Progress report 2 for 2017-Team-Up-7 INTiMALS

Company Name: RainCode Labs
Project reference: 2017-TEAM-UP-7
"INTiMALS: Intelligent Modernisation Assistance for Legacy Software"
Reporting Period: Report No. 2 for the period 01/10/2018 to 01/07/2019

General status of the project at M18

The following table summarizes the project's overall progress with respect to the entire 3-year work plan and shows that we are again on schedule regarding the different work packages

WP Name	**Expected Status**	**Expected %**	**Actual Status**	**Actual %**	**Comments** (changes since last report)
WP1 A shared language-parametric meta-model for software repositories and software patterns	4 deliverables complete out of 6 in total	4/6 = 66%	4 out of 6 deliverables complete	66%	Deliverables **P1.3, P1.4** complete.
WP2 Frequent subgraph mining algorithms for uncovering software patterns	3 deliverables complete out of 6 in total	3/6 = 50%	3 out of 6 deliverables complete	50%	Final version **R2.2** delivered. (was delayed in previous report) Deliverable **R2.3** complete.
WP3 Towards a pro-active modernisation assistant through matching algorithms for mined patterns	1 deliverable complete out of 3 in total	1/3 = 33%	1 deliverable complete	33%	No changes expected since last report.
WP0 Evaluation of developed technology on legacy systems at Raincode Labs	3 deliverables complete out of 6 in total	3/6 = 50%	1 deliverables complete	50%	Final version **R0.2** delivered. (was delayed in previous report) Deliverable **P0.3** complete.

Main achievements of the period

Before highlighting the achievements of the current period, we repeat the project's overall objective:

The project's overall objective is to develop an *intelligent modernisation assistant* for legacy software systems. The assistant pro-actively recommends engineers source code *modernisation actions* by comparing their ongoing development efforts with insights gained by treating software systems as data, in particular their source code and development history. The assistant will draw its intelligence from continuously mining for previously-unknown patterns both in the current state of the system's source code and structure (e.g., so-called programming idioms, coding conventions, library usage protocols) and in past and ongoing changes made to this source code (e.g., so-called systematic edits, repetitive changes, ...). The modernisation recommendations made by the assistant will appear increasingly informed as it refines or uncovers more previously-unknown patterns in the source code and version repositories it mines. The success of the modernisation assistant —its apparent learning ability— hinges on the quality of the pattern mining algorithms it incorporates.

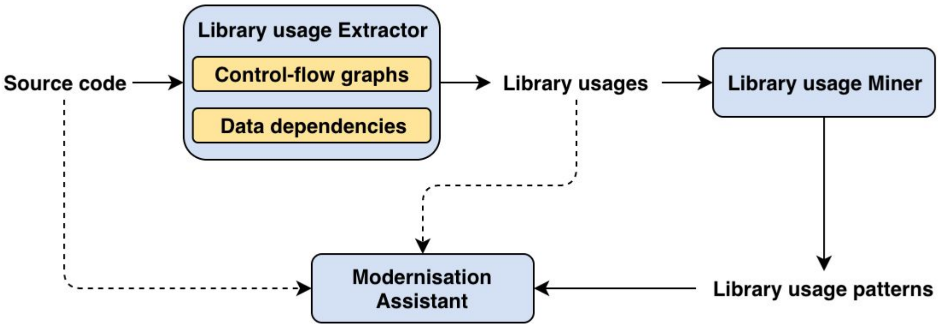
The activities towards realizing this intelligent modernisation assistant have progressed according to plan.

Whereas the previous report was concerned with the assistant's functionality to draw information from syntactic patterns (Y1), we will now focus on the progress we have made regarding usage patterns, based on semantic information (Y2). Before doing so, however, we first report on two remaining tasks regarding syntactic pattern mining that were delayed in Y1, but which are currently completed (cf. deliverables R0.2 and R2.2):

We first evaluated our INTiMALS' FREQTALS syntactic pattern mining algorithm on a more contemporary language like Java. We conducted a case study on a medium-sized Java software system where we analysed the quality of the patterns found. The results show (i) a significant reduction of the execution time and the number of discovered patterns with respect to the original FREQT frequent tree mining algorithm on which FREQTALS is based, (ii) that many of the discovered patterns highlight relevant code regularities, and (iii) that some of the patterns found are significantly larger than expected, which is a good thing to make them more interpretable by developers.

We then ran the FREQTALS algorithm on a legacy system in COBOL. This raised several technical issues that needed to be addressed through changes to the FREQTALS mining algorithm. Core issues encountered while running FREQTALS on legacy systems are its scalability in time and memory needed to mine the patterns, and in the number of results returned by the miner. Mining time was addressed by employing the parallel nature of the problem and optimising the implementation. Scalability in results was addressed by a combination of additional pattern constraints, maximising the found patterns and clustering them based on their matches. In the end, we managed to apply the algorithm successfully to a sample Cobol repository. In the second half of Y2 we plan to carry out more experiments on both Cobol and Java (and perhaps a third language) to validate the quality of the created mining algorithm and toolchain and to improve its performance further

In parallel, we also started to make progress regarding the mining of usage patterns, based on semantic information. We will now briefly describe the part of the INTiMALS assistant that is concerned with library usage, in its current state at M18.



Similar to the pipeline to obtain syntactic pattern information in Y1, there is another pipeline that focuses on finding library usage patterns. This pipeline is depicted in the Figure above, and consists of the following three components:

1. **Library usage extractor.** This first component extracts library usage instances from source code. Such an extractor is implemented for each programming language that the modernisation assistant accommodates. Each extractor will however produce its output according to our language-agnostic metamodel, which now includes support for representing library usage patterns.
   A library usage itself contains all references to a library of interest within a section of code (e.g. within a method), and how these references relate to each other in terms of control/data flow. In

particular, these references are either calls to the library, or definitions/uses of variables with a library type. Each library usage is represented as a graph structure that incorporates both the control and data-flow information of references to the library of interest.

2. **Library usage miner.** After producing all the graphs that represent library usages, the mining component then looks for patterns among these graphs, to get a better view of which functionality and types of a library are commonly used together. In essence, the library usage miner will look for the most commonly occurring subgraphs within all of these library usages. These subgraphs represent the library usage patterns.

3. **Modernisation assistant.** Once all library usage patterns have been mined, they can be explored in the modernisation assistant application. Aside from browsing the patterns themselves, we retain traceability links back to the original source code, such that all corresponding library usages matching with a pattern can be shown in context, in the source code. Aside from helping to understand how libraries are most commonly used in projects, the assistant can also make use of partial matching to look for anomalies, i.e. those library usages that only partially correspond to a pattern, which can indicate a bug.

Similar to our pipeline for syntactic patterns, this pipeline also is designed to be programming language-agnostic. To support an additional language, a new library usage extractor should be implemented, which exports to the INTiMALS metamodel for library usage. While such an extractor requires a fair amount of implementation effort, the core difficulties are to extract control flow and data flow information. Luckily, a compiler/interpreter of a programming language will face the very same problems, and might already expose an API to easily retrieve such information for a given program, which can greatly reduce the effort to provide a library usage extractor. The main inherent challenge that should be considered is to provide a clear separation between what is considered application code and library code, which is less trivial for legacy languages that may have an explicit notion of libraries.

We now report on the current state of the implementation of our framework, some preliminary results, as well as some of the challenges we have faced:

1. **Library usage extractor.** We currently have a fully functional extractor for library usage in Java code, using the Soot source code analysis framework to obtain control- and data-flow information. A slice is computed on the inferred control flow graphs to retain only those parts that are relevant to our library of interest. Data definition/usage information is then inserted into this sliced graph to obtain our representation of library usage, which is then exported to the INTiMALS language-agnostic metamodel XML format. With regards to the Cobol extractor, we can already make use of the compiler's functionality to produce "cross-references" for a given Cobol program, representing the required control and data-flow information for library usage. We are currently working on a tool that composes this cross-reference information into library usage graphs, conforming with our metamodel.

2. **Library usage miner.** A language-agnostic tool has been developed for mining library usages in the INTiMALS metamodel XML format. It reuses large portions of an existing, scalable library usage mining tool for Java applications: frequent itemset mining is first used to group roughly similar library usages into clusters. For each of these clusters, a SAT-based encoding is used to quickly check for subgraph isomorphisms between pairs of library usages, which forms the core of looking for commonly occurring subgraphs within library usages. These subgraphs ultimately are what we consider library usage patterns. While functional, the tool has currently only been tested on small

projects, and has not been fully evaluated yet.

3. **Modernisation assistant.** Library usages have not been fully integrated yet with the modernisation assistant, which will be our next step to properly evaluate our library usage pipeline against a number of use cases in both Java and Cobol code. Initial steps have also been made towards looking for partial matches of syntactical patterns and usage patterns, which is required to look for anomalies of patterns in the modernisation assistant. For syntactical patterns, our pattern matching algorithm has been extended to allow nodes to be skipped during the matching process. If only a small number of nodes are skipped for a given AST, it can be considered to be an anomaly. Similarly, for library usage patterns, we can measure the graph edit distance between a library usage and a pattern, to determine whether a given library usage should be considered an anomaly.

The following table provides a detailed overview of the status of all corresponding prototypes (deliverables starting with P) and reports (deliverables starting with R) scheduled for the entire duration of the project. Deliverables in grey were already completed since the previous progress report; those in green have been completed in the current progress report; deliverables in white are due in the future.

ID	Description	Due	Status
P1.1	Prototype implementation of the meta-model and an accompanying importer for syntactic information	**M03**	complete
P1.2	Prototype implementation of its representation for coding conventions & idioms	**M03**	complete
P1.3	Extension of meta-model and importer with support for semantic information	**M15**	complete
P1.4	Prototype implementation of its representation for usage patterns	**M15**	complete
P1.5	Extension of meta-model and importer with support for change information	M27	
P1.6	Prototype implementation of its representation for edit patterns	M27	
P2.1	Frequent subgraph mining system for discovering patterns in attributed graphs	**M06**	complete
R2.2	Report on its application to mining for coding conventions & idioms in syntactic information	**M06** **M12**	complete
P2.3	Adaptation of the frequent subgraph mining algorithm to support usage patterns	**M18**	complete
R2.4	Report on its application to mining for usage patterns in semantic information	M24	
P2.5	Adaptation of the frequent subgraph mining algorithm to support edit patterns	M30	
R2.6	Report on its application to mining for edit patterns in change information	M36	
P3.1	Prototype offering assistance for program comprehension in the form of a browser for mined patterns	**M09**	complete
P3.2	Prototype offering assistance for anomaly detection in the form of a browser for partial matches of mined patterns	M21	
P3.3	Prototype offering assistance for program modernisation in the form of edit recommendations that complete a partial match for a mined pattern	M33	
P0.1	Prototype of legacy software importer (syntactic information) for the INTiMALS meta-model	**M04**	complete
R0.2	Report on the evaluation of INTiMALS' support for mining coding conventions & idioms in legacy systems	**M07-M12**	complete
P0.3	Extension (semantic information) of the legacy software importer for the INTiMALS meta-model	**M16**	complete
R0.4	Report on the evaluation of INTiMALS' support for mining usage patterns	M24	
P0.5	Extension (version information) of the legacy software importer for the INTiMALS meta-model	M28	
R0.6	Report on the evaluation of INTiMALS' support for edit patterns in UC3	M36	

<u>Problems encountered :</u>

Deliverables delayed

The project has progressed according to plan. The deliverables that were delayed in Y1, **R0.2** and **R2.2**, have now been completed.

Anticipated changes

Aside from the project’s current progress, we would like to use this section to discuss anticipated changes for the remainder of the project.

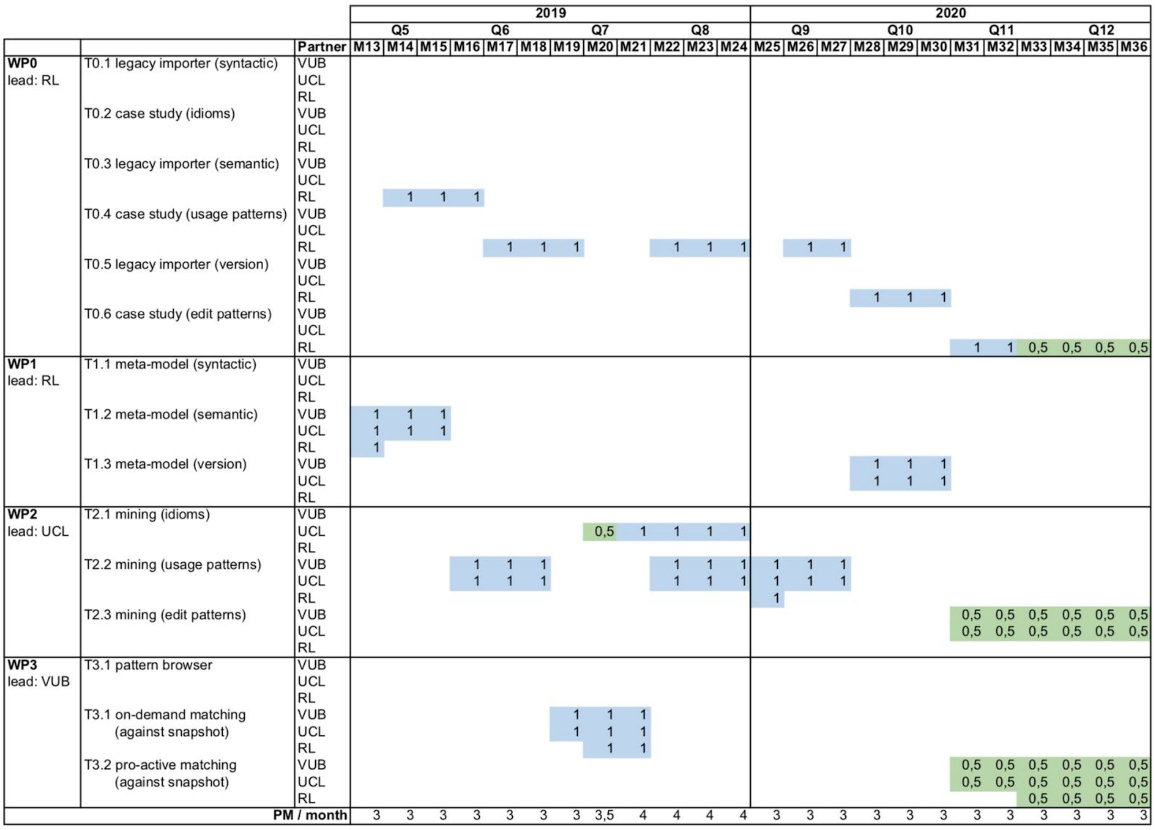
Changes in Y2: at this stage in the INTiMALS project we have developed a syntactic pattern miner that can produce detailed results for both Java and Cobol. However, evaluating the quality of the produced results still takes a lot of human resources, more than what we could invest with the current staff. Indeed, in parallel we needed to advance the project to explore mining of usage patterns, based on semantic information, and yet other techniques in Y3. In the meantime it would be a lost opportunity not to profit from the already developed syntactic pattern miner to conduct a deeper evaluation and analysis of the quality, quantity and relevance of the patterns found with that miner. Therefore, with the agreement obtained from Innoviris, in the second half of Y2 the UCL partner will hire a Phd student until the end of the year to perform part of this evaluation. This is shown as an extension of T2.1 in the Gantt chart below. (1-person months are shown in blue; 0,5-person months are shown in green) We will report on this evaluation in an additional deliverable **R2.2b**.

Changes in Y3: whereas the project’s first and second year respectively focus on mining for syntactic and usage patterns in legacy source code, Y3 focuses on edit/change patterns. To mine such patterns, multiple versions of a legacy project are required as input. We anticipate such legacy version information will be very scarce at best, and completely unavailable at worst. (This was also foreseen as a risk of T0.5 in the project proposal.) Consequently, we have planned the following changes to Y3, also shown in the Gantt chart below.

The focus on edit patterns will be reduced, and instead of looking for such patterns in *legacy* source code, RainCode also has an interest in finding edit patterns in the tools they develop, which are written in contemporary languages. For instance, complete git versioning information is available for the Raincode TIALAA compiler (pure C# + two parser generators + XML binding framework, all in a separate repository suitable for analysis) and for the C# runtime library used in other Raincode compilers (COBOL, PL/I, HLASM). This change of focus affects our planning as follows:

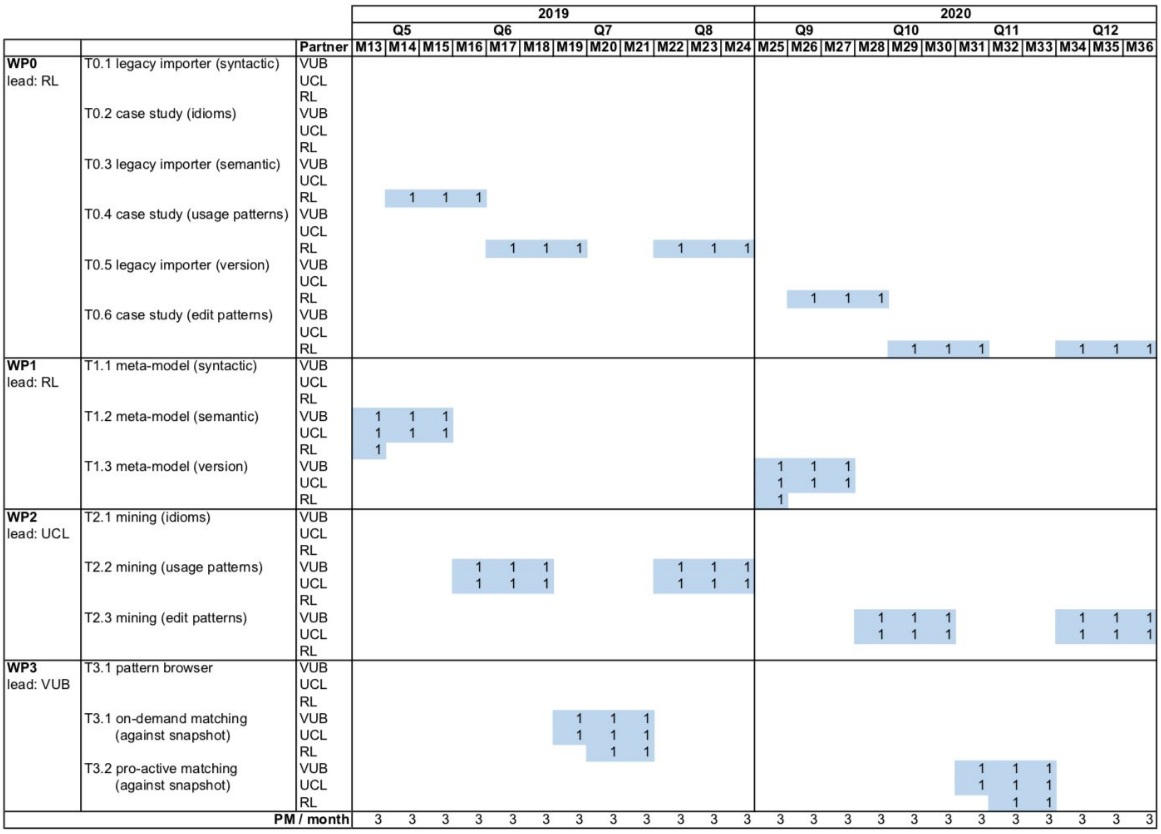
- T0.5 is now used to develop an importer for C# version information, instead of an importer for legacy code version information.
- Fewer person-months are assigned to T1.3, as VUB already has prior experience developing a metamodel for version information in contemporary languages.
- The tasks dedicated to mining and evaluating edit patterns (T0.6, T2.3) were previously split up into two phases of 3 months. To reduce our focus on edit patterns, we decided to merge these two phases into one 6-month phase, with roughly half the amount of person months assigned to it. This reduction enables us to develop and evaluate the final version of the modernisation assistant (T3.2) in parallel with T0.6 and T2.3.

Updated Gantt chart of Y2-Y3



			2019												2020											
			Q5			Q6			Q7			Q8			Q9			Q10			Q11			Q12		
		Partner	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
WP0 lead: RL	T0.1 legacy importer (syntactic)	VUB																								
		UCL																								
		RL																								
	T0.2 case study (idioms)	VUB																								
		UCL																								
		RL																								
	T0.3 legacy importer (semantic)	VUB																								
		UCL																								
		RL		1	1	1																				
	T0.4 case study (usage patterns)	VUB																								
		UCL																								
		RL					1	1	1			1	1	1		1	1									
	T0.5 legacy importer (version)	VUB																								
		UCL																								
		RL																1	1	1						
	T0.6 case study (edit patterns)	VUB																								
		UCL																								
		RL																			1	1	0,5	0,5	0,5	0,5
WP1 lead: RL	T1.1 meta-model (syntactic)	VUB																								
		UCL																								
		RL																								
	T1.2 meta-model (semantic)	VUB	1	1	1																					
		UCL	1	1	1																					
		RL	1																							
	T1.3 meta-model (version)	VUB																1	1	1						
		UCL																1	1	1						
		RL																								
WP2 lead: UCL	T2.1 mining (idioms)	VUB																								
		UCL								0,5	1	1	1	1												
		RL																								
	T2.2 mining (usage patterns)	VUB				1	1	1				1	1	1	1	1	1									
		UCL				1	1	1				1	1	1	1	1	1									
		RL													1											
	T2.3 mining (edit patterns)	VUB																			0,5	0,5	0,5	0,5	0,5	0,5
		UCL																			0,5	0,5	0,5	0,5	0,5	0,5
		RL																								
WP3 lead: VUB	T3.1 pattern browser	VUB																								
		UCL																								
		RL																								
	T3.1 on-demand matching	VUB							1	1	1															
	(against snapshot)	UCL							1	1	1															
		RL								1	1															
	T3.2 pro-active matching	VUB																			0,5	0,5	0,5	0,5	0,5	0,5
	(against snapshot)	UCL																			0,5	0,5	0,5	0,5	0,5	0,5
		RL																					0,5	0,5	0,5	0,5
		PM / month	3	3	3	3	3	3	3	3,5	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3

For comparison, the original Gantt chart of Y2-Y3

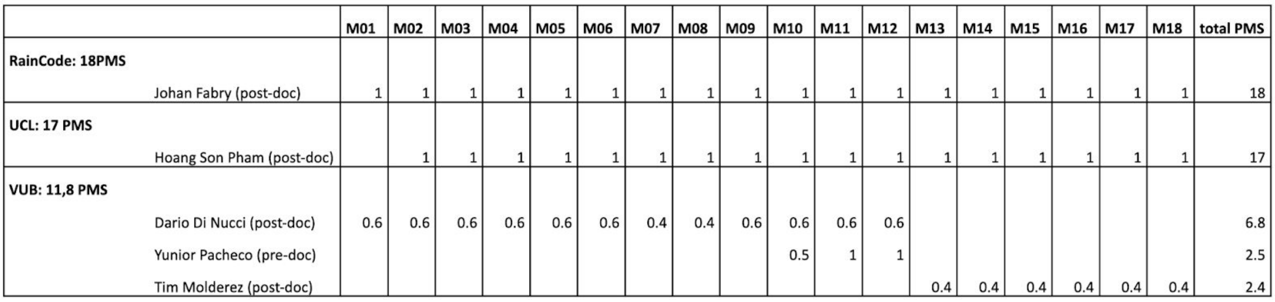


			2019												2020											
			Q5			Q6			Q7			Q8			Q9			Q10			Q11			Q12		
		Partner	M13	M14	M15	M16	M17	M18	M19	M20	M21	M22	M23	M24	M25	M26	M27	M28	M29	M30	M31	M32	M33	M34	M35	M36
WP0 lead: RL	T0.1 legacy importer (syntactic)	VUB																								
		UCL																								
		RL																								
	T0.2 case study (idioms)	VUB																								
		UCL																								
		RL																								
	T0.3 legacy importer (semantic)	VUB																								
		UCL																								
		RL		1	1	1																				
	T0.4 case study (usage patterns)	VUB																								
		UCL																								
		RL					1	1	1			1	1	1												
	T0.5 legacy importer (version)	VUB																								
		UCL																								
		RL														1	1	1								
	T0.6 case study (edit patterns)	VUB																								
		UCL																								
		RL																	1	1	1			1	1	1
WP1 lead: RL	T1.1 meta-model (syntactic)	VUB																								
		UCL																								
		RL																								
	T1.2 meta-model (semantic)	VUB	1	1	1																					
		UCL	1	1	1																					
		RL	1																							
	T1.3 meta-model (version)	VUB													1	1	1									
		UCL													1	1	1									
		RL													1											
WP2 lead: UCL	T2.1 mining (idioms)	VUB																								
		UCL																								
		RL																								
	T2.2 mining (usage patterns)	VUB				1	1	1				1	1	1												
		UCL				1	1	1				1	1	1												
		RL																								
	T2.3 mining (edit patterns)	VUB																1	1	1				1	1	1
		UCL																1	1	1				1	1	1
		RL																								
WP3 lead: VUB	T3.1 pattern browser	VUB																								
		UCL																								
		RL																								
	T3.1 on-demand matching	VUB							1	1	1															
	(against snapshot)	UCL							1	1	1															
		RL								1	1															
	T3.2 pro-active matching	VUB																			1	1	1			
	(against snapshot)	UCL																			1	1	1			
		RL																				1	1			
		PM / month	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Due to our reduced focus on edit patterns, additional time is freed up that will be dedicated to study some of the tasks of Y2 in additional depth. In particular, we have currently mined for usage patterns based on the existing BigGroum tool. We intend to perform a more extensive evaluation of this approach, and compare it with graph mining algorithms from the data mining literature, starting with the SubDue algorithm[1]. To develop a miner using these algorithms, T2.2 has been extended at the beginning of Y3. This work will result in an additional deliverable **P2.3b**. Finally, the comparison itself is done as an extension of T0.4, resulting in deliverable **R2.4b**.

Staffing

There continue to be some minor delays in staffing, mainly at the VUB partner, as illustrated in the following table:



	M01	M02	M03	M04	M05	M06	M07	M08	M09	M10	M11	M12	M13	M14	M15	M16	M17	M18	total PMS
RainCode: 18PMS																			
Johan Fabry (post-doc)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	18
UCL: 17 PMS																			
Hoang Son Pham (post-doc)		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	17
VUB: 11,8 PMS																			
Dario Di Nucci (post-doc)	0.6	0.6	0.6	0.6	0.6	0.6	0.4	0.4	0.6	0.6	0.6	0.6							6.8
Yunior Pacheco (pre-doc)										0.5	1	1							2.5
Tim Molderez (post-doc)													0.4	0.4	0.4	0.4	0.4	0.4	2.4

Despite these delays, the overall progress of the work plan has not been affected:

- As mentioned in the anticipated changes, a more in-depth evaluation of our syntactic pattern miner will be performed in the second half of Y2. The UCL partner will hire a recently graduated student, Céline Deknop, until the end of the year to perform part of this evaluation.
- VUB has now staffed an additional 0.4 FTE part-time researcher, a PhD graduate with expertise in mining source code patterns and edit patterns. Dario Di Nucci has continued to assist in VUB's tasks after M12, but is now working pro-bono on the project. Likewise, the PI also ensures the timely delivery of all due prototypes pro-bono. Yunior Pachecco will rejoin the project at the end of Y2, for a 3-month period. At the start of Y3, a senior PhD student will also join the project full-time.

Overall vision of results and prospects for the future (To be completed only at the end of the project)

Implementation of the business plan and valorisation

Attending different conferences has allowed the divulgation of Raincode's participation in the project to the research community. Amongst others, Johan Fabry, the Raincode Engineer, presented an informal demonstration of our toolchain and the current version of the pattern browser (deliverable P3.1) during SCAM 2018, the 18th IEEE International Working Conference on Source Code Analysis and Manipulation, held end of September 2018. At the renowned <Programming> 2019 conference he presented a demo entitled "A Language-Parametric Toolchain for Mining Idiomatic Code Patterns", on April 3, 2019. He also presented a paper entitled "A Language-Parametric Modular Framework for Mining Idiomatic Code Patterns" and gave some examples of patterns mined from a Java case at the annual Seminar on Advanced Tools and Techniques for Software Evolution (SATToSE 2019), held in the first half of July 2019. At the same

[1] http://ailab.wsu.edu/subdue/

seminar Johan gave a live demo of the syntactic pattern mining applied to legacy Cobol code. Earlier in July he gave a similar presentation and demo at a weekly research seminar organised by the UCL partner. In the context of library usages, Yunior Pacheco has presented the paper "Mining Scala Framework Extensions for Recommendation Patterns" at the International Conference on Software Analysis, Evolution and Reengineering (SANER) in February 2019. Finally, a paper jointly written by the project partners and entitled "Mining Patterns in Source Code using Tree Mining Algorithms" has very recently been accepted, and will be presented at the International Conference on Discovery Science (DS2019) in October 2019.

<u>Evolution of the company</u>

Overall the company is having a normal positive evolution.