Measuring the progress of an Industrial Reverse Engineering Process

Brice Govin* †, Nicolas Anquetil †, Anne Etien †, Arnaud Monegier *

* THALES AIR SYSTEMS, Parc tertiaire SILIC, 3 Avenue Charles Lindberg 94628 Rungis Cedex, France
{brice.govin, arnaud.monegier}@thalesgroup.com
† RMod team, Inria Lille Nord Europe, University Lille 1, CRIStAL, UMR 9189, 59650 Villeneuve d'Ascq, France
{brice.govin, nicolas.anquetil, anne.etien}@inria.fr

***Abstract*—In an industrial process of production, project managers look for ways to visualise the progress of their project. Reverse engineering projects do not escape this requirement. These projets have the particularity to provide information, more or less accurate, about the whole project, from documentation to source code. Comparison between the number of treated entities (lines of code or entities) and the global corresponding number should ease the task of computing project progress. However, in large projects, studying all entities can be an endless task. Engineers need ways to help them in such a task. This paper sets up the general issue of how to follow progress in a reverse engineering process. We also propose an early solution to this issue and discuss about its improvements.**

I. Introduction

In the current industrial world, project managers face issues such as resources management or project progress. They want to see the remaining time and budget for their project. To address these issues, several tools and techniques have been designed (for resources management in [1] for example). Project progress often are addressed through dashboards that gather lots of metrics.

Computing progress of reverse engineering projects [2] seems to be eased by the availability of the whole source code of the project. Comparing the number of treated lines of code or the number of treated entities to the global corresponding number, is one possible way to compute project progress.

However, large projects raise a new, and more practical issue. Indeed, browsing and studying every piece of source code can be a long and fastidious task for developers.

This paper is build as follow. Section II details the issue of how to follow progress in a reverse engineering process. Section III exposes the early idea on which we have based our solution. Section IV makes a brief state of the art in displaying the progress of a RE process. Section V concludes and presents the work to be done for the solution.

II. Issue

Following progress in reverse engineering projects is eased by the source code. Since the whole source code of the project is available for a RE project, computing the amount of treated lines of code or entities over all the lines of code or entities of the project can be easily done.

In large projects, the fastidious task of treating every entities or lines of code can be managed thanks to a top-down approach. For instance, in an object-oriented written software, a top-down approach would be to first understand the purposes of packages and then to analyse relationships between them. The next step would be to study classes then methods and sometimes even to analyse statements. During this process, considering a package as treated raise the question of how to consider its entities. If its entities are also considered as treated, analysing them will not increase the progress metric because they already have been considered as treated. In the opposite, it may happen that the content of an entity is never studied for different reasons. This top down approach mainly used in reverse engineering process does not seem adapted to the progress metrics intuitively defined. Hence, the issue to be tackled here is to find a way to ease the progress metric computation in a RE project

We propose an early solution on this issue in the next section by distinguishing two kinds of possible marks on an entity to reverse engineer.

III. Solution

Progress over a RE project can be naively done by setting a *treated* characteristic on an entity. A characteristic is a binary attribute for an entity. Thus, an entity can be either *treated* or *not treated*, with a default characteristic value set as *not treated* since nothing has been done on the entity. We introduce a mark action to set *treated* and *not treated* characteristic.Indeed, most of the time, entities distinction over a binary characteristic is done through a marking system (*e.g.* highlighting in a text, open/close state in a bug tracker).

On one hand, since the T_n (*not treated*) characteristic is the default one for an entity, there is no need to define a specific action. This characteristic will neither be detailed nor used in this paper. T_e (*Treated*) characteristic has to be associated to a specific marking action. To tackle the scale issue, it should be possible to spread this action to the children of an entity. It corresponds basically to apply a T_e to an entity and all its children.

On the other hand, when analysing entities, developers often browse briefly other entities without really treating them. Such an event should be considered as well in the

progress computation. Then a *browsed* characteristic has also to be implemented. A marking action is associated to the B_e (*browsed*) characteristic. Since to analyse an entity, one has to browse it, it is not necessary to create an action for spreading B_e.

To complete the set of potential action, one should be able to spread both B_e and T_e from an entity to its children. Spreading marks is an action that counters the issue of analysing every pieces of code. When an engineer decides that he treated not only an entity but also its children, he can avoid the fastidious task of going through every children to mark them, thanks the spreading action .

Figure 1 shows the different marking actions. Graph a) displays a spread action of T_e mark. Graph b) displays an action of spreading simultaneously B_e and T_e. Graphs c) and d) are two examples of simply marking an entity with either B_e or T_e.

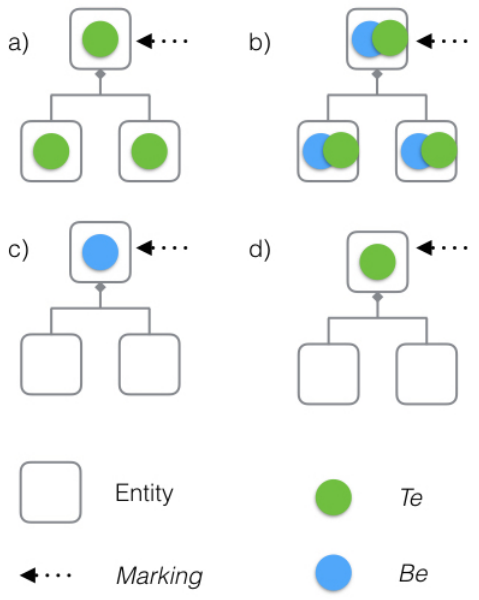

Fig. 1. Spreading of actions in entities containment hierarchy

In practice, every action that includes T_e (graphs a) b) and d) in figure1) has to be done manually by the developers each time they considered it. It does not seem acceptable to provide an automatic marking for this kind of characteristic since it is totally dependent on the developer. However, action for B_e can be applied automatically. Automatic application of B_e will depend on the kind of source code representation developers are manipulating. For example, if the source code is represented by a model, B_e can be automatically applied the first time an entity is analysed.

Progress computation can then be done using the three characteristics T_n, B_e and T_e. Since an entity can be marked as B_e or T_e or both, from our point of view, the only important characteristics are these two. Therefore, progress metric equation should only consider B_e and T_e as variables. The definition of a characteristic provides the following equation for a progress p of a project P:

$$p(P) = \frac{\sum_{i=1}^{n}(B_{ei} + T_{ei})}{2 * n} \quad (1)$$

where :

- n is the number of software entities contained in the system to reverse engineer.
- X_i represented the characteristic X of the entity i
- $\forall i \in [|1; n|]$, $B_{ei} \begin{cases} = & 1 \quad marked \\ = & 0 \quad otherwise \end{cases}$
- $\forall i \in [|1; n|]$, $T_{ei} \begin{cases} = & 1 \quad marked \\ = & 0 \quad otherwise \end{cases}$

Currently, such a formula has not yet been implemented. Nonetheless, we started the implementation of the marking system. We can easily get the amount of marked entities regarding the kind of marking (T_n, B_e, T_e).

IV. Related works

Researches in controlling software projects have been conducted since the very beginning of software production. Numerous papers [3], [4], [5] propose processes to ease the management of a software project. In [3], the author defines a process to manage software project. This process is currently wide spread and known as the V-Cycle. Most of the researches done about software project processes gave birth to processes that are widely used in industry nowadays (*e.g.*V-Cycle, spiral model).

Other works define and use a set of metrics to manage a software project [6], [7]. In such a case, metrics are often related to complexity, for example structural complexity [8]. Software element complexity appears to be a point to include in the computation of progress of a software reverse engineering process.

V. Further work

As further work, we planned to finish first the implementation of this progress metric. The next step will be to test this metric on a real life project. This project has to be reverse engineering of a large software system in order to validate the scalability of the marking process. We tried to take into account that marking can be a fastidious task and added a spreading action. However, we do not know if this action will really ease the process on a large software.

The equation 1 does not consider complexity of an entity. Such an information is also relevant when computing project progress. Treating a complex entity should be more valuable for the progress than treating a simple one. For example, an accessor method has not same complexity as a method that implement an algorithm then they should weight the same in the progress metric. Complexity can be a concrete complexity (*e.g.* cyclomatic) or a combination of two or more complexities. We tried to refine equation 1 to consider what we call *hierarchical complexity*, explained below the equation 2.

$$p(P) = \frac{\sum_{i=1}^{n}(B_{ei} + T_{ei}) * s_i}{2 * \sum_{i=1}^{n} s_i} \quad (2)$$

where $\forall i \in [|1; n|], s_i$ is the value of the entity i. [1]

Then the value of an entity derives from the amount of hierarchical level directly beneath it plus one as depicted in figure 2. Element A has two containments level beneath it so its value is 3. Element A1, A2.1 and A2.2 do not contain any element so their value is 1. Element A2 has only one level of containment thus it has a value of 2.

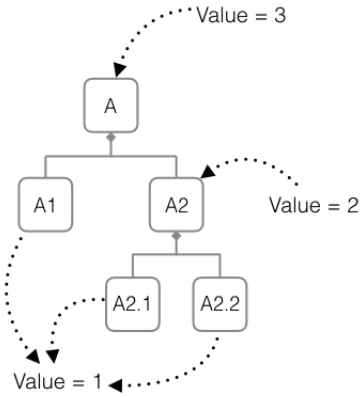

Fig. 2. Value of a software entity

This implementation defines s_i as a hierarchical complexity value and will weight the project progress metric. As stated previously, the complexity metric that defines s_i can be any kind of complexity metric, even a mix between several complexity metric. We did not define yet the granularity of our containment graph. This is a task to be done next, in order to define more clearly the granularity of the progress metric. Adding a weight related to the complexity of an entity is a way to have a more accurate progress metric.

To conclude, we have proposed an early idea of a solution for the issue of following a RE project progress. This solution considers two kinds of entity marking, B_e (*browse marking*) and T_e (*treat marking*), for computing the progress. An equation of a RE project progress has been written with the two kinds of entity marking as variables. Since we are currently acting on a RE project, we focused our solution on it. Nonetheless, we truly believe this solution can be used in any kind of hierarchical projects. Some further works have been planned to first test the marking process, then the project progress metric, and finally to improve the accuracy of the project metric.

REFERENCES

[1] J. M. Wilson, "Gantt charts: A centenary appreciation," *European Journal of Operational Research*, vol. 149, no. 2, pp. 430–437, 2003.

[2] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, Jul. 2009. [Online]. Available: http://rmod.lille.inria.fr/archives/papers/Duca09c-TSE-SOAArchitectureExtraction.pdf

[3] P. Rook, "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, p. 7, 1986.

[4] S. Mathur and S. Malik, "Advancements in the v-model," *International Journal of Computer Applications*, vol. 1, no. 12, 2010.

[5] J. Wateridge, "How can is/it projects be measured for success," *International journal of project management*, vol. 16, no. 1, pp. 59–63, 1998.

[6] R. B. Grady, "Successfully applying software metrics," *Computer*, vol. 27, no. 9, pp. 18–25, 1994.

[7] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[8] D. P. Darcy, C. F. Kemerer, S. Slaughter, J. E. Tomayko *et al.*, "The structural complexity of software an experimental test," *Software Engineering, IEEE Transactions on*, vol. 31, no. 11, pp. 982–995, 2005.

[1] Equation 1 is a specific case of equation 2 were $\forall i, s_i = 1$