Development plan - Version 2

Zohour Abouakil
Sofia Boutahar
David Courtinot
Xiaowen Ji
Fabien Sauce

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

Signatures

Quality responsible :
Clients :

Sommaire

Part I

Project description and objectives

I.1 Surroundings of the project

Le projet long à l'ENSEEIHT 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

Part II

Project organization

II.1 Role definition

Project manager

Quality manager

Test manager

Test manager

Configuration manager

Documentation manager

Chain development

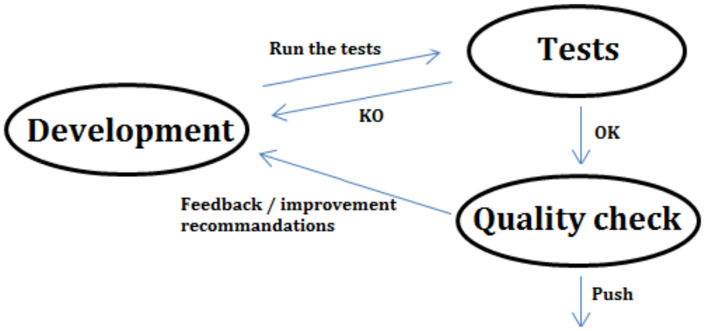


Figure II.1 - Schéma descriptif de la chaîne de développement

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

Sprint 3 backlog :

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

II.3.2 Planning

Task list	21-Jan	22-Jan	23-Jan	26-Jan	27-Jan	28-Jan	29-Jan	30-Jan	2-Feb	3-Feb	4-Feb	5-Feb	6-Feb	9-Feb	10-Feb	11-Feb	12-Feb	13-Feb
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													
Recurring tasks:																		
- Design review				Z D X F	Z D X F	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:																		
- jeu de tests MC 1																	Z F	Z F
- jeu de tests AST/GFC 1																	D X S	D X S

	16-Feb	17-Feb	18-Feb	19-Feb	20-Feb	23-Feb	24-Feb	25-Feb	26-Feb	27-Feb
Recurring tasks:										
- Design review	Z D X F S	Z D X F S								
- Code review						Z D X F S				
Sprint 2:	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
- Implementation Model Checking 2										
- Implementation Parsing AST 2 (VF)										
- Implementation conversion AST to GFC 2										
- Validation:										
- jeu de tests MC 2										
- jeu de tests AST/GFC 2										
- Validation du produit mergé										

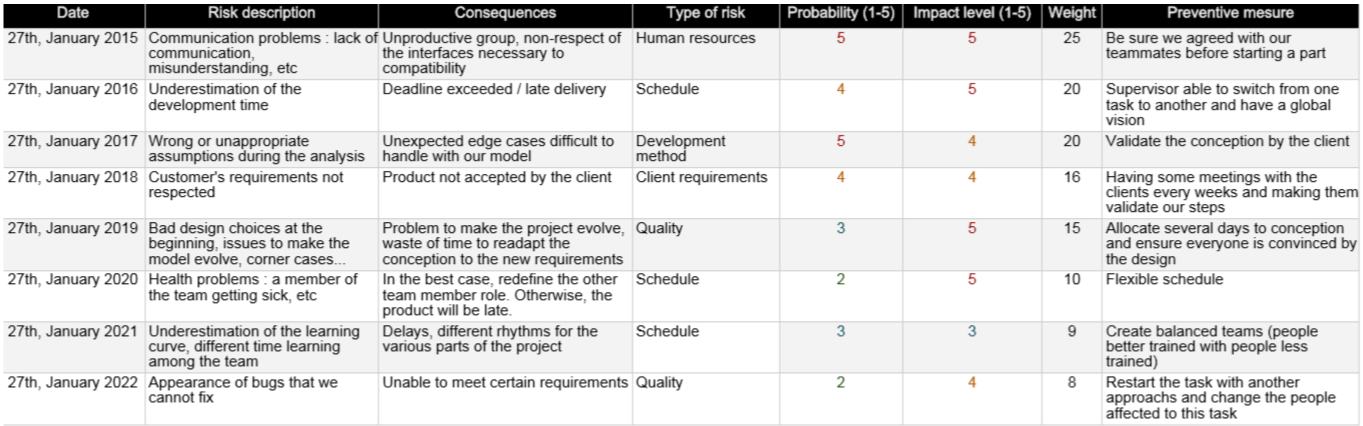
Legend:

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

	2-Mar	3-Mar	4-Mar	5-Mar	6-Mar	9-Mar	10-Mar	11-Mar	12-Mar	13-Mar
Recurring tasks:										🟥
- Design review	Z D X F S	Z D X F S								🟥
- Code review						Z D X F S				🟥
										🟥
Sprint 3:	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥
- Implementation Model Checking 3 (VF)										🟥
- Implementation conversion AST to GFC 3 (VF)										🟥
- Validation:										🟥
- Test MC 3										🟥
- Test AST/GFC 3										🟥
- Validation of product										🟥

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