

# Development plan

Zohour ABOUAKIL  
Sofia BOUTAHAR  
David COURTINOT  
Xiaowen JI  
Fabien SAUCE

Recherche de motifs dans un code C++ à l'aide de la logique temporelle

# Sommaire

<b>I</b>	<b>Project description and objectives test test</b>	<b>2</b>
I.1	Surroundings of the project . . . . .	2
I.2	Project description . . . . .	2
I.2.1	Main idea . . . . .	2
I.2.2	Related technologies . . . . .	2
I.2.3	Project parts . . . . .	2
I.2.4	To conclude . . . . .	3
I.3	Final project . . . . .	3
I.3.1	Define priorities . . . . .	3
I.3.2	Deliverable documents . . . . .	3
<b>II</b>	<b>Project organization</b>	<b>4</b>
II.1	Role definition . . . . .	4
II.2	Development organisation . . . . .	5
II.2.1	Usage of Scrum method . . . . .	5
II.2.2	Team repartition approach . . . . .	5
II.3	Tasks organisation . . . . .	5
II.3.1	Tasks definition . . . . .	5
II.3.2	Planning . . . . .	6
<b>III</b>	<b>Risk management</b>	<b>8</b>
<b>IV</b>	<b>Code management</b>	<b>10</b>
IV.1	Quality management . . . . .	10
IV.1.1	Automated coding style checks . . . . .	10
IV.2	Test strategy . . . . .	10
IV.3	Configuration management . . . . .	10
<b>V</b>	<b>Appendices</b>	<b>11</b>

# Partie I

## Project description and objectives test test

### I.1 Surroundings of the project

Le projet long à l'ENSEEIH Organisation du projet

Le client c est qui ?? Les noms, leurs fonctions, les motivations du projet

Nos motivations – pas sur

### I.2 Project description

#### I.2.1 Main idea

#### I.2.2 Related technologies

- Coccinelle
- Clang

#### I.2.3 Project parts

- Parser
- CTL
- Model checking

**I.2.4 To conclude**

## **I.3 Final project**

**I.3.1 Define priorities**

**I.3.2 Deliverable documents**

## Partie II

# Project organization

### II.1 Role definition

Project manager

Quality manager

Test manager

Test manager

Configuration manager

Documentation manager

## Chain development

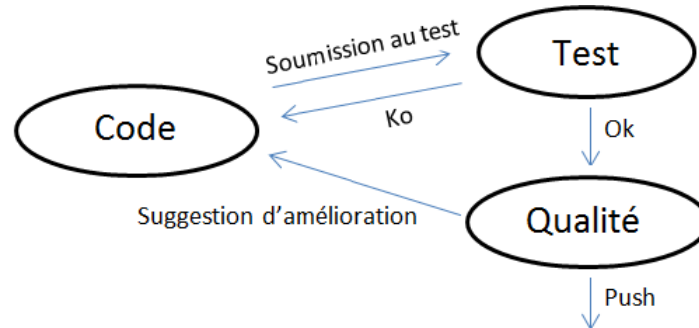


Figure II.1 - Schéma descriptif de la chaîne de développement

## II.2 Development organisation

To secure our evolution we can use :

### II.2.1 Usage of Scrum method

We will try to use Scrum method, which is actually widely used, and recognised for its effectiveness. At first, we will define a *product backlog* containing all desired functionalities in the final product. In fact, this report is also a part of *product backlog*. Next, we will divide the project into three *sprints* (which means iterations). A *sprint backlog* is defined for each *sprint*, including all we need to realise at the end of an iteration. Each *sprint* lasts two weeks and lies in improve the software incrementally, so that it is close to *product backlog*.

At the end of each *sprint*, we will organise a meeting, in order to review the progress and propose improvements or modifications of planning, but in the process of a *sprint*, we cannot modify the *sprint backlog*. At last, each day starts with a *scrum meeting*, on the meeting, each team member present his objective of the day and his actual difficulties.

### II.2.2 Team repartition approach

We will use an approach inspired by the XP (extreme programming) method. In fact, we found it unnecessary that the team members work separately, and we found it excellent to work in pairs, in order to prevent errors and bias of the program structure, so that we can save times of testing and debugging. So four of us work in pairs and the last one works individually. The group will changes as the tasks are completed.

## II.3 Tasks organisation

### II.3.1 Tasks definition

Sprint 1 backlog :

- AST parsing of procedure C++ code
- CFG conversion from parsed AST
- Model checking with simple properties

—

Sprint 2 backlog :

- AST parsing of object oriented C++ code
- CFG conversion from parsed AST
- Model checking with simple criteria

—

Improved CFG conversion from parsed AST ; Model checking with complex criteria ;

### **II.3.2 Planning**

[illegible]



## Partie III

# Risk management

Date	Risk description	Consequences	Type of risk	Probability (1-5)	Impact level (1-5)	Weight	Preventive mesure
27th, January 2015	Communication problems : lack of communication, misunderstanding, etc	Unproductive group, non-respect of the interfaces necessary to compatibility	Human resources	5	5	25	Be sure we agreed with our teammates before starting a part
27th, January 2016	Underestimation of the development time	Deadline exceeded / late delivery	Scope of project	4	5	20	Supervisor able to switch from one task to another and have a global vision
27th, January 2017	Wrong or inappropriate assumptions during the analysis	Unexpected edge cases difficult to handle with our model	Development method	5	4	20	Validate the conception by the client
27th, January 2018	Customer's requirements not respected	Product not accepted by the client	Client requirements	4	4	16	Having some meetings with the clients every weeks and making them validate our steps
27th, January 2019	Bad design choices at the beginning, issues to make the model evolve, corner cases...	Problem to make the project evolve, waste of time to readapt the conception to the new requirements	Quality	3	5	15	Allocate several days to conception and ensure everyone is convinced by the design
27th, January 2020	Health problems : a member of the team getting sick, etc	In the best case, redefine the other team member role. Otherwise, the product will be late.	Scope of project	2	5	10	Flexible schedule
27th, January 2021	Underestimation of the learning curve, different time learning among the team	Delays, different rhythms for the various parts of the project	Scope of project	3	3	9	Create balanced teams (people better trained with people less trained)
27th, January 2022	Appearance of bugs that we cannot fix	Unable to meet certain requirements	Quality	2	4	8	Restart the task with another approaches and change the people affected to this task

## Partie IV

# Code management

### IV.1 Quality management

#### IV.1.1 Automated coding style checks

For ensuring that our coding rules are respected and evaluate the quality of our sources, we have used a tool called *Scalastyle* that enables, using an easy-to-use xml configuration file, to check some properties on a Scala code. Combined with a specific pulgin, this can be use to generate warnings or errors in the IDE the developer is using. Our settings can be found in appendix A.

### IV.2 Test strategy

### IV.3 Configuration management

Partie V

Appendices