Development plan - Version 2

Zohour Abouakil
Sofia Boutahar
David Courtinot
Xiaowen Ji
Fabien Sauce

Reference : model-checking.dev-plan
January 29^{th} 2015

Signatures

Quality responsible :
Clients :

Table des matières

Part I

Project description and objectives

I.1 Surroundings of the project

I.1.1 What is "Projet long" ?

To get our ENSEEIHT engineering diploma, we are required to take part in a project called "Projet long" in teams of five students to work on a common project.

The project started on January 19, and will last eight weeks. It ends up with a defense in which we promote our work in front of a jury which evaluates us against different aspects :

- Project management and organization
- Technical accomplishment
- Report and defense presentation
- English evaluation

All over the project, we have to work side by side with the client for whom we have to deliver, at the end of the project, a product that suits their expectations. Furthermore, we are also supervised by Mr. Jean-Francois COIFFIN. He is in charge of helping us through his experience in the project management and organization.

I.1.2 Who are our clients ?

The subject that we work on was made by Mr. David Doose and Mr. Julien Brunel, two researchers at Aerospace Lab ONERA. They are working on robot development using C++ language. This is why they are in the need of a model checking tool to assert some properties on their embedded system's code. The client already has a similar product, called Coccinelle. However, Coccinelle is limited to looking for patterns in C code.

I.1.3 What made us choose this subject ?

We chose to work on that project because of the originality of the subject, since it is mixing theoretic computer science and technical advanced principles. Moreover, studying model checking and temporal logic to assert properties on a source code was a topic that we studied in ENSEEIHT courses. This project is an opportunity to apply this theory and dive deeper into it.

I.2 Project description

I.2.1 Main idea

The client is waiting for a prototype that allows a search of patterns on a C++ code. The patterns will be expressed in terms of temporal logic properties.

I.2.2 Project parts

The project can be divided into two main parts :

- **Parser :** takes a C++ code file as an input and transforms it into a data structure -a graph called CFG, as in Control Flow Graph- to explore every possible execution trace
- **Model checking algorithm :** this algorithm takes a property to check and tries to find the nodes in the graph that verify it.

After realizing both parts separately, we have to make it work togetger : this is the final product. The following graph shows the main steps in our project.

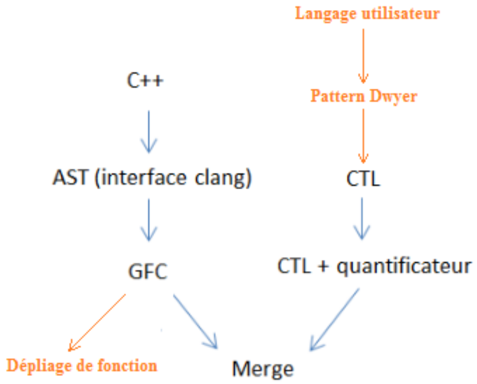


Figure I.1 - Project parts

The tasks colored in orange represent possible (and optional) extensions of the product.

I.2.3 Related technologies

To simplify the C++ code parsing we will use an intermediate tool : Clang. It is a compiler front-end for C, C++, Objective-C and Objective-C++ and developed by Apple. It takes a C++ file as input and outputs a file describing an AST (Abstract Syntax Tree).

I.3 Final project

I.3.1 Functionalities

Features expected by the customer :

- implementation of a parser for the AST generated by Clang
- a conversion from AST to a Scala model representation of the code in terms of graph control flux
- independently of the two preceding items , algorithms for the analysis of CTL properties (Computation Tree Logic) on some control flow graphs
- adding to the previous item some quantifiers such as "exists". This is known as CTL-V (CTL with quantified variables).

The following extensions can be added afterwards :

- unfolding function calls on a given depth
- creating a user language to interface with the system

I.3.2 Deliverable documents and define priorities

Deliverables expected by the client are :

- Documented source code in Scala language
- Test strategy

Deliverables expected by the supervisor and ENSEEIHT are :

- Report in PDF format
- Development plan in PDF format
- Presentation supports

Part II

Project organization

II.1 Roles definition

Project manager

The project manager is primarily concerned about communications with the industrial and the customers. It has a leading role in the organization and planning of the tasks.

Supervisor

The supervisor has a global technical view of the project. He supervises the advancement of simultaneous tasks. Otherwise, he can rearrange groups and objectives if an unforeseen occurs. The supervisor has also to participate in coding or documenting an assigned task. Nevertheless, it is not his primary function.

The team supervisor can change from one week to another.

Quality manager

The quality manager is in charge of checking that every deliverable documents meets the quality standards. In other words, any produced code will pass under the watchful eye of the quality manager before being validated. He also ensures the quality and consistency of all documents produced by the team.

Test manager

The test manager is responsible of the validation and testing in global environment written by the developers (each developer has its own set of unit tests). He does not only run tests, he also determines whether the tests are complete or not (code coverage).

Configuration manager

The configuration manager should take care of every tool we are going to use, make some choice about which tools are better than other (example : Scalastyle, an Eclipse plugin that we use for automated quality checs). In particular he will handle the installation and the follow up of a version tool as Github for example.

Chain development

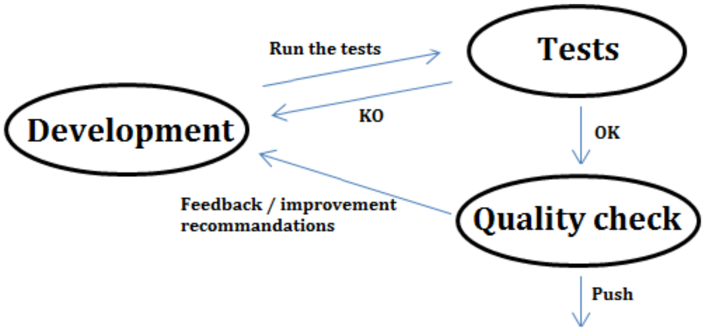


Figure II.1 - Schéma descriptif de la chaîne de développement

II.2 Development organisation

To secure our evolution we will use some methods inspired of famous project management techniques.

II.2.1 The Scrum method

We will try to use the Scrum method, which is widely used, and recognized 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 the *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*. To finish, 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. Considering the amount of code that we will have to write, we find it unnecessary that the five team members work separately, and we consider as excellent to work in pairs, in order to prevent errors and bias of the program structure, so that we can save times in testing and debugging. As a consequence, four of us will work in pairs and the last one works individually. The groups repartition may change 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

Sprint 3 backlog :

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

II.3.2 Planning

Task list	21-janv	22-janv	23-janv	26-janv	27-janv	28-janv	29-janv	30-janv	02-févr	03-févr	04-févr	05-févr	06-févr	09-févr	10-févr	11-févr	12-févr	13-févr
Redaction of specificaitons				Z D X	Z D X	Z D X												
Training:																		
- Scala	Z X F S	Z X F S	Z X F S	Z X F S	Z X F S	Z X F S	Z X F S	Z X F S										
- CTL and model checking	D	D	Z D X F S	Z D X F S	Z D X F S													
Initial conception				Z D X F	Z D X F	Z D X F												
Recurring tasks:																		
- Design review														Z D X F				
- Code review														Z D X F S				
Sprint 1:																		
- Implementation Model Checking 1						Z F	Z F	Z F	Z D F	Z D F	Z D F	Z D	Z D	Z D	Z D	Z F		
- Implementation Parsing AST 1						D X	D X	D X	X	X	X	X	X	X	X	X		
- Implementation conversion AST to GFC 1									D S	D S	D S	D S	D S	D S	D S	D S		
- Validation:																		
- testing MC 1																	Z F	Z F
- testing AST/GFC 1																	D X S	D X S

	16-févr	17-févr	18-févr	19-févr	20-févr	23-févr	24-févr	25-févr	26-févr	27-févr
Recurring tasks :										
- Design review	Z D X F S	Z D X F S								
- Code review						Z D X F S				
- Report					Z D X F S					Z D X F S
Sprint 2:										
- Implementation Model Checking 2										
- Implementation Parsing AST 2 (VF)										
- Implementation conversion AST to GFC 2										
- Validation:										
- testing MC 2										
- testing AST/GFC 2										
- Validation of product										

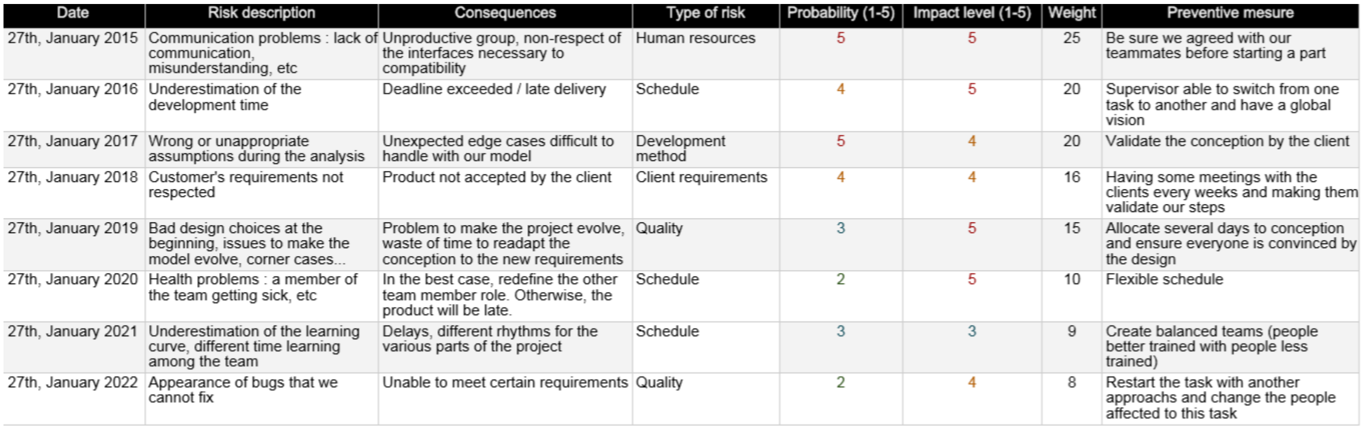
	02-mars	03-mars	04-mars	05-mars	06-mars	09-mars	10-mars	11-mars	12-mars	13-mars
Recurring tasks :										
- Design review	Z D X F S	Z D X F S								
- Code review						Z D X F S				
- Report				Z D X F S		Z D X F S				
										O
Sprint 3:										R
- Implementation Model Checking 3 (VF)										A
- Implementation conversion AST to GFC 3 (VF)										L
- Validation:										S
- testing MC 3										
- testing AST/GFC 3										
- Validation of product										
- Orals preparation : slides, content…					Z D X F S	Z D X F S	Z D X F S			

Legend:

Member	Responsibility
Zohour Abouakil	Project management
David Courtinot	Quality management
Xiaowen Ji	Configuration management
Fabien Sauce	Documentation
Sofia Boutahar	Testing

Part 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	Schedule	4	5	20	Supervisor able to switch from one task to another and have a global vision
27th, January 2017	Wrong or unappropriate 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.	Schedule	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	Schedule	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 approachs and change the people affected to this task

Figure III.1 - Analyse des risques

Part IV

Code and documentation 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 are the following :

Rule	**Description**	**Value**
FileLengthChecker	Check the number of lines in a file	1500
FileLineLengthChecker	Check the number of characters in a line	140
FileTabChecker	Check that there are no tabs in a file	enabled
ClassNamesChecker	Check that class names match a regular expression	[̂A-Z][A-a-z]*$
ClassTypeParameterChecker	Checks that type parameter to a class matches a regular expression	[̂A-Z_]$
FileTabChecker	Check that there are no tabs in a file	enabled
CyclomaticComplexityChecker	Checks that the cyclomatic complexity of a method does exceed a value	12
EmptyClassChecker	If a class/trait has no members, the braces are unnecessary	enabled
EqualsHashCodeChecker	Check that if a class implements either equals or hashCode, it should implement the other	enabled
MagicNumberChecker	Checks for use of magic numbers instead of constants (safer)	ignore = -1, 0, 1

MethodLengthChecker	Checks that methods do not exceed a maximum length	50
MethodNamesChecker .	Check that method names match a regular expression	[̂a-z][A-Za-z0-9]*(_=) ?$
MultipleStringLiteralsChecker	Checks that a string literal does not appear multiple times	allowed = 2
NotImplementedErrorUsage	Checks that the code does not have ? ? ? operators	enabled
NullChecker	Check that null is not used	enabled
NumberOfMethodsInTypeChecker	Check that a class/trait/object does not have too many methods	maxMethods = 30
NumberOfTypesChecker	Checks that there are not too many types declared in a file	maxTypes = 20
ObjectNamesChecker	Check that object names match a regular expression	[̂A-Z][A-Za-z]*$
ParameterNumberChecker	Maximum number of parameters for a method	maxParameters = 5
RedundantIfChecker	Checks that if expressions are not redundant, ie easily replaced by a variant of the condition	enabled
ScalaDocChecker	Checks that the ScalaDoc on documentable members is well-formed	enabled

IV.1.2 Verification by pair

As we have opted for an XP model for the programming aspect of the project, we consider that a code has passed the quality test if at least the two members of a pair have checked it. This is up to the quality manager to ensure this has been done, otherwise he should do it himself.

This is specific to the code quality checks and does not apply to the rest of the delivrable documents.

IV.2 Test strategy

IV.3 Configuration management

All the delivrable documents are managed on a git repository, including documentation and reports. Anyone is allowed to commit at anytime, however any push must have been authorized by the quality responsible after the code has been thoroughly tested against a set of tests by the test responsible.

Part V

Appendices