Output of "Yoshi Vis"

2.6 Software Design

2.6.1 Design technologies

Based on the project requirements and design of the "Yoshi" prototype. The team has decided that "Yoshi Vis" will be developed with Python v.3.4. Distributed version control technology is GitHub. The repository for the project should be developed in https://github.com/NinjaTrappeur/yoshi-viz.

2.6.2 Data flow Diagram

The data flow is described on Figure 9. First the data is retrieved from the GitHub repositories, from the Retrieval Component, in this sense the data is the retrieved metrics. Then the data is passed to the Processing component, which is responsible for deciding what type/s is the observed community. Then the decision and related metrics are passed to the Visualization component, which is responsible for representing them as Output.

2.6.3 Exceptions and error handling

According to the predefined algorithm elaborated together from the team and the customer, presented on Figure 6, a community might be typified as more than one type. While from theoretical point of view this should be not feasible, in practice this is often observed and normal for communities. For this reason, in such cases it should be specified in the output that the community has more than one type. The related types should be presented and additional information about both of this types should be given in the output.

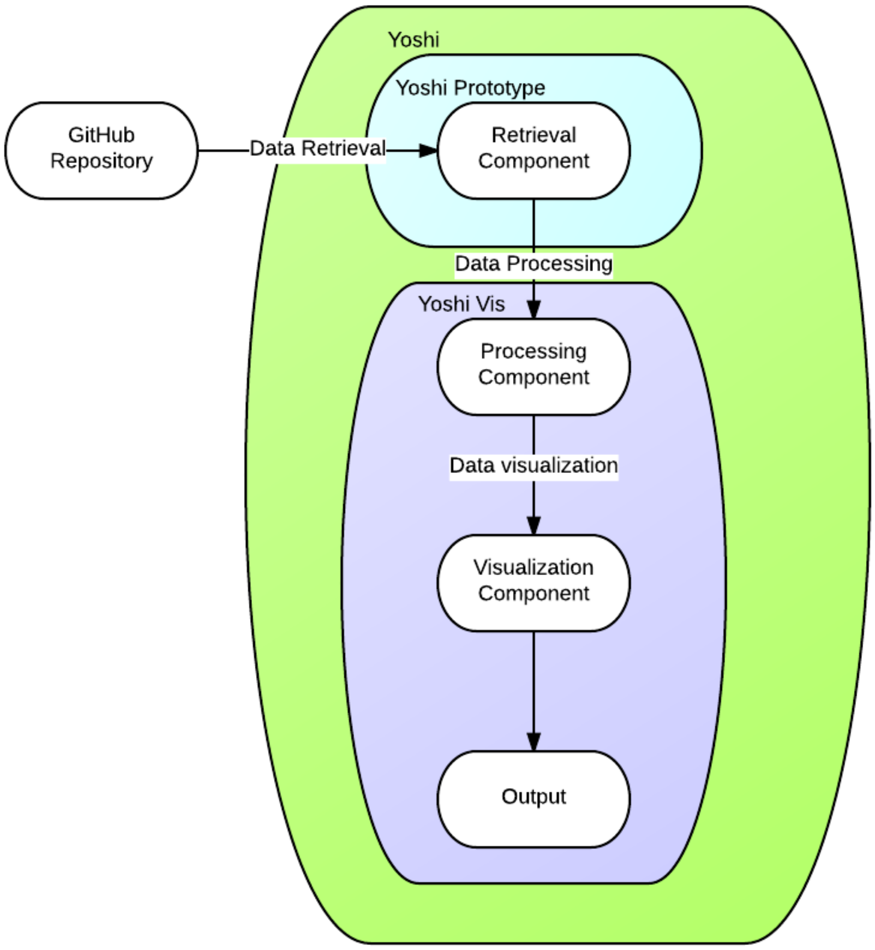


Figure 9. Data Flow Diagram

References:

[1] Yoshi - https://github.com/maelstromdat/YOSHI

[2] '*"Let me measure my self!" Said Open-Source'* - D.A. Tamburri, E. di Nitto, P.Lago

[3] 'Organizational Social Structures for Software Engineering' - D.A. Tamburri, P. Lago, H.V. Vliet

[4] 'Uncovering Latent Social Communities in Software Development' – D.A. Tamburri, P. Lago, H.V. Vliet

[5] 'Discovering Open-Source Community Types: An Automated Approach' – A. Leta

[6] 'Project Requirements' - Yoshi - http://www.fer.unizg.hr/rasip/dsd/projects/yoshi/documents