Toward maintainance of a commencial software: Defects model

HOUEKPETODJI Mahugnon Honore*†, Nicolas Anquetil‡
*University of Lille, France
†Inria Lille - Nord Europe, France
homahugnon@gmail.com, nicolas.anquetil@inria.fr

***Abstract*—Legacy systems are old software that style does useful tasks. In industrial software companies, legacy systems are often crucial for the company business model and represent a longterm business investment. Legacy systems are known to be hard to maintain. This is the case in a french company whose main product is twenty years old software written in PowerBuilder (we call it SPB). Our longterm goal is to help reengineer. But how to validate our intervention? Using moving average, and regression, we evaluate the maintenance state of SPB and produce a dashboard to monitor our future actions. We validate our results with the developer team. We In this paper, we present a lightweight defects model to help planning commercial software maintenance**

***Index Terms*—Lagacy system, Defect model .**

I. Introduction

Software companies usually invest time and energy to improve the quality of the software they develop to respond to the rising market demands. Features are developed in a hurry sometimes. As results, they less allow resources for software refactoring to remove source code defects. As the software is out of control and it becomes hard to improve or maintain. Rewriting this software require time and a lot of resources. At this point, one of the solutions is to reverse-engineer them.

The main business product is SPB of our company. Our longterm goal is to reverse-engineer SPB. With over 3,000,000 lines of codes, continually updated for more than 20 years, SPB is not versioned and not unit tested. Bugs is registered in a database without correlation with which part of the system is responsible for the bug. Old versions of SPB are lost until 2012. The original developers of SPB are part of the developers' team. So the currents developers only know SPB partially. So we could not completely rely on developers view of the system. Besides, there is also a misunderstanding between developer team and the business team. As the business team doesn't know the state of SPB. In this condition, it is completely impossible to successfully reengineer SPB without 15 years of the useful data model and our analytics.

Our data model and analytics have to the main goal. It provides a report on the state of SPB to the entire company and shows the business team the need to reengineer SPB. Furthermore, it will help us to monitor our future task on SPB. In this paper, we will present the data model and the analytics we made.

This paper is structured as follow:

II. Related Work

III. Background

A. Presentation of SPB

SPB is build with PowerBuilder. PowerBuilder is an enterprise development tool that allows you to build many types of applications and components. A powerBuilder application components are grouped by libraries. A PowerBuilder library contains differents type of Objects: Datawindow, User object, Global function, etc. In SPB libraries can be grouped by business module . Figure 1 give an overview of SPB achitecture.

B. Ticket

A ticket is related do a task to do. This task can be fixing a bug, writing documentation, adding a new feature, etc. A ticket is open for a task . Once the task is done and validated, the ticket is close. A ticket has the following characteristics:

- the libraries it is related to
- the creation date
- the closing date
- time spent by a developer
  - time to analyze
  - time to implement solution
  - time to test
- the estimation of the time needed by the developer to work on the ticket

IV. Methodology

We collected data from the company Tickets database . Then we procced to data cleaning. The data analytics process is guided by the following research rule:

- Pose problem
- Make experimentations on data
- Validation of results

A. Data collection and data cleaning

The ticket database contains data from 1998 until 2019. This data includes tickets differents teams and softwares in the company. In the field related to the library a ticket is related to, the name of the library is not always well written.

The datas are consistent from 2004 to 2019 . We remove all tickets, non related to SPB and reformat durations per ticket. By inteviewing developers, we categorize tickets into to sets: tickets related to defects and tickets related two evolutions.

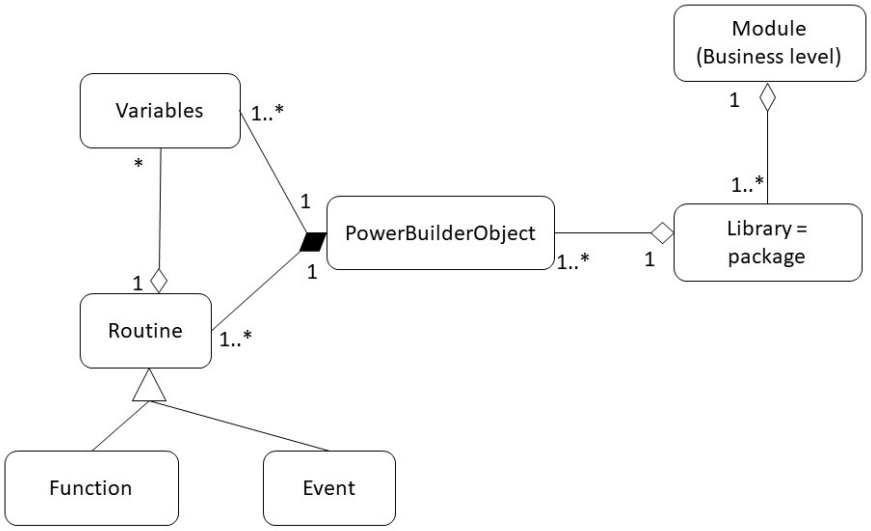

Fig. 1. Simplified SPB achitecture. Very simplified architecture of SPB

B. Problems

- Is there a correlation between defects and evolutions per libraries:
  - Does evolutions cause defect?
  - Does evolutions occur after defects occur?
- Is there a relationship between defects and libraries:
  - Does defects in a librarie cause defect in other libraries?
  - Does some libraries always have defects together?
- How long in average does it take to open and close a ticket?
- How long in average does a developer spend on a ticket ?
- How long in average does a developer spend to test a ticket after development ?
- Does the team manager estimate well the time needed by the developer?
- If the time estimated by the team manager is good, does the developer finish because of time constraint?

C. Experimentations

For each problem, we experiement on the ticket database.

1) Correlation between defects : We plot defects and evolution per library over time. In x axis we have years and in y axis we have the number of defects or evolutions. According to the visualizations, some libraries trends to as many defects as evolution over time. For some libraries, we observe that they do not have defects. Some others seem to have defects after evolution.

V. Results and Discussion

VI. Summary and Conclusions