UNIL | Université de Lausanne

Unicentre
CH-1015 Lausanne
http://serval.unil.ch

Year : 2019

THE ORIGINS OF SEVERE SOFTWARE DEFECTS ON EVOLVING INFORMATION SYSTEMS: A DOUBLE CASE STUDY

Hillah Nico

Hillah Nico, 2019, THE ORIGINS OF SEVERE SOFTWARE DEFECTS ON EVOLVING INFORMATION SYSTEMS: A DOUBLE CASE STUDY

Originally published at : Thesis, University of Lausanne

Posted at the University of Lausanne Open Archive http://serval.unil.ch
Document URN : urn:nbn:ch:serval-BIB_A02DBD5A439B3

Droits d'auteur
L'Université de Lausanne attire expressément l'attention des utilisateurs sur le fait que tous les documents publiés dans l'Archive SERVAL sont protégés par le droit d'auteur, conformément à la loi fédérale sur le droit d'auteur et les droits voisins (LDA). A ce titre, il est indispensable d'obtenir le consentement préalable de l'auteur et/ou de l'éditeur avant toute utilisation d'une oeuvre ou d'une partie d'une oeuvre ne relevant pas d'une utilisation à des fins personnelles au sens de la LDA (art. 19, al. 1 lettre a). A défaut, tout contrevenant s'expose aux sanctions prévues par cette loi. Nous déclinons toute responsabilité en la matière.

Copyright
The University of Lausanne expressly draws the attention of users to the fact that all documents published in the SERVAL Archive are protected by copyright in accordance with federal law on copyright and similar rights (LDA). Accordingly it is indispensable to obtain prior consent from the author and/or publisher before any use of a work or part of a work for purposes other than personal use within the meaning of LDA (art. 19, para. 1 letter a). Failure to do so will expose offenders to the sanctions laid down by this law. We accept no liability in this respect.

UNIL | Université de Lausanne

FACULTÉ DES HAUTES ÉTUDES COMMERCIALES

DÉPARTEMENT DES SYSTÈMES D'INFORMATION

THE ORIGINS OF SEVERE SOFTWARE DEFECTS ON EVOLVING INFORMATION SYSTEMS: A DOUBLE CASE STUDY

THÈSE DE DOCTORAT

présentée à la

Faculté des Hautes Études Commerciales
de l'Université de Lausanne

pour l'obtention du grade de
Docteur ès Sciences en systèmes d'information

par

Nico HILLAH

Directeur de thèse
Prof. Thibault Estier

Jury

Prof. Rafael Lalive, Président
Prof. Yves Pigneur, expert interne
Prof. Periklis Andritsos, expert externe
Prof. Jean-Henry Morin, expert externe

LAUSANNE
2019

UNIL | Université de Lausanne

FACULTÉ DES HAUTES ÉTUDES COMMERCIALES

DÉPARTEMENT DES SYSTÈMES D'INFORMATION

THE ORIGINS OF SEVERE SOFTWARE DEFECTS ON EVOLVING INFORMATION SYSTEMS: A DOUBLE CASE STUDY

THÈSE DE DOCTORAT

présentée à la

Faculté des Hautes Études Commerciales
de l'Université de Lausanne

pour l'obtention du grade de
Docteur ès Sciences en systèmes d'information

par

Nico HILLAH

Directeur de thèse
Prof. Thibault Estier

Jury

Prof. Rafael Lalive, Président
Prof. Yves Pigneur, expert interne
Prof. Periklis Andritsos, expert externe
Prof. Jean-Henry Morin, expert externe

LAUSANNE
2019

UNIL | Université de Lausanne
HEC Lausanne
Le Décanat
Bâtiment Internef
CH-1015 Lausanne

IMPRIMATUR

Sans se prononcer sur les opinions de l'auteur, la Faculté des Hautes Etudes Commerciales de l'Université de Lausanne autorise l'impression de la thèse de Monsieur Nico HILLAH, titulaire d'un bachelor en Technologies de l'information de Methodist University College Ghana, et d'un master en Systèmes d'information de l'Université de Neuchâtel, en vue de l'obtention du grade de docteur ès Sciences en systèmes d'information.

La thèse est intitulée :

THE ORIGINS OF SEVERE SOFTWARE DEFECTS ON EVOLVING INFORMATION SYSTEMS: A DOUBLE CASE STUDY

Lausanne, le 10 janvier 2019

Le doyen

[illegible]

Jean-Philippe Bonardi

Members of the thesis committee

Dr. Thibault Estier

Senior Lecturer and Researcher at the Faculty of Business and Economics (HEC) of the University of Lausanne

Thesis supervisor

Professor Rafael Lalive

Professor at the Faculty of Business and Economics (HEC) of the University of Lausanne

President of the Jury

Professor Yves Pigneur

Professor at the Faculty of Business and Economics (HEC) of the University of Lausanne

Internal member of the thesis committee

Professor Jean-Henry Morin

Professor at the Institute of Information Service Science of the University of Geneva

External member of the thesis committee

Professor Periklis Andritsos

Professor at the Faculty of Information of the University of Toronto

External member of the thesis committee

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Nico HILLAH

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my satisfaction.

Signature : Th. Estier Date : 7 janvier 2019

Prof. Thibault ESTIER
Thesis supervisor

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Nico HILLAH

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my satisfaction.

Signature : Yves Pigneur Date : 5 janvier 2019

Prof. Yves PIGNEUR
Internal member of the doctoral committee

University of Lausanne

Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Nico HILLAH

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my satisfaction.

Signature : [illegible] Date : January 6, 2019

Prof. Periklis ANDRITSOS

External member of the doctoral committee

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Nico HILLAH

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my satisfaction.

Signature : [illegible] Date : January 9, 2019

Prof. Jean-Henri MORIN
External member of the doctoral committee

Abstract

Version française au recto

Software problems do not only induce high financial loss, but also sometimes induce human loss. Those problems are due to the presence of software bugs, failures, errors, and defects in software systems. These software anomalies, and in particular the software defects, have a huge impact not only on business activities but also on the cost of developing and maintaining these software systems. In order to identify their sources, particularly the ones causing severe impacts on the systems' operations, we conducted two case studies. We analyzed software defects of two systems over a period of a year and a half. We classified these software defects, according to their trigger factors and according to their severity impact. Conducting these studies led us to propose "the origins of severe software defects method" order to identify trigger factors that cause severe software defects on a given evolving system. We also found that the group of technology trigger factors causes more severe defects than the other groups of trigger factors for this type of systems.

We divide this manuscript into two main parts. In the first part, we will present the synthesis of our four published research papers. In the second part, we will present these four published articles in full.

Résumé

English version at the front

Les pannes de logiciels n'entraînent pas uniquement d'immenses pertes financières, mais provoquent parfois aussi des pertes en termes de vies humaines. Ces pannes sont la plupart du temps provoquées par la présence de bugs, d'erreurs, de failles ou de défauts au sein de ces logiciels. A savoir que ces anomalies, en particulier les défauts, ont un impact considérable sur les activités économiques et sur le coût de développement et de maintien des systèmes de ces logiciels. Afin d'identifier les facteurs qui sont à la source des défauts les plus coûteux, nous avons étudié deux systèmes évolutifs. A travers plusieurs études, nous avons analysé les défauts de ces systèmes sur une période d'une année et demie. Ces études nous ont permis de classer les défauts sur la base de leurs facteurs déclencheurs d'une part, et sur la base du degré de sévérité d'autre part. Ceci nous a amené à proposer la méthode "the origins of severe software defects method" pour aider à l'identification des facteurs déclenchants les défauts coûteux d'un système évolutif. En plus de cette méthode, ces études nous ont permis d'identifier que les facteurs du type technologique, comparés aux autres types de facteurs, sont à l'origine de la majorité des défauts coûteux pour ce type de systèmes.

Ce manuscrit est divisé en deux parties. En première partie, nous allons présenter la synthèse de nos articles publiés et animés lors de diverses conférences scientifiques à travers le monde. Enfin, dans la seconde partie, nous mettrons à disposition du lecteur l'intégralité de ces quatre articles.

Acknowledgements

First of all, I would like to thank the almighty knowledge, wisdom, and mystery. I would also like to thank people who directly or indirectly helped me to conduct this research. More specifically, I would like to thank my professor Thibault E. for giving me the opportunity to conduct this research. I also want to say thank you to him for his time, advice and guidance during this research journey. I would also like to thank my contacts in the Swiss government body who provided me with the necessary data and information to conduct this study.

A special thanks to my jury members, namely Jean-Henry M., Yves P., Periklis A., Thibault E., and Rafael L. for their time and guidance.

I want to send my gratitude to all the past and present professors of the DESI department as well as other universities for the fruitful conversations I had with them, especially Solange G., Stéphanie M., Jacques D., Yves P., Christine L., Alessandro V., Benoît G., Marco T., and Mauro C. I will also thank my colleagues for their support and the great moments they accepted to share with me during all these years, especially Thomas, Gabriella, Bertil, Laurent, Ricardo, Martin, Kenny, Gael, Natacha, Gabor, Alexandre, Jean-Sebastien, Fabio, Hazbi, Dina, Dana, Bastien, Laura, Vaibhav, Manon, Clement and all the student assistants I collaborated with. I will thank also the past and present secretaries of the DESI department, as well as all the administrative staff of the HEC Lausanne faculty, for the time and advices. A special thanks to Sophie M. for her professional contribution to the success of this project.

I will also thank my family and friends for their support in special, Nana C. H., Angelis G., Dzifa A., Amavi H., Dzidonou A., Ama A., Ayele H., Joseph H., Kweku D., Adwoa A., Kafui D., George, Albert H., Kendra S., Danielle S., and Bernadette M.

Finally, a special thank you to all the people I met during this journey who directly or indirectly led me to successfully conduct this research. To all nations God bless you all.

List of Abbreviations

Abbreviation	Explanation
ACH	IS architecture
ASD	Analyzed Software Defect
B.IS	Business/IS alignment
CRs	Change Requests
E-type	Evolving type
ETL	Extract, Transform, and Load
EVOLIS	EVOLution of Information Systems
IEEE	Institute of Electrical and Electronics Engineers
IS	Information Systems
KPI	Key Performance Indicator
SD	Software Defect
SDLC	Software Development Life Cycle
SDM	SD Management system
SDs	Software Defects
TCH	Technology
UI	IS/user fit
W-	Weighted score

Table of Contents

List of Figures

List of Tables

<u>Part I: Synthesis of Published Papers</u>

1 Introduction

« Gnothi seauton »

Temple of Delphi (Pytho).

1.1 Motivation and Context

1.1.1 Motivation

In common language, it is usual to call software defects (SDs) bugs. As software systems are part of our daily life, so problems come with their usage. Nowadays, there is no single day in the world without a bug affecting our daily activities. Not only do they affect our daily operations, but they also cause a huge financial loss when they happen. This loss ranges from one to billions of dollars. Unfortunately, the consequence of a software defect (SD) may concern more than money, but also cause loss of human lives. All domains of business and non-business activities are concerned: from banking to hospital, from education to nuclear activities, from economy to transportation. Here are some famous financial and human life loss caused by a software bug in our history:

- IT companies
    - In 1999, half a million British citizens discover that their passports could not be issued on time due to bugs in a new system brought on by Siemens, and the incapacity of the staff using this system in their daily routine. In fact, Siemens did not provide a good training for system users; this prevented them to accurately use the system [1].
- Education domain
    - A group of hackers had stolen data from 77 million users from the Edmodo online educational platform in May 2017 [2]. The stolen data belonged to students using this platform, to their parents, and to their teachers.

- Transportation domain
    - A bug that affected the IT system of British Airways made the company cancel all flights between Heathrow airport and Gatwick airport in May 2017.
    - Ten years earlier, in 2007, at Los Angeles International Airport, US., more than 17,000 planes were grounded on the floor due to a bug which prevented the entire network of the United States Customs and Border protection to shut down [1]. According to them, a network card sending an incorrect message across the shared network caused this bug until this message hit the USCBP systems.
- Military domain
    - An attack killed 28 American soldiers in February 1991 in Saudi Arabia due to an existing bug found later in the anti-missile software system.
- Financial domain
    - On 1st August 2012, an American trading company named Knight Capital group lost 440 million dollars in only 30 minutes due to a bug in its trading system [3]. In fact, this bug was able to cause this high damage due to the configuration of the business processes.
- Cyber criminality and Health domain
    - The virus WannaCry indeed made the National Health System of England cry in May 2017, when a group of hackers used an unidentified SD in the Microsoft Windows operating system to launch a ransomware attack not only in England but also all over the world, causing cancellations of 19,000 health care appointments [4].
    - In September 2018, 50 million Facebook account logins were reset due to a data breach. *"The breach was caused by an exploit of three bugs in Facebook's code that were introduced with the addition of a new video uploader in July of 2017*" [5].

We have to specify that the examples listed here are just the tip of the iceberg, the full visible part cannot be covered, not even to mention the immersed part of it.

In this context, it becomes important to know these SDs in order to eliminate them. It is in that respect that we conducted two case studies in the domain of education by studying the SDs of two systems. We did this, in order to identify their sources as factors that trigger them, and identify among these factors the ones that cause high severe impacts of the studied systems.

1.1.2 Context

According to Lehman, there are two classes of software: (1) software developed to meet a fixed set of requirements, and (2) software developed to solve a real-world problem, which changes with time. This second class of software is the evolving system (E-type system) [6]. We conducted this research on two E-type systems. We named them *system A* and *system B*.

In fact, these previous examples concern not all systems, but systems with evolving programs (*E-type* programs). These E-type programs "*are programs that depend on or interact with the real world. They must always be adapted to match any changes in the real world that affect whether the program satisfies its stakeholders 'objectives'*" [7]. Systems with E-type programs are called E-type systems. One of their characteristics is that E-Type systems are under the influence of different factors identified by Cook et al. [8]. There are (1) stakeholder factors such as system users and developers; (2) architecture factors such as internal system components (e.g. a relational database), and (3) global process factors such as the business processes of an organization. Each of these factors may be a source of defect in a system while requesting a change to it. In order to avoid financial losses similar to the examples we presented earlier, it becomes essential to identify the ones that may cause such severe impacts on this type of systems among these factors.

Another characteristic of E-type systems is their particularity to require a considerable financial budget for their development project, and to have a costly maintenance phase (up to 80% of their total cost). Furthermore, a large population of stakeholders over a long period (years) use them. Our selected studied systems present all of these characteristics. In chapter three, we will present these selected systems in more detail.

1.2 Research Gap

Even though one can argue that SDs are all made from human error, studies have shown that at the time the SD goes on live or manifest itself, they have been triggered by different sources. Different researchers have studied software systems in order to identify the sources of their errors [8], [9]. Considering the list of examples provided earlier in this section, we could observe that these sources vary from one to another. The origins of some of these SDs are related to the system itself or its internal components, e.g. the Facebook attack; while others were caused by the inability of system users to use the system, e.g. the case of the British passport delay. Others are related to business processes, e.g. Knight Capital group case, and finally, a SD in a different system can cause a crash of another system, e.g. the L.A airport case. This raises two fundamental questions: Firstly, which origins or sources of SDs cause more losses to E-type systems, to the system owner as well as to the community using these types of systems? And secondly, derived from the preceding question, how to identify these sources? To provide an answer to these questions are the goals of this research. In the next section, we will define our research question in detail, present the methodology as well as the concept of this research, and then present the plan of this thesis.

1.3 Research Question

The main research question we address in this thesis is *"Which types of trigger factors generate the most severe SDs on a given E-type software system?"*. Answering this question led us to a related question, *"How to identify the origins of severe defects on evolving information systems?"*.

In order to answer our main research question, we conducted case studies on two E-type systems. We used a case study methodology to study these SDs in their natural environment, and to get the sense of how they happened and were treated. We studied the change history data of these systems. In addition, in order to provide an answer to the main question, we divided the question into two sub-questions as follows:

(1) The first sub-question aims to identify factors that trigger most SDs. To answer this sub-question, we studied two systems by classifying their SDs based on their trigger factors. In fact, we identified the trigger factor for each SD. To perform this classification, we used *EVOLIS framework* [9], a framework which proposes the grouping of "*factors having a direct influence on information systems*" [9]. We presented the results of this part in detail in our first published paper (see Appendix 1).

(2) The second sub-question consists of evaluating the impact that a SD has on a given E-type system. To answer this second sub-question, we also performed a second classification of the SDs of the same systems by classifying these SDs based on their severity impact. We used a severity scale model to do this classification. We also validated the results of this classification in our second published paper (see Appendix 2).

Finally, we combined the results of both sub-questions in order to answer the main question of this research. We presented the results of this part in two published papers (see Appendix 3 and 4).

Furthermore, answering this research question led us to find and propose a method on how to determine these severe trigger factors of a given E-type system. We will present this method in chapter three (see section 3.4). In the next section, we will present how we conducted this research.

1.4 Methodology: Case Study Presentation

To conduct this research, we used a quantitative case study methodology. According to Benbasat et al. [10] one of the reasons the case study methodology is viable in the information systems field is that *"the researcher can study information systems in a natural setting, learn about the state of the art, and generate theories from practice."* In fact, in order to observe SDs in their natural settings, we conducted two case studies on two different systems. The first case study was on a school management system used to manage students' grades and different certificates. We named this system "*system A"*. The second case was also on a school management system, but this system function was to help school directors in planning their school classes and assign teachers to their duties. We named this second system "*system B"*. Both systems are used for more than 95,000 students and for more than 10,000 teachers. These systems are developed in-house using the scrum agile method. In the next section, we will present the concept under which this research falls.

1.5 Concept of Study: Software Defects Management

1.5.1 Software Defects Management

The SDs management is crucial to any software team and software owners. There are different studies, which attempt to provide a solution on how to control or manage these SDs [11]–[13]. SD management consists of identifying SDs, collecting them, correcting, and mining them in order to subtract knowledge and understand their characteristics. This research mainly focused on their mining aspect. The mining of defects helps the software development teams to reduce the cost of correcting them, to detect defective modules, and to have efficient resource planning. There are different choices in mining software engineering data, e.g. code base data [14], execution traces data, change history data, mailing lists [15]. For this research, we analyzed the change history data of both systems. Different researchers propose models, tools, and schemas for mining SDs. The most prominent ones are: taxonomies [16],[17], root cause analysis [18], classification schemes and standards SDs [19],[20]. Wagner presents a complete definition of these approaches as follows: "*Defect taxonomies are categorizations of faults, mostly in code, that are based on the details of the implementation solution, e.g., Wrong type declaration, wrong variable scope, or wrong interrupt handling. A well-known example of this kind is the taxonomy of Beizer* [21]. *An even more detailed approach is root cause analysis where not only the faults themselves are analyzed, but also their cause, i.e., the mistakes made by the development team. The goal is to identify these root causes and eliminate them to prevent faults in the future. Root cause analysis has, for example, been used at IBM* [22]. *In general, root cause analysis is perceived as rather elaborate and the cost/benefit relation is not clear. Therefore, defect classifications aim at reducing the costs, but sustain the benefits at the same time. The categorization uses more coarse-grained defect types that typically have multiple dimensions."* [13]. We conducted our research under the umbrella of the defect classification approach.

1.5.2 Software Defects Classification

In the software life cycle, the classification of defects presents many advantages [23]. There are different existing schemes and standards in classifying SDs [20]. (1) The Orthogonal Defect Classification (ODC) of IBM was developed in 1992 by R. Chillarege et al. [24] and it classifies defects across *"the dimensions (1) defect type, (2) source, (3) impact, (4) trigger, (5) phase found, and (6) severity"* [13]. (2) The HP Defect Origins, Types and Modes, the approach of Hewlett Packard, was developed by the HP software metrics in 1986 [25] and this scheme classifies the defects according to their types, their origins, and their mode [13]. (3) The IEEE standard 1044-2009 [19] is proposed by IEEE standard bodies on how to classify software anomalies. Other defect classification studies have performed SDs classification using classical data mining techniques and algorithm such association rules [26], Naïve Bayes Model [27], and clustering analysis [28]. In the next section, we will present the plan of this thesis.

1.6 Plan

The presentation of our research will be in two parts. In the first part, we present a synthesis of our research. In the second part, we will present the four papers we published in order to communicate the results of this research to the scientific community. The structure of the thesis is presented as follows:

Chapter 1 is the introduction. In this current chapter, we presented the context as well as the motivation of our work and the plan of the thesis.

In chapter 2, we will present the literature review. In this chapter, we will present the concepts and paradigms of information system evolution, of SD mining and software maintenance. We will also present software anomalies, in particular, the SD and its lifecycle.

In chapter 3, we will present the methodology we used to conduct this research. We will also present the systems on which the case studies were done and present the data we collected to perform this research. This chapter presents in detail the different steps we followed in order to identify trigger factors of severe SDs on a given E-type system.

In chapter 4, we will present our results as well as some difficulties we faced while conducting this research. We will also provide explanations for our findings, and finally, we will propose a method we named "the origins of severe software defects method" as the main contribution of this research.

Chapter 5 consists of presenting our theoretical and practical contributions before concluding our work with possible future works.

In order to give an insight of the papers we published to communicate our findings, we add them to this manuscript in the form of appendices. This part represents the second part of the manuscript.

Appendix 1 consists of the first paper we published concerning the identification of SDs trigger factors on both systems (A and B) [29]. Here we also presented a conceptual tool in mining change requests (CRs) as SDs in our case.

Appendix 2 consists of the second published paper. In this paper, we presented the evaluation of SDs impacts based on a severity scale model [30]. We conducted this study on only system A. Here, we also presented a conceptual tool to improve the management of SDs.

Appendix 3 is the first paper we published concerning which type of SDs triggers have severe impacts on our studied E-type systems [31]. We conducted this study only on system A.

Appendix 4 is the second paper we published concerning which type of SDs triggers have severe impact on our E-type systems. In this paper, we also present our method to identify the factors triggering severe SDs on E-type systems [32]. The study was only on system B.

In Appendices 5, 6, and 7 we will present in detail the data analysis and two examples of the raw data from our studied systems.

In summary, our research consists of three studies. For each study, there is a question, a methodology, and the method used to answer this question, the results or solution we found, and the paper in which we communicate our findings. We summarized all this information in Table 1.1.

Table. 1.1 Research summary

	Study 1	**Study 2**	**Study 3**
Question (Which)	Sub-question 1: How to identify factors that trigger most SDs?	Sub-question 2: Evaluation of impacts a SD has on a given E-type system.	Research question: Which types of trigger factors generate the most severe SDs on a given E-type software system?
Methodology/Method or tools (How)	Case study/EVOLIS Framework [9]	Case study/Severity Attribute of IEEE Standard 1044-2009 [19]	Case study/severe SD trigger factors method
Findings (Solution, Results)	Change indicator conceptual tool [29]	SD Managerial tool [30]	Technology and architecture severe trigger factors as leading trigger factors [31]. A four-step method to identify trigger factors causing severe SDs on an E-type system [32] The origins of severe software defects method
Published paper (Where)	Appendix 1: *The Application of Change Indicators in Mining Software Repositories* [29]	Appendix 2: *A Conceptual Tool to Improve the Management of Software Defects* [30]	Appendix 3: *Severe Software Defects Trigger Factors A Case Study of School Management System* [31], and Appendix 4: *Classification of Software Defects Triggers: A Case Study of School Resource Management System* [32]

2 Literature Review

In this part, we will confront the environment of SDs by defining the existing paradigms and theories related to our research. A reported SD is called a “change request (CR)”. The implementation of solutions to a CR in the form of “change response” drives the evolution of E-type systems. In fact, Evolution is defined as the gradual development of something [33]. This thing can be an electronic system, an organism, or a software system. In our case, this thing is a software system. Different theories and laws have emerged in regards to the concept of evolution, e.g. Lehman’s laws [6], Moore’s laws [34], and the theory of Darwin [35] which is the best known evolution theory in the world. We will first introduce the information systems’ evolution paradigm followed by the software evolution, software maintenance activities, and then software anomalies with a particular attention to the SDs management field.

2.1 Information Systems’ Evolution Paradigm

“Information systems are combinations of hardware, software, and telecommunications networks that people build and use to collect, create, and distribute useful data, typically in organizational settings.” [36]. The Evolution of information systems (IS) is a new paradigm that emerged from the work of Truex et al. [37] who defined this paradigm as *“the notion of continuous change”*. Researchers have analyzed the question of information systems’ evolution from different perspectives. IS researchers tackle this question using either a technical lens or a managerial lens, both lenses are interdependent.

The first group of researchers provided an answer to the IS evolution question by analyzing characteristics that the IS must have, in order to evolve. This group constitutes the technical lens. A second group provided a more general management answer in the form of technology strategy, e.g., technology road mapping. Finally, a third group provided an answer to this question by designing artifacts such as EVOLIS framework [9], and models such as the software evolution laws [6] where Lehman presented five laws that guide the software evolution. These second and third groups fall under the managerial perspective.

- **Technical lens**

Under the technical lens, researchers have defined, analyzed, and presented the characteristic an IS component must possess for a system to evolve effectively over time during its life cycle. These systems are mainly E-type systems, which are computer programs that must undergo continual evolution to remain satisfactory and operate or address a problem or an activity in the real world [38]. The evolution is manifested through the maintenance of these systems. System maintenance is the implementation of the change response as a solution to software anomalies such as software failures or software defects. One of the important characteristics these systems must possess according to IS evolution paradigm is *flexibility*. Thus, J.H et al. [39] defined flexibility as the ability to respond to change. In regard to information systems, they insisted on the fact that "*a flexible system can be modified in a timely and cost-effective way in order to satisfy different requirements at different points in time.*" [39].

- **Managerial lens**

Under the managerial lens, IS researchers have proposed artifacts in order to study the IS evolution question; for example, frameworks such as EVOLIS [9]; a technology managerial tool such as technology road mapping; and models such as the Chapin et al. maintenance model [40]. Considering the technology road mapping tool, it represents a powerful technique for supporting technology management and planning, especially for exploring and communicating the dynamic linkages between technological resources, organizational objectives and the changing environment [41]. It also helps IS decision makers to design a concrete plan and be able to manage effectively the evolution of their IS. There are different types of technology road mapping such as integrated planning, long-range planning, and service/capability planning.

Among the IS elements, our focus is on software systems. The research field that studies the software system is called "software engineering". Software Engineering is defined as "*the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*" [42]. In the next part, we will present the software evolution paradigm, a branch of software engineering under which we conducted this research.

2.2 Definition of Software Evolution and Software Maintenance

2.2.1 Software Evolution

Software evolution is part of the IS evolution paradigm and falls under both the technical and the managerial lens approaches. It addresses only the software system as the principal subject of study. It emerged from the software maintenance and evolution work of Lehman in 1969. As a precursor of this domain, he theorized the software evolution by providing eight laws that are known as laws of software evolution [43]. These laws are summarized in Table 2.1. In addition to these laws, he also clarified the possible approaches to study software evolution. They made a distinction between the "*how*" of software evolution and the "*what*" of software evolution. Cook et al.[7] also presented a similar definition for both approaches as follows:

- "***Explanatory:*** *concerned with understanding causes, processes and effects. This approach attempts to achieve a holistic view and considers, for example, the impact of software evolution on the effectiveness of organizations and the planning of organizational change.*
- ***Process improvement*****:** *concerned with the development of better methods and tools. This approach addresses such questions as 'how should software engineering activities such as design, maintenance* [40],[44] *refactoring* [45], *reengineering etc., be used to manage the effects of software evolution?'*." [7].

Our research falls under the process improvement category, more precisely under the software maintenance activities. In fact, software evolution is manifested through the maintenance of these systems. System maintenance is the implementation of the change response as a solution to software anomalies such as software failures or SDs. In the next section, we will present the software maintenance activities.

Table. 2.1 Lehman's Laws of software evolution [43]

No.	Brief Name	Law
I. 1974	Continuing Change	E-type systems must be continually adapted, else, they become progressively less satisfactory.
II. 1974	Increasing Complexity	As an E-type system evolves, its complexity increases unless work is done to maintain or reduce it.
III. 1974	Self-Regulation	E-type system evolution process is self-regulating with distribution of product and process measures close to normal.
IV. 1980	Conversation of Organizational Stability (invariant work rate)	The average effective global activity rate in an evolving E-type system is invariant over product lifetime.
V. 1980	Conversation of Familiarity	As an E-type system evolves, all associated with it, developers, sale personnel, users, for example, must maintain mastery of its content and behavior [6] to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence, the average incremental growth remains invariant as the system evolves.
VI. 1980	Continuing Growth	The functional content of E-type systems must be continually increased to maintain user satisfaction over lifetime.
VII. 1996	Declining Quality	The quality of E-type system will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII. 1996	Feedback System (first stated 1974, formalized as law 1996)	E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback system and must be treated as such to achieve significant improvement over any reasonable base.

2.2.2 Software Maintenance

Software maintenance is the activity performed to change software systems. These changes are the sources of software evolution as well. Software maintenance is defined in IEEE Standard 1219-1998 [46] as: "*The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.*" In this domain, not only Swanson was the first researcher to provide a classification of maintenance activities [47], but his work also laid the groundwork to conduct studies with it. He classified software maintenance activities into four main groups [47]:

- **Adaptive maintenance** is a software maintenance activity performed in response to changes in data and processing environments**.**

- **Corrective maintenance** is a software maintenance activity performed in response to software failures such as processing failure and performance failure.
- **Perfective Maintenance** is software maintenance performed to eliminate processing inefficiencies and enhance performance.
- **Preventive maintenance** is a software maintenance activity performed to improve the maintainability of the system.

In the study field of software maintenance, other prominent researchers such as K.H. Bennett and V.T. Rajlich [48] argue that software maintenance is not a "single uniform phase" as portrayed in the traditional Software Development Life Cycle (SDLC) [49] but rather it is comprised of several distinct stages, each of them with different technical and business perspectives. Their proposed model is called "Staged model" [48] and comprised these following stages:

- **Initial development -** the first functioning version of the system is developed **[**48**]** .
- **Evolution -** the engineers extend the capabilities and functionality of the system to meet the needs of its users, possibly in major ways [48].
- **Servicing -** the software is subjected to minor defect repairs and very simple changes in function [48] .
- **Phase out -** no more servicing is being undertaken, and the owners seek to generate revenue from the use for as long as possible [48].
- **Close down -** the software is withdrawn from the market, and any users directed to a replacement system if this exists [48] .

Software teams perform software maintenance activities in order to correct software anomalies. In the next section, we will present different types of software anomalies, in particular the management of SDs.

2.3 Software Anomalies

2.3.1 Definitions

The Oxford dictionary defines anomaly as *"Something that deviates from what is standard, normal, or expected."* [50]. Similarly, IEEE standard 1044-2009 [19] suggested that the word anomaly is used to refer to any abnormality, irregularity, inconsistency, or variance from expectations. It may be used to refer to a condition or an event, to an appearance or a behavior, to a form or a function: *"Different terms such as error, bug, problem, incident, failure, fault, defects, have been used as synonyms of software anomaly"* [46].

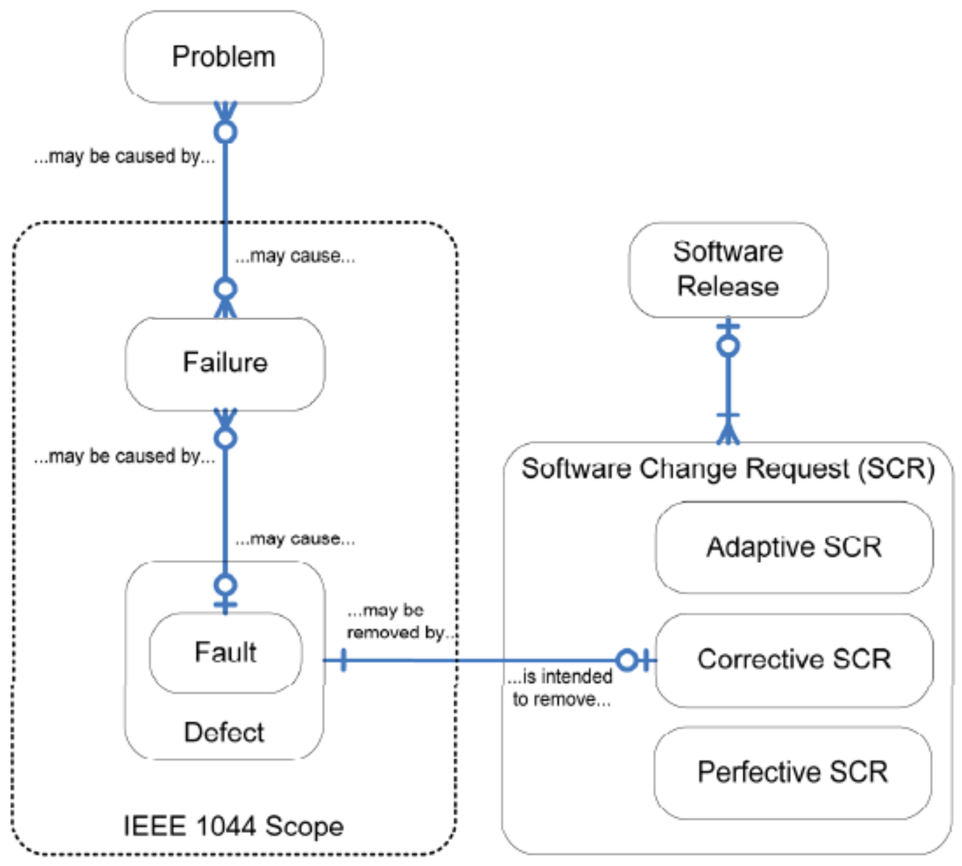
Thus, current studies used these terms to refer to software anomalies; some refer to it as a bug [51], or as a defect [24], [52], and others as an error [53]. The IEEE standard 1044-2009 [19] provides a simple and complete definition of the most significant anomalies in the context of software systems. We summarize these definitions in Table 2.2.

Table. 2.2 Software anomalies definitions [19]

Software Anomalies	Definition
Failure	Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits.
Error	A human action that produces an incorrect result.
Fault	A manifestation of an error in a software.
Defect	An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced (adapted from the Project Management Institute).

2.3.2 Relationships

The IEEE standard 1044-2009 [19] presents the description of the existing relationship between these terms (see Table 2.3). Furthermore, they provide a simple entity relationship diagram to understand these existing relations (see Fig. 2.1).



NOTE 1—The rounded rectangles represent entities (things of interest) and the lines connecting the rectangles represent relationships between entities. The symbols at the line ends indicate the number of entities at the end of the line. An open circle near a line end indicates that as few as zero are permitted (i.e., participation is optional); the absence of the circle indicates at least one is required (i.e., participation is mandatory). The three-legged "crows feet" symbol indicates that many entities are permitted to participate; the absence of the crows feet symbol indicates that no more than one entity may participate. A rounded rectangle appearing within another rounded rectangle indicates a parent–child relationship, wherein the contained entity is a subtype of the containing entity (supertype). The relationships represented graphically in this diagram are further described in Table 1.

NOTE 2—This diagram is not intended to mandate a particular notational methodology, and it is not intended as a schema for a database.

Fig. 2.1 IEEE standard 1044-2009's relationships modeled as an entity relationship diagram [19]

Table. 2.3 Relationships among software anomalies [19]

Class/entity pair	Relationships
Problem-Failure	A problem may be caused by one or more failures. A failure may cause one or more problems.
Failure-Fault	A failure may be caused by (and thus indicate the presence of) a fault. A fault may cause one or more failures.
Fault-Defect	A fault is a subtype of the super type defect. Every fault is a defect, but not every defect is a fault. A defect is a fault if it is encountered during software execution (thus causing a failure). A defect is not a fault if it is detected by inspection or static analysis and removed prior to executing the software.
Defect-Change **Request**	A defect may be removed via completion of a corrective change request. A corrective change request is intended to remove a defect. (A change request may also be initiated to perform adaptive or perfective maintenance.)

2.3.3 Lifecycle

In this part, we will present the occurrence of these anomalies in the life cycle of a software system. In the traditional SDLC model, a software system life cycle starts with the requirement phase [49]. In this phase, the user sends a software system requirement to the development team. Followed by the specification phase where the requirements are formalized in terms of output, input, and functionalities. The third phase is the design phase where the architecture of the system is determined. In the fourth phase, which is called the implementation phase, a development team implements the specifications in the form of codes. In implementing the coding of these different modules, a software team may introduce an error into the system. Before deploying the system to users, the system is tested. At this point, when software teams identify system errors they refer to them as software defects. The new software defect will be corrected if the software developers detect it. If not, it will be part of the deployed system to the users. In this case, according to the IEEE standard 1044-2009 [19] , this defect becomes a system fault. System fault may cause one or more failures. In case it causes failures, the system users will report those failures to the software maintenance team. In the last phase, which is the maintenance phase, a software team analyzes software failures to identify the fault, which is causing the failure(s) and corrects it. This correction activity is expressed in the form of a CR. In this case, the change is named a corrective CR. The correction takes the form of a corrective maintenance in this case. The adaptive and the perfective CR are the other types of CRs (see Fig. 2.1).

We must specify that this presented lifecycle differed from the one we have in this research. The reason being that the methodology used to develop our studies' system is the agile scrum method [54]. We will present this methodology in detail in the next chapter (see section 3.3.1), and present our SD lifecycle in the same chapter (see section 3.3.2).

Among these software anomalies, our focus will be on the software defects as they are the main subject of this research. In the next section we will present this software anomaly and its actual position within the IS and computer science field in more detail.

2.4 Software Defects Management

Nowadays, the management of SDs does not only consist of identifying, assigning, and correcting them but also in mining them. IEEE standard 1044-2009 defines a defect as: *"An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced"* [19]. Not only the software defects (SDs) are present in the whole life cycle of a software product, but different studies also proved that 80% of the total cost of the software life cycle is associated with the management of the SDs [23]. Having this high impact on the software product, SDs management must be crucial to software teams as well as to organizations.

In the last decade, SDs management has received a considerable amount of attention from researchers. In fact, SDs management has been the center of interest for many studies in different software studies' subdomains such as software project management, software engineering and evolution [12], [30], [55]–[57]. Due to the diversity of these studies, we group them into branches based on their interest in SDs management.

2.4.1 SDs Collection and Storing

The first branch deals with questions such as how to collect and store these SDs. Studies related to this branch provided answers to questions such as how to collect SDs or which SDs characteristics must be documented [55]. These studies propose solution tools named bug-tracking systems to help to collect SDs. They are in the form of a central hub accessible by project managers and software developers to manage the software products. Some of these online tools are Jira [56] and Bugzilla [57].

2.4.2 Assigning and Solving of SDs

The second branch deals with questions such as how to assign SDs to developers or how to deal with the problem of an SDs duplication [58]. The research in this branch proposes techniques and methods such as algorithms to automatically assign SDs to the right developer [59]–[61] and also techniques to eliminate the duplication of SDs [59].

2.4.3 SDs Triage and Mining Approaches

The third branch deals with the triage and the mining of SDs. There are different studies which propose solutions on how to mine SDs [60], [61]. The defects are the source of software failures and problems. Software failures are defined as "*Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits"* [19]. In the software life cycle, the mining of defects presents many advantages [23]. Researchers as well as practitioners in this branch proposed schema and taxonomies for mining SDs. Well-known schemas are (1) The Orthogonal Defect Classification (ODC) of IBM [24]: this model was developed in 1992 by R. Chillarege et al. [24] and it classifies defects across "the dimensions (1) defect type, (2) source, (3) impact, (4) trigger, (5) phase found, and (6) severity" [62]; (2) The HP Defect Origins, Types and Modes, being Hewlett Packard's approach. The HP software metrics developed this model in 1986 [25]. This scheme classifies the defects according to their types, their origins and their mode [13], the root cause analysis [18], and standards like the IEEE standard 1044-2009 [19]. In the same context, they also apply data mining methods such as the Naïve Bayes Model [27], clustering [28], or the regression model [23] to classify SDs. In fact, the classification of the defects helps the software development teams to reduce the cost of correcting SDs, and to detect defective modules. These studies are not an exhaustive list of studies related to SDs management. The value generated from SDs has made its management crucial to software teams as well as to organizations.

In order to identify the origins of SDs, The ODC, HP defects origins, root cause analysis and other enumerated mining schemas limited their analysis on software code. They analyzed this internal component (software code) of software systems in other to identify existing errors. Thus, they only focus on the technology factors as origins of SDs. Other internal and external factors that influence E-type systems such as systems' users and business processes are not addressed. The goal of this research is to fill this gap by proposing a method. This method helps identifying the source of SDs by considering internal as well as external SDs trigger factors. Furthermore, it also helps us to answer our research question, which is to identify the ones triggering most of the severe SDs on a given E-type system among these factors.

After reviewing the concepts and backgrounds of this research, we will present how we conducted this research in the next chapter.

3 Methodology

For our research, we conducted two case studies on two different systems. We named them System A and System B. The method used to conduct these studies is described in the following steps:

- Pre-step 0. Data collection: this pre-step consists of collecting SDs of the E-type system to study. In our case, we collected the SDs of system A and B from the Jira repository [56].
- Step 1. Identification of trigger factors for each SD: in this step, we classify the SDs based on the EVOLIS framework [9] in order to identify their trigger factors. In this research, we refer to this step as "*EVOLIS classification*".
- Step 2. Evaluation of the impact a SD has on the studied E-type system: here we classify the same SDs based on their severity level using the severity attribute of IEEE standard 1044-2009 [19]. We refer to this classification as "*Severity classification*".
- Step 3. Identification of SDs that cause high severe impact on our studied E-type systems: at this level, we classify the SDs based on both the EVOLIS framework and the severity attribute of IEEE standard 1044-2009 [19]. We named it "*EVOLIS and Severity classifications".*

We performed these steps on each system studied. In the next section, we will present the case study methodology, followed by the presentation of the studied systems. Then, we will proceed to present our method in general, and then present each step in detail.

3.1 Case Study Definition and Types

Case study research can be either a qualitative, a quantitative research methodology, or both [63]. There are numerous definitions of case research [64]–[68]. Nevertheless, Benbasat et al. [10] provided the one that is most suitable in the IS research field. They define case research as follows: *"A case study examines a phenomenon in its natural setting, employing multiple methods of data collection to gather information from one or a few entities (people, groups, or organizations). The boundaries of the phenomenon are not clearly evident at the outset of the research and no experimental control or manipulation is used"* [10]. Yin [65], [69] and Stake [70] make major contributions with their works to the field of case studies research. Based on their work, Baxter and Jack [71] proposed a résumé of existing types of case study research (see Table 3.1). In the context of this research, our goal is to observe the similarity between cases. Thus, we adopted the multiple-case studies approach for this research.

Table. 3.1 Type of case studies research [71]

Case Study Type	Type Definition
Explanatory	This type of case study would be used if you were seeking to answer a question that sought to explain the presumed causal links in real-life interventions that are too complex for the survey or experimental strategies. In evaluation language, the explanations would link program implementation with program effects [69].
Exploratory	This type of case study is used to explore those situations in which the intervention being evaluated has no clear, single set of outcomes [69].
Descriptive	This type of case study is used to describe an intervention or phenomenon and the real-life context in which it occurred [69].
Multiple-case studies	A multiple case study enables the researcher to explore differences within and between cases. The goal is to replicate findings across cases. Because comparisons will be drawn, it is imperative that the cases are chosen carefully so that the researcher can predict similar results across cases, or predict contrasting results based on a theory [69]
Intrinsic	Stake [70] uses the term intrinsic and suggests that researchers who have a genuine interest in the case should use this approach when the intent is to better understand the case. It is not undertaken primarily because the case represents other cases or because it illustrates a particular trait or problem, but because in all its particularity and ordinariness, the case itself is of interest. The purpose is NOT to come to understand some abstract construct or generic phenomenon. The purpose is NOT to build theory (although that is an option; [70]).
Instrumental	Is used to accomplish something other than understanding a particular situation. It provides in sighting to an issue or helps to refine a theory. The case is of secondary interest; it plays a supportive role, facilitating our understanding of something else. The case is often looked at in depth, its contexts scrutinized, its ordinary activities detailed, and because it helps the researcher pursue the external interest. The case may or may not be seen as typical of other cases [70].
Collective	Collective case studies are similar in nature and description to multiple case studies [69].

3.2 Why Case Study Methodology?

To be able to analyze the impact of system defects reported by system users as well as system developers, and the change derived from the occurrence of this failure, we have to observe two systems in their natural settings. We use case study methodology to conduct this research since it addressed contemporary phenomena in their natural context. Considering the contemporary phenomena, we are looking at CRs done on software systems developed using the agile method. In the natural context, we are considering two sub-units of the same institution, which are in charge of managing the SDs of these systems.

Furthermore, to know if this methodology is more suitable for our research project or not, we also identify and compare the characteristics of our research with the ones proposed by Benneth et al. [48] by providing an answer to each of these 11 points. In fact, they presented 11 key characteristics of case studies research must possess (see Table 3.2) [10]. The results of the comparison between the proposed characteristics and our research project are presented in Table. 3.3.

Table. 3.2 Key characteristics of an IS case study [10]

1. Phenomenon is examined in a natural setting.
2. Data are collected by multiple means.
3. One or few entities (person, group, or organization) are examined.
4. The complexity of the unit is studied intensively.
5. Case studies are more suitable for the exploration, classification and hypothesis development stages of the knowledge building process; the investigator should have a receptive attitude towards exploration.
6. No experimental controls or manipulation are involved.
7. The investigator may not specify the set of independent and dependent variables in advance.
8. The results derived depend heavily on the integrative powers of the investigator.
9. Changes in site selection and data collection methods could take place as the investigator develops new hypotheses.
10. Case research is useful in the study of "why" and "how" questions, because these deal with operational links to be traced over time rather than with frequency or incidence.
11. The focus is on contemporary events

Table. 3.3 The characteristics of our case project.

Key Characteristics of Case studies	***Key Characteristics of our Research.***
Phenomenon is examined in a natural setting.	*Our project consists of studying software defects of two systems in an organizational context. These systems had been observed in their natural setting which is the organization managing them.*
Data are collected by multiple means.	*We collected data from two main sources: the first one is a software failure reporting system (EasyVista* [72]*). The second is a software repository system (Jira* [56]*).*
One or few entities (person, group, or organization) are examined.	*The studied systems belong to one institution, with two different project groups in charge of each one of them.*
The complexity of the unit is studied intensively	*We analyzed each software defect in detail by looking at its description and summary.*
Case studies are more suitable for the exploration, classification and hypothesis development stages of the knowledge building process; the investigator should have a receptive attitude towards exploration.	*Our main goal is to classify the software defects, according to their severity and factors that trigger them.*
No experimental controls or manipulation are involved.	*We did not conduct any experimental or manipulation during this research.*
The investigator may not specify the set of independent and dependent variables in advance.	*No independent variable nor dependent variables were set in advance of the studies.*
The results derived depend heavily on the integrative powers of the investigator.	*We were able to combine in an innovative manner, both classifications to reach the main objective of this study.*
Changes in site selection and data collection methods could take place as the investigator develops new hypothesis.	*We had collected data from another software project to evaluate our hypothesis.*
Case research is useful in the study of "why" and "how" questions because these deal with operational links to be traced over time rather than with frequency or incidence.	*How to identify trigger factors causing the most severe SDs on E-type systems?* *How do software defect triggers affect the operation and the functionalities of the systems? These are questions we address with our studies.*
The focus is on contemporary events.	*Managing software defects activities represent up to 80% of the total cost of a software system. In addition, SDs cause huge financial losses to government bodies, organization, and individuals.*

3.3 Presentation of the Institution

The institution we choose to conduct this study on is a governmental educational institution, which is in charge of providing software solutions to the public college schools in one of the French regions in Switzerland. Among the systems that they provided, two main ones draw our attention and have the necessary characteristics for our study. In fact, we choose these systems because they continuously evolved since their deployment and the project team has kept track of their SDs. In other words, they have a complete SDs report history data. They are:

System A: this system is a grading management system for schools. More than 10,000 teachers use it to manage the evaluations, school reports, and grades of more than 95,000 students. They also use it to deliver school certificate as well as control the students' attendance. It was deployed in the ending of 2012 (see Table 3.4).

System B: it is used to manage the repartition of teachers and students in different schools and classes. It addresses the administrative management of public schools in the region. It is used by more than 80 schools with almost 1,500 users (deans and directors), and it manages the records of students (more than 95,000), teachers (more than 10,000), and 5,000 teaching assistants (see Table 3.4).

Table. 3.4 Summary of studied systems

	System A	**System B**
Data Range	January 2015-April 2016	January 2015-April 2016
Number of SDs	675	581
Number of Users	15000	1500
Development Budget (including maintenance)	More than 1.5 million dollars	More than 2.5 million dollars

Both systems are developed based on the agile scrum method. In the next section, we will present this method.

3.3.1 Agile Methodology and Scrum Method

For the purpose of our study, let us briefly look at the characteristics of agile methodology and scrum method in particular.

There are different definitions of this agile methodology. For Henderson-Sellers and Serour [73], agility involves both the ability to adapt to different changes and to refine and fine-tune development processes as needed [73]. Lee and Xia [74] define software development agility "*as the software team's capability to efficiently and effectively respond to and incorporate user requirement changes during the project life cycle"* [74]. Agile methodology proposes different methods such as a dynamic software development method. The popular one is extreme programing, Scrum [75]. To provide software systems' solutions to its clients, the institution uses the Scrum method in developing these systems. "*Scrum method consists of focusing on project management in situations where it is difficult to plan ahead, with mechanisms for 'empirical process control'; where feedback loops constitute the core element. Software is developed by a self-organizing team in increments (called 'sprints'), starting with planning and ending with views. Features to be implemented in the system are registered in a backlog. Then, the product owner decides which backlog items should be developed in the following sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the scrum master, is in charge of solving incidents that stop the team from working effectively*" [76].

3.3.2 Software Defect Life Cycle in the Studies' Projects.

In this section, we will present the state of software anomalies in the context of this research. The software studies are developed using the scrum methodology; meaning that the scrum master—in this case the project leader—is the person that controls the evolution of the software project, by requesting software changes from the software development team and adding new functionalities to the existing system. The process of the SD and CR goes as follows:

System users report failures they encountered using a particular system to a help desk member. This help desk person reports the failure into a bug repository system (EasyVista [72]) to be solved. A sub-team of the help desk analyzes the reported failure and addresses it. In case this sub-team person provides a solution to correct this failure, they communicate the solution to the client and close the case. If not, the failure is then reassigned to the software development project team. They analyze this failure in turn to identify the software defect causing it. After identifying the SDs, they proceed to request a change to correct the detected SD. These SDs and their requests are saved into a software repository tool which, in our case, is Jira [56]. From this point on, the outsourced software development team takes charge of the implementation of a response to achieve the requested change. We summarized this process in Fig. 3.1

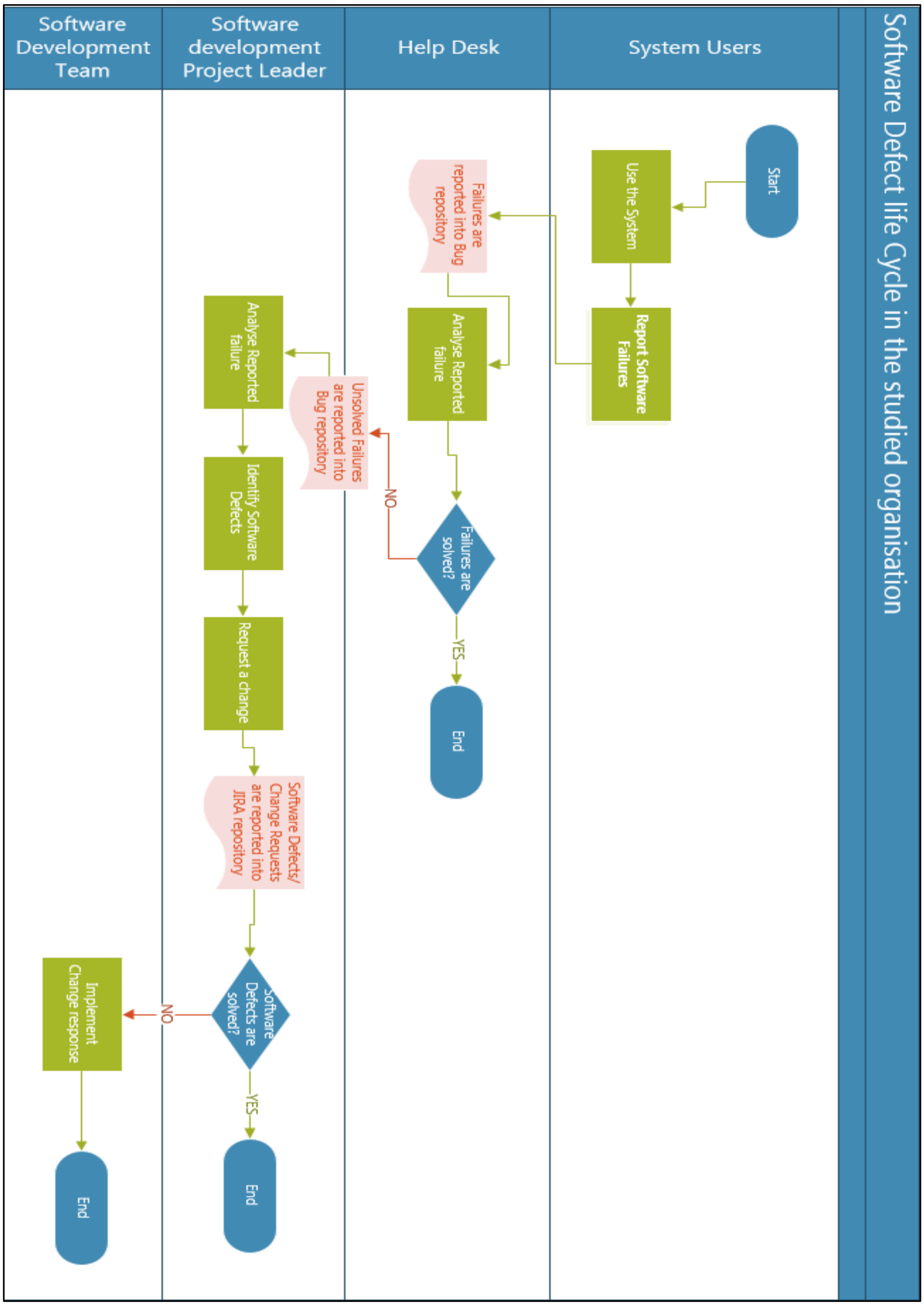

Software Defect life Cycle in the studied organisation
Software Development Team
Software development Project Leader
Help Desk
System Users
Start
Use the System
Report Software Failures
Failures are reported into Bug repository
Analyse Reported failure
Failures are solved?
YES
End
NO
Unsolved Failures are reported into Bug repository
Analyse Reported failure
Identify Software Defects
Request a change
Software Defects/ Change Requests are reported into JIRA repository
Software Defects are solved?
YES
End
NO
Implement Change response
End

Fig. 3.1 Software defect life cycle within the studied projects

3.4 Severe SD Trigger Method

As said in the beginning of this chapter, our method of conducting this research consisted of a pre-step and three main steps. We applied this method on both case studies. The overview of this method goes as follows:

In the pre-step (0), we collected SDs of a selected system for studying. In the first step (1) (EVOLIS classification), we took each SD to find its trigger factor or source. In the second step (2) (Severity classification), we took again each SD and evaluated its severity impact (cost) on the system. We then took each couple of "trigger factor-impact" and classified them into two groups: "the severe group" and "the no-severe group". This classification is based on our severity-weighting model. We will present in detail this severity-weighting model in the next chapter (see section 4.3.1). In the final step (3), we only concentrated on the severe couples and ranked them according to the level of damage they may do on system operations (EVOLIS-Severity classifications). We present these steps as well as the activities we conducted under each one of them in Fig. 3.2.

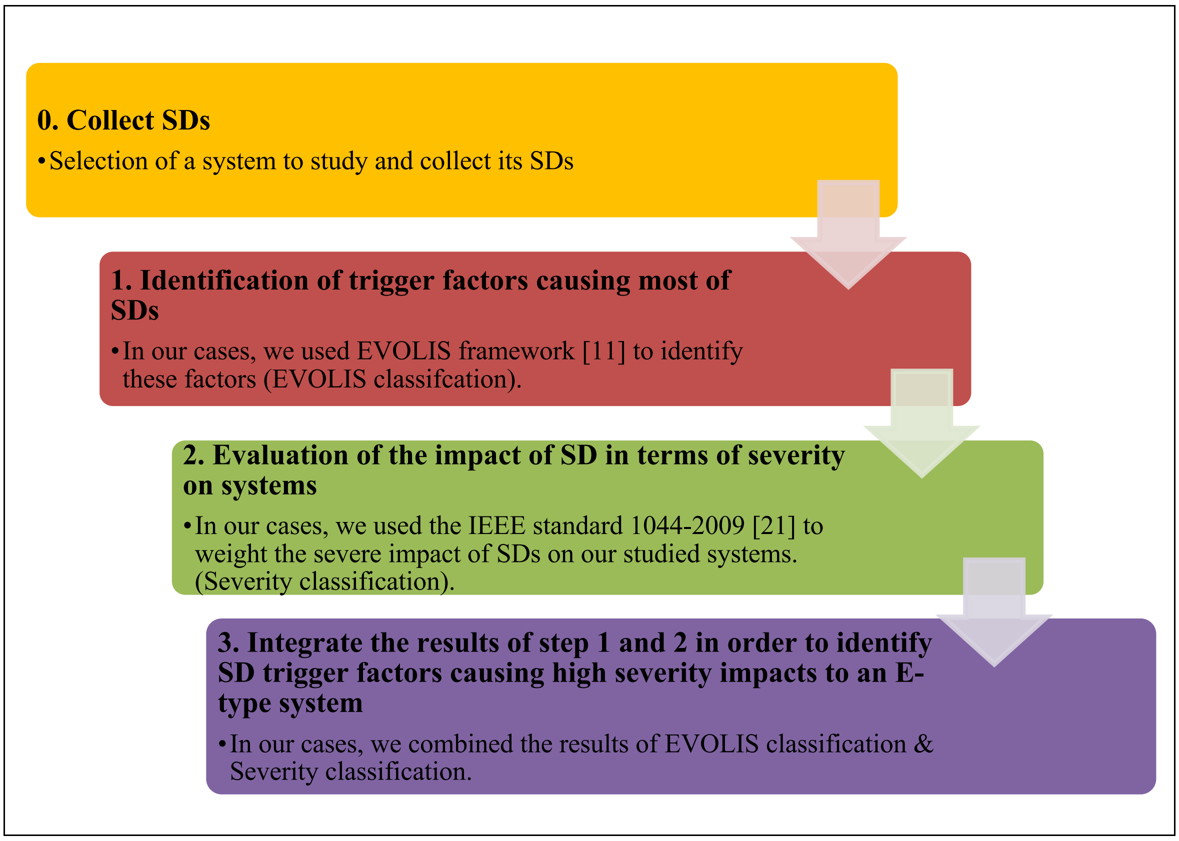

0. Collect SDs
•Selection of a system to study and collect its SDs
1. Identification of trigger factors causing most of SDs
•In our cases, we used EVOLIS framework [11] to identify these factors (EVOLIS classifcation).
2. Evaluation of the impact of SD in terms of severity on systems
•In our cases, we used the IEEE standard 1044-2009 [21] to weight the severe impact of SDs on our studied systems. (Severity classification).
3. Integrate the results of step 1 and 2 in order to identify SD trigger factors causing high severity impacts to an E-type system
•In our cases, we combined the results of EVOLIS classification & Severity classification.

Fig. 3.2 Steps to identify trigger factors causing most of severe SDs to a system

In addition, we illustrate this process with the IS architecture trigger factors block of EVOLIS framework [9] (see Fig. 3.3). We present this framework in one coming section (see section 3.6.1). We applied the same process to each SD analyzed of both studied systems. In the next section, we will present each step of this process in detail.

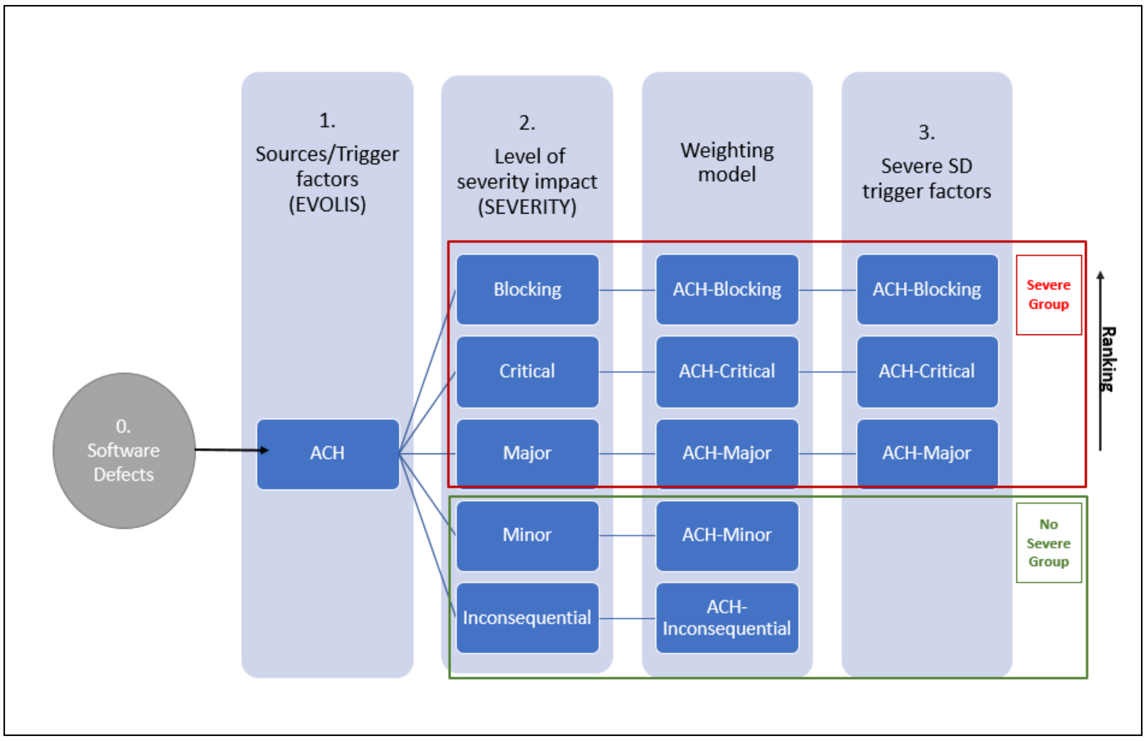


Fig. 3.3 Illustration of the steps to identify severe SD trigger factors using the IS architecture factors

3.5 Pre-step 0: Data Collection

We collected our data from two main databases. The first one is EasyVista [72]: it contains both information related to software incidents or failures, as well as SDs or CRs. The second one is Jira [56] and only contains data related to SDs or CRs. We only considered data related to SDs in our project. Therefore, in both cases, we only focused on data related to the SDs and CRs. We performed our data collection in three steps:

- The first step is to choose systems having a software defect repository. This was an important step because not all systems and organizations we contacted have put this repository in place.
- The second step consists of collecting the SDs data. For the selected systems, we collected archival data from a software repository for both systems. The data were presented in tabular format (Microsoft Excel) with the following information:

For each system, there is the SD unique ID; the name of the person or organization who reports the software failure; the help desk person who receives the complaint; the date of reporting, and the date the defect was fixed; the name of the person who implemented the CR, the department or service concerned with the reporting; the description of the SD; the summary (resumé) of the change to implement; and finally, the project member team who confirmed the defect.

- The third step consists of designing a database (see Fig. 3.4) for each studied system. In this step, we also proceeded to identify essential data that we needed to conduct our research. We had grouped these data in a database. We present the essential data retained for our study in Table 3.5. In this table, not only do we describe the essential data, but we also present the classification data. The essential data needed for our study are divided among four tables (see Fig. 3.4). In fact, our database is made up of four tables. There are EVOLIS, System, Severity, and the SoftwareDefects tables. The relationships among these tables are described as follows: A system has zero or more SDs. A SD falls under one type of severity or not. Similarly, a SD may fall under one type of EVOLIS block (category) or not.

Table. 3.5 Data attributes and their description

Attributes	**Definition**
ID	The unique number of each reported SD
Reported Date	The SD reporting date
Description	The description of the SD
Résumé	The possible action to perform to correct the SD
Type of Request	The type of request from the SD reporter
Severity	The type of Severity
EVOLIS	The type of SD trigger factor
Solved Date	The SD solving date
Status	If the SD has been solved or not

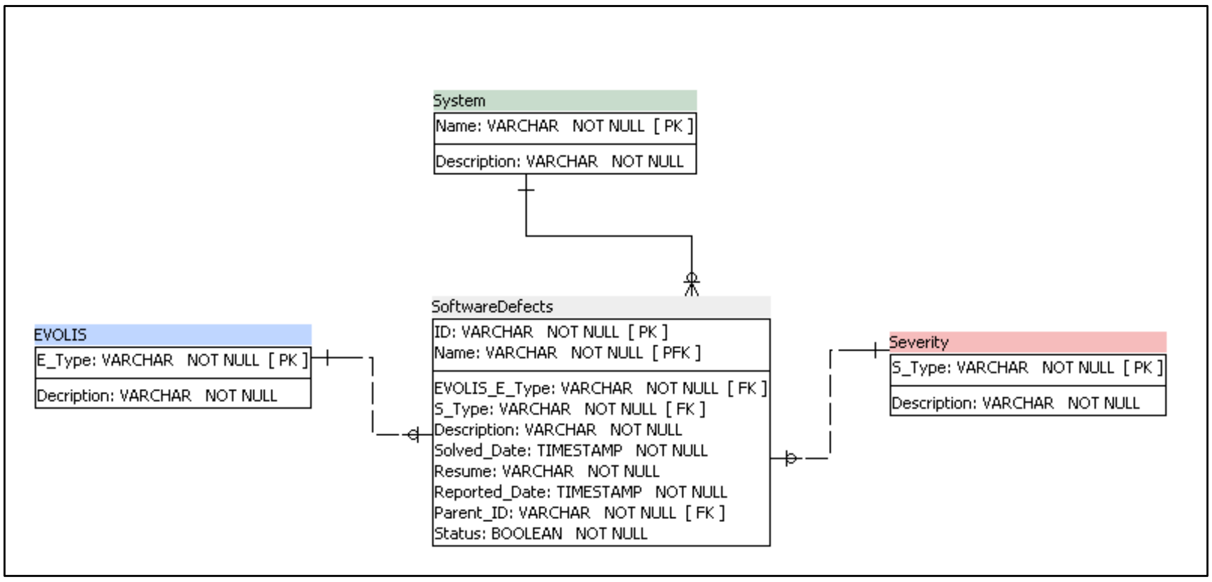

System
Name: VARCHAR NOT NULL [PK]
Description: VARCHAR NOT NULL
SoftwareDefects
ID: VARCHAR NOT NULL [PK]
Name: VARCHAR NOT NULL [PFK]
EVOLIS_E_Type: VARCHAR NOT NULL [FK]
S_Type: VARCHAR NOT NULL [FK]
Description: VARCHAR NOT NULL
Solved_Date: TIMESTAMP NOT NULL
Resume: VARCHAR NOT NULL
Reported_Date: TIMESTAMP NOT NULL
Parent_ID: VARCHAR NOT NULL [FK]
Status: BOOLEAN NOT NULL
EVOLIS
E_Type: VARCHAR NOT NULL [PK]
Decription: VARCHAR NOT NULL
Severity
S_Type: VARCHAR NOT NULL [PK]
Description: VARCHAR NOT NULL

Fig. 3.4 Project' database schema

We collected data over the period of three years from January 2013 until December 2016. However, for specific use cases we selected segments of data over a period of 16 months from January 2015-April 2016 for both systems A and B. System A has nine released versions over this period. The first version of the system A had been released mid-2012. System B has 10 released versions over the same period of time. System A had 675 SDs and system B had 581 SDs (see Table 3.4). In the following sections, we will present the classifications of these SDs.

3.6 Step 1: Trigger factors Identification (EVOLIS Classification)

We conducted this first classification on both systems. The main goal of this first classification is to group SDs according to their trigger factors. We validated the results of this first classification in our first paper published in *Trends and Advances in Information Systems and Technologies Volume 2, 2018* [29] (see Appendix 1). We did this classification for three main reasons:

1. To identify SD groups that have a sudden rise over a period of time.
2. To propose a model to manage SDs or CR implementation and manage their unexpected rise.
3. To improve decision making in the domain of software maintenance.

We did this classification based on the EVOLIS framework [9]. In the next section, we will present this framework in detail.

3.6.1 The EVOLIS Framework

One of the concepts we used to conduct this research is the EVOLIS (EVOLution of Information Systems) framework [9]. This framework was proposed by two researchers of the Evolution lab in the Information System department of HEC Lausanne. In fact, Dr. A. Métrailler presented this framework as part of his PhD thesis directed by Dr. T. Estier. The main purpose of this framework is to help software portfolio managers to manage the evolution of their systems.

3.6.1.1 Definition

The EVOLIS framework [9] groups the factors having a direct influence on a system into four blocks. It provides a means of studying software defect reports with their CR based on their trigger factors. According to the authors, EVOLIS [9] framework classified SDs as an evolution based on factors that trigger them: "*EVOLIS can be caused by a large variety of factors: bugs that need to be fixed, users that wish to have new functionalities, new market opportunities that require new software features, performance standards that the system must reach, technical changes in the environment with which the system must interact, obsolescence of applications and so on*" [9]. EVOLIS presents four main categories or blocks of SDs with a fifth-block as the cost to compare the different phases of system evolution with each other (see Fig. 3.5). The cost block is related to the financial impact that will result from a change in a system*: "It is the consideration the cost-benefit in case of evolving an IS"* [9]. The four main blocks are the IS architecture change requests block, the Technology change requests block, the IS/user fit change requests block, and the Business/IS alignment change requests block. In the next section, we will present each of these blocks in detail.

3.6.1.2 IS architecture

The IS architecture change requests block (ACH), is a group of factors that triggers software change based on integration and interoperability needs. According to the authors, this type of change request concerns "*different types of integration evolution, namely an evolution of integration among components of the system, among business functionalities, or an integration with systems outside of the company.*" [9].

3.6.1.3 Technology

The Technology change requests block (TCH) is related to factors that trigger software change based on the operational needs of the software as well as the hardware platforms as information system components. As an example, they stated that *"when reason like* *performance, updates, preventive maintenance and so on motivate evolutions of the software or hardware."* [9].

3.6.1.4 IS/user fit

The IS/user fit change requests block (UI) is related to factors that trigger software change based on the system users' satisfaction in terms of ease of use and usefulness of the system [77]. They defined it as any request related to the user interface, the user documentation, and aptitude to use the system. Simply said, the authors "*classify as IS/user fit each activity during an evolution regarding directly users or when the evolution only alters the fit between IS and users without altering business functionalities*" [9].

3.6.1.5 Business/IS alignment

The Business/IS alignment change requests block (B.IS) is a group of factors that triggers software change based on the IS alignment with business process and activities. It "*addresses the co-alignment between business and information systems."* [9]. There are two types of alignment under this category: company external environment alignment, and evolution-oriented alignment.

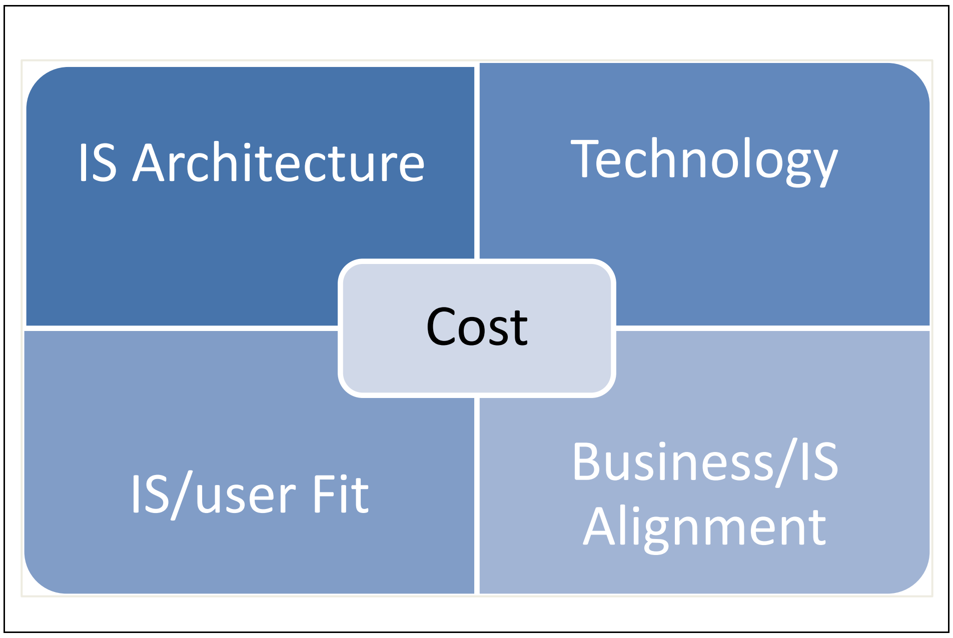


Fig. 3.5 The five Blocks of EVOLIS (Adapted from the of EVOLIS Framework) [9]

3.6.2 EVOLIS Classification

As described earlier, each software failure report is characterized by a source, a description, and a help desk person handling the failure. The failures that could not be solved by the help desk team became SDs. These SDs were saved in the repository (Jira [56]). The classification of the SDs is done based on our EVOLIS classification approach. In the next section we will present in detail how we adapted the EVOLIS framework to classify our SDs data.

3.6.2.1 Our EVOLIS classification adapted method

Overall, our EVOLIS classification adapted method is described as follows:

Based on the definition of EVOLIS blocks and SD descriptions, we defined seven classes (see Table 3.6). Each of these seven classes is associated with one of the four blocks of EVOLIS (see Table 3.7). Each of these classes is composed of two or more subclasses. Each subclass is then characterized by defined key factors. Factors are combined keywords that are identified based on the semantic analysis of the SDs résumé and description. The semantic analysis is conducted on the raw description and résumé of the SDs we collected for both systems.

E.g., the EVOLIS user fit block is composed of user, user interface, and user testing classes. The user interface class is made up of two subclasses: ease of use and system usefulness. The ease of use subclass is characterized by key factors such as display, button, click, screen, color and visual. These factors are keywords identified while conducting a semantic analysis on SD's descriptions in French.

Table. 3.6 Our defined seven classes

1. SDs related to the user ability to manipulate the system
2. SDs related to the user interface
3. SDs related to system error or system bug
4. SDs related to another system different from the system in use
5. SDs related to the business and processing rules
6. SDs related to the system database and based mainly on user access privileges
7. SDs related to testing of the system done by the user.

Table 3.7 presents the classification of the seven main classes into the EVOLIS blocks.

Table. 3.7 Classification of the seven classes into EVOLIS SD trigger factor categories

		User	**User Interface**	**User test**	**System Bug**	**Another system**	**Rules/ Process**	**User Privilege/ Database**
EVOLIS	Business/IS alignment (B.IS)						●	
	IS/user fit (UI)	●	●	●				
	Technology (TCH)				●			●
	IS architecture (ACH)					●		

3.6.2.2 *Application of our EVOLIS classification adapted method*

We applied this method in analyzing both the data we collected for system A and B. For each SD, we proceeded as follows.

First, we conducted a semantic analysis of the reported description and résumé of the SD to identify some keywords. Second, based on the semantic analysis, we combined two or more keywords as key factors. Third, based on the SD' key factors, we assigned the SD to its corresponding subclass. Fourth, based on the SD' subclass, we classified the SD according to the corresponding class. Finally, we classified the SD in one of the four EVOLIS blocks based on its corresponding class identified previously. In summary, the classification of each SD is done by following these steps:

1. Semantic analysis of the SD description and résumé to identify keywords
2. Identification of factors based on the SD description and résumé keywords
3. Classification of the SD into a subclass
4. Classification of the SD into one of the seven classes
5. Classification of the SD into one of the four blocks of EVOLIS

E.g. here are some examples of how we identified a trigger factor of a SD using our EVOLIS classification. These examples are based on the raw data in French (see Appendix 7) we collected for both systems (see Table 3.8).

Table. 3.8 Identification of SD trigger factors examples

Steps	Examples	
Semantic analysis of the SD description in French *(keywords are marked in bold)*	Raw Description : *"Ici, **le bouton** + est **affiché** dans l'**écran** (capture 2). Il existait car il y avait des travaux qui étaient concernés par cet **affichage**. Ces travaux ont été déplacés et/ou supprimés. Il n'y a plus de travaux liés. Le + reste affiché. L'ouverture de la **fenêtre** (capture 1) montre qu'il n'y a pas de travaux. Si les travaux n'existent pas ou plus, le + ne devrait pas s'**affiche**r."* Résumé: ***Fenêtre** "Note de l'élève des groupes liés" persistante*	Raw Description : *Bonjour, Je n'arrive pas à transférer à Prilly primaire une élève futur 1P qui va déménager de Crissier à prilly ! "Une **erreur** est apparue !"* Résumé : ***erreur applicative/de développement***
Identification of key factors in the SD description and résumé	**Le bouton affiché**, **écran**, **affichage**, **fenêtre**. (In English, these keywords are button, screen, window, display)	**Erreur, erreur applicative, erreur de dévéloppement** (In English, these key factors are error, system error, software error)
Classification into a subclass	Key factors display, and window fall under the ease of use subclass	System error fall under the subclass of system fault
Classification of the SD into one of the seven classes	Ease of use subclass falls under the user interface class	The system fault subclass falls under the system Bug class
Classification into one of the four blocks of EVOLIS	The interface class is associated to the user fit block of EVOLIS. Thus, the trigger factor of this SD is an IS user-fit factor.	The system bug class is associated to the technology block. Thus, the trigger factor of this SD is a technology factor.

The results of applying this EVOLIS adapted method on system A and B are as follows:

- Results of System A EVOLIS classification

EVOLIS	**ACH**	**B.IS**	**TCH**	**UI**	**Total**
2015	***133***	***93***	***144***	***149***	***519***
Jan	12	8	16	7	43
Feb	10	5	12	4	31
Mar	15	13	19	16	63
Apr	22	11	10	12	55
May	4	8	10	7	29
Jun	9	2	19	11	41
Jul	1	1	2	6	10
Aug	1	4	9	13	27
Sep	15	8	8	22	53
Oct	16	4	9	10	39
Nov	23	19	18	23	83
Dec	5	10	12	18	45
2016	***12***	***47***	***26***	***71***	***156***
Jan	3	15	11	26	55
Feb	4	4	6	10	24
Mar	0	13	6	27	46
Apr	5	15	3	8	31
Total	***145***	***140***	***170***	***220***	***675***

- Results of System B EVOLIS classification

EVOLIS	**ACH**	**B.IS**	**TCH**	**UI**	**Total**
2015	***87***	***22***	***251***	***90***	***450***
Jan	7	2	25	12	46
Feb	7	3	11	2	23
Mar	6	2	20	5	33
Apr	7	1	17	17	42
May	12	4	31	13	60
Jun	17	3	59	15	94
Jul	5	3	20	9	37
Aug	4	2	17	1	24
Sep	6	1	18	6	31
Oct	3	1	4	5	13
Nov	4	0	12	3	19
Dec	9	0	17	2	28
2016	***20***	***8***	***73***	***30***	***131***
Jan	9	0	16	6	31
Feb	5	2	27	12	46
Mar	1	2	13	1	17
Apr	5	4	17	11	37
Total	***107***	***30***	***324***	***120***	***581***

The analysis of these results at this step led us to propose a "change indicator conceptual tool". The purpose of this conceptual tool is to help software maintenance teams and software portfolio managers to identify sudden rises of SDs by putting in place the right process. We will present this conceptual tool in the next chapter in detail (see section 4.2.4).

3.7 Step 2: Evaluation of SD Impact on E-type Systems (Severity Classification)

3.7.1 The Roles of Evaluating SD Severity Impact

The aim of this second classification is to distinguish the severity impact of each SD on our studied systems, and to improve the project management of mining SDs. One of the proposed classification methods recommended by IEEE standard 1044-2009 [19] is the classification of SDs based on severity. They argue that "*having a standard way to classify software anomalies enables better communication and exchange of information regarding anomalies among developers and organization"* [19]. In addition, "*it enables insight into the types of anomalies that organization produce during development of their project*" [19]. For these reasons, for our second classification we selected the severity attribute of IEEE standard 1044-2009 [19] (see Table 3.9).

There are three main goals for this classification:

1. To identify the degree of severity impact each SD has on the system.
2. To improve the project management of mining SDs or CRs.
3. To improve SDs' mining decision making by implementing control measures.

3.7.2 Evaluation of SD Impact on our E-types Studied Systems

The IEEE standard 1044-2009 [19] defines the severity attribute as *"The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering."* [19]. The five values of severity are classified from the most significant to the least significant ones (see Table 3.9).

Table. 3.9 Severity values [19]

Attribute	Value	Definition
Severity	Blocking (B)	Operation is inhibited or suspended pending correction or identification of suitable workaround.
	Critical (C)	Essential operations are unavoidably disrupted, safety is jeopardized, and security is compromised.
	Major (Maj)	Essential operations are affected but can proceed.
	Minor (Min)	Nonessential operations are disrupted.
	Inconsequential (Inc)	No significant impact on operations.

As defined by the IEEE standard 1044-2009 [19], this classification is done based on the judgment of the impact a SD has on its system. This judgment is done by the people responsible for software engineering. The people responsible for software engineering are either the software maintenance team or the software development team. In determining a SD severity impact on a system, they must also consider the context of the organization in which the software system is used. In our case, the people responsible for software engineering are both the software development and the software maintenance teams.

For our studied systems, either the software development team or the software maintenance team or both make the determination of the severity values of the SDs. As recommended by IEEE standard 1044-2009 [19], the classification of SD severity is done as follows in our research:

- If the SD completely prevents the system to process its essential operations, or prevents an entire component of the system to be operational (complete shutdown of the system component) then it is considered having a blocking severity impact on the system. Any SD of this sort falls under the Blocking severity level.

E.g., “*Enseignant : impossible d'assigner un enseignant dans un nouvel établissement s'il a le statut EXT.”*. Assigning a teacher to a class is an essential operation of this system. The description of this SD states that “Teacher: Impossible to assign a teacher to a new school if he has EXT status (external status)”. This SD completely prevents the system to process this essential operation, thus this SD has a blocking severity impact on the system and falls under the Blocking severity value category.

- If a SD partially prevents the system to run its essential operation, and compromises the security of the system, then this SD is considered having a critical severity impact on the system. Thus, this SD falls under the Critical severity level.

- If a SD affected essential operations of a system without interrupting them to proceed, then the SD is considered having a major severity impact on the system. This type of SD falls under the Major severity level.
- If a SD affects only the non-essential operation of a system, then the SD is considered having a minor severity impact on the system. Thus, the SD is classified under the Minor severity level.

E.g., "*Dans le cas de rôles multiples (SUPPORT + ADMIN + CDIR), la création d'un message en mode ADMIN porte la mention CDIR.*". Creating a message using the system is a non-essential operation. The résumé of this SD in French states, "In the case of multiple roles (SUPPORT + ADMIN + CIDR), the creation of a message in ADMIN mode is marked CDIR.". This SD does not prevent the system administrator to create or send a message but only shows the wrong message marked in the admin mode. Thus, this SD has a minor severity impact on the system. It is classified under the Minor severity level.

- If a SD does not have any significant impact on neither the essential operation nor the non-essential ones, then it is considered having an inconsequential severity impact on the system. Thus, a SD of this sort falls under the Inconsequential severity value category.

Based on this "if then" approach, we proposed a summary of the characteristics a SD has under each severity level (see Table 3.10). We proposed this table to inform others SD mining researchers and practitioners on how to determine the severity impact or value of a SD (see Table 3.10).

In fact, to determine the severity level of a SD, an answer must be provided for each of these following questions:

- Question 1 (Q1): Did the SD have an impact on essential operations?
- Question 2 (Q2): Did the SD have an impact on non-essential operations?
- Question 3 (Q3): Can essential operations proceed?
- Question 4 (Q4): Can non-essential operations proceed?

There are three possible answers to each of these questions, either: *NO or YES or Partially.* The combination of the answers to these questions determine the level of severity of a SD. E.g., if the analysis of the SD provides the following answers (YES to the first two questions and NO to the last two questions), then this SD falls into the Blocking severity level.

Table. 3.10 Evaluation of the severity value of a SD

	Impact of a SD		**System operational status**	
	Q1. Impact on Essential Operations	Q2. Impact Non-essential Operations	Q3. Essential Operations can proceed	Q4. Non-Essential Operations can proceed
Severity Value				
Blocking	YES	YES	NO	NO
Critical	YES	YES	Partially	NO
Major	Partially	YES	Partially	NO
Minor	NO	YES	YES	Partially
Inconsequential	NO	Partially	YES	YES

For the purpose of this study, we define any SD as severe as long as it belongs to one of these severity levels: (1) Blocking (B), (2) Critical (C), and (3) Major (Maj). The Minor (Min) and the Inconsequential (Inc) are not considered as severe SDs. Furthermore, we also introduced a weighting model to express the value of each SD's severity according to their severity impact. We will present this model and the results of applying it, in the next chapter (see section 4.3). We also presented this weighting model and the results of this second classification in detail in our second and third paper. In addition, in the second paper, we also proposed a "SD managerial conceptual tool" to help SD mining teams to align their mining strategy and objectives with the ones of the software owners. We will present this second conceptual tool in the next chapter (see section 4.3.3). The second paper is published in *Business Modeling and Software Design – 8th International Symposium, BMSD 2018* [30] (see Appendix 2), and the third paper is published in *Digital Science 2018 Advances in Intelligent Systems and Computing, vol 850 pp 389-396* [33] (see Appendix 3).

- Results of System A Severity classification

Severity	Blocking	Critical	Major	Minor	Inconsequential	Total
2015	***51***	***60***	***266***	***127***	***15***	***519***
Jan	2	3	28	10	0	43
Feb	1	5	14	11	0	31
Mar	7	8	34	14	0	63
Apr	2	5	37	9	2	55
May	3	1	20	5	0	29
Jun	0	2	25	14	0	41
Jul	0	1	5	4	0	10
Aug	2	7	8	7	3	27
Sep	5	11	18	15	4	53
Oct	7	4	22	6	0	39
Nov	15	8	37	20	3	83
Dec	7	5	18	12	3	45
2016	***25***	***24***	***64***	***43***	***0***	***156***
Jan	8	9	25	13	0	55
Feb	5	3	10	6	0	24
Mar	5	9	15	17	0	46
Apr	7	3	14	7	0	31
Total	***76***	***84***	***330***	***170***	***15***	***675***

- Results of System B Severity classification

Severity	Blocking	Critical	Major	Minor	Inconsequential	Total
2015	***57***	***112***	***135***	***145***	***1***	***450***
Jan	3	13	12	18	0	46
Feb	2	4	9	8	0	23
Mar	4	6	14	8	1	33
Apr	5	9	7	21	0	42
May	16	13	15	16	0	60
Jun	6	32	29	27	0	94
Jul	7	3	16	11	0	37
Aug	4	8	10	2	0	24
Sep	4	10	8	9	0	31
Oct	3	2	3	5	0	13
Nov	0	4	5	10	0	19
Dec	3	8	7	10	0	28
2016	***18***	***18***	***29***	***65***	***1***	***131***
Jan	3	7	7	14	0	31
Feb	4	3	13	26	0	46
Mar	4	2	2	9	0	17
Apr	7	6	7	16	1	37
Total	***75***	***130***	***164***	***210***	***2***	***581***

3.8 Step 3: Identification of Trigger Factors Causing Severe SDs on our Studied Systems (Classification based on both EVOLIS and Severity)

We combined the two results in order to answer our main research question which is to "identify trigger factors that cause most severe SDs". Subsequently, we grouped these results into a two-dimensional table. Each dimension represents the results obtained for each previous classification project. We also presented these studies in detail in our third and fourth papers. This third paper is published in *Digital Science 2018 Advances in Intelligent Systems and Computing, vol 850 pp 389-396* [31] (see Appendix 3). The fourth paper is published in *ICITS19* [32] *Advances in Intelligent Systems and Computing, vol 918* (see Appendix 4).

- System A

EVOLIS / **Severity**	ACH	B.IS	TCH	UI	Total
Blocking	**18**	**24**	**19**	**15**	**76**
Critical	**12**	**21**	**30**	**21**	**84**
Major	**100**	**66**	**93**	**71**	**330**
Minor	**15**	**28**	**27**	**100**	**170**
Inconsequential	**0**	**1**	**1**	**13**	**15**
Total	145	140	170	220	675

- System B

EVOLIS / **Severity**	ACH	B.IS	TCH	UI	Total
Blocking	**16**	**2**	**51**	**6**	**75**
Critical	**30**	**5**	**74**	**21**	**130**
Major	**34**	**8**	**92**	**30**	**164**
Minor	**27**	**15**	**107**	**61**	**210**
Inconsequential	**0**	**0**	**0**	**2**	**2**
Total	107	30	324	120	581

In the next section, we will discuss these results and highlight the contributions we made out of them.

4 Results Analysis

In this section, we will present our results and findings. For each classification, we will present and discuss its results and other solutions these results lead us to. We analyzed the results in three steps: first, we started with the EVOLIS [9] classification, followed by the Severity classification, and finally, the integrated classification for both systems.

4.1 Data Collection of SDs (Step 0)

As shown in chapter 3 (see section 3.5), the results we obtained in this pre-step were determinant to proceed with the rest of our studies. In fact, in this pre-step, we obtained data with sufficient quality necessary to perform our three classifications (steps).

4.2 First Classification: EVOLIS Classification (Step 1)

4.2.1 Analysis

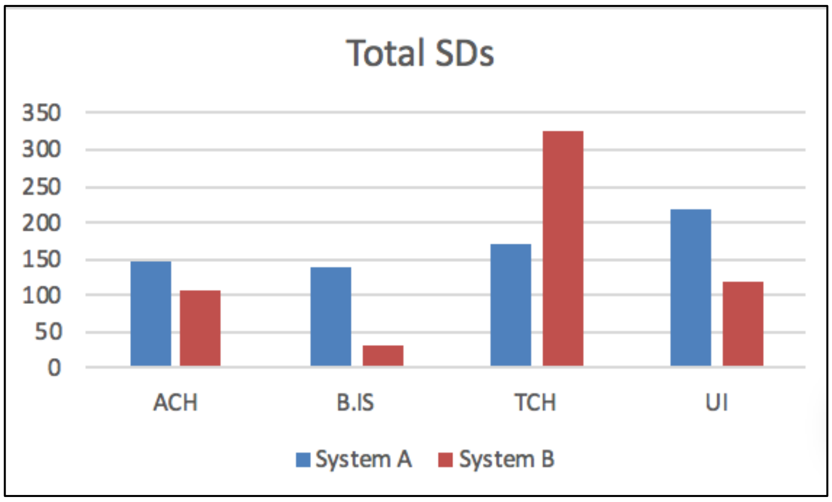


Fig. 4.1 SDs of system A and B classified based on EVOLIS [9]

We obtained the following results by only focusing on a fixed period for both systems. In fact, knowing the time each SD was reported helps us consider only the SDs which happened during this time interval for both systems. The time period set for this study went from January 2015 to April 2016. In addition, being aware of the reporting time of SDs helped us to do a comparison of both systems during the same interval of time. We did this comparison in order to observe their evolution over the same period and identify some of their characteristics such as the number of SDs per trigger factors during this time interval.

In overall, we can say that over the studied period the majority of SDs were triggered either by the IS/user fit factors (220 SDs) for system A, or by the Technology factors (324 SDs) for system B (see Fig. 4.1). For each system, the results showed that:

For system A, the IS/user fit factors were in the lead with 220 SDs, followed by the Technology factors (170 SDs), the IS architecture factors (145 SDs), and finally the Business/IS alignment factors (140 SDs).

For system B, the Technology factors (324 SDs) came first, followed by the IS/user fit factors (120 SDs), then the IS architecture (107 SDs), and finally the Business/IS alignment factors (30 SDs).

Therefore, to sum up based on our first classification; there are two main factors trigger SDs in the context of the studied systems. They are either IS/user fit or Technology trigger factors. Going further in our analysis by looking at these results monthly, we have faced two main questions that we will present in the next section, as well as the solution we propose to solve them. We presented similar results concerning this EVOLIS classification in our first published paper in *Trends and Advances in Information Systems and Technologies Volume 2*, 2018 [29] (see Appendix 1).

4.2.2 Change Indicator Concept

4.2.2.1 Monthly analysis

In order to study the evolution of our E-type systems over time in more detail, we decided to use the month as unit of time. In fact, to choose the month as time unit allowed us to observe the variations of the number of SDs per trigger factors over this specific period and compare it to the results we obtained over the entire period of our study. Considering this monthly period helped us identifying the sudden rise of SDs. This particular rise is recurrent over the years for a specific period, e.g., for each year covered by our research, we found that the month of March has a number of user fit SDs which are problematic. This observation led us to propose the "change indicator conceptual tool" in order to detect similar anomalies and control the evolution of the SDs of E-type systems.

During the analysis of the EVOLIS classification results for both systems, we observed that the number of SDs varied not only among different blocks or among categories of triggers, but also from one month to another. These observations raised two questions: (1) *How to evaluate the status of SDs level? (2) On which bases can one prove that SDs of one month are overloaded compared to other months?* To answer these questions, we have introduced a change indicator concept. We derived this concept from the existing Key performance indicator.

4.2.2.2 Brief presentation of Key Performance Indicator (KPI)

In management, performance measurement [78] is crucial for the survival of each business unit: *"The measurement of performance is important because it identifies current performance gaps between current and desired performance and provides an indication of progress towards closing the gaps."* [79]. Key performance indicators [80] are set to evaluate the performance of business units. They are used for different objectives. In case the defined objectives are not reached, different actions are put in place to reach them. Similarly, in our context, based on questions that we identified in classifying these SDs, we introduce a conceptual tool as solution to identify when goals defined by system owners have not been met, and when to take corrective actions to reach those defined goals. In the next section, we will present the application of the change indicator on both systems.

4.2.2.3 Definition the change indicator

The absence of standards to evaluate which month has the number of SDs overload compared to others has prompted us to introduce an indicator as the accepted limit of SDs over a fixed period of time. In our cases, we selected a monthly period. The purpose of this indicator is to alert software maintenance teams on the status of the SDs level in each category. In addition, this is set as a limit for each category of SDs above which SDs must not only be solved, but the concerned category may possibly be investigated.

The Oxford English dictionary defines indicator as "*A thing that indicates the state or level of something*." [81]. Based on this definition, we set the indicator artifact as an arithmetic value that informs us on the SDs level in each category. We defined the condition of this artifact indicator as follows: "*If the number of change requests for a particular month in a category is greater than twice the total average so far within the same category; then an investigation may be conducted for this month, within this category*". This investigation will lead to identify possible hidden problems, and derives actions to correct them. The condition of our indicators is expressed as follows:

n being the total number of months

And X, being the monthly Software defect number to test

$$If \quad X_n > 2(\frac{\sum_{i=1}^{n-1} X_i}{n-1})$$

Then investigate & adopt mitigation actions

We must underline that the set limit of measures to observe depends on the goal set by each software team or each organization [47]. In our case, our emphasis is on the sudden increase in SDs in a month compared to the number of SDs in previous months. The application of this formula implies that the change indicator calculated for the first month is irrelevant. Furthermore, in case of a progressive rise in SDs, another formula will be most appropriate for setting the indicators. Our indicator rule is only an example of the form a change indicator may have.

4.2.2.4 Implementation of the change indicator on system A

The results obtained by applying the change indicator to system A are summarized in Fig. 4.2 (see Appendix 5.2).

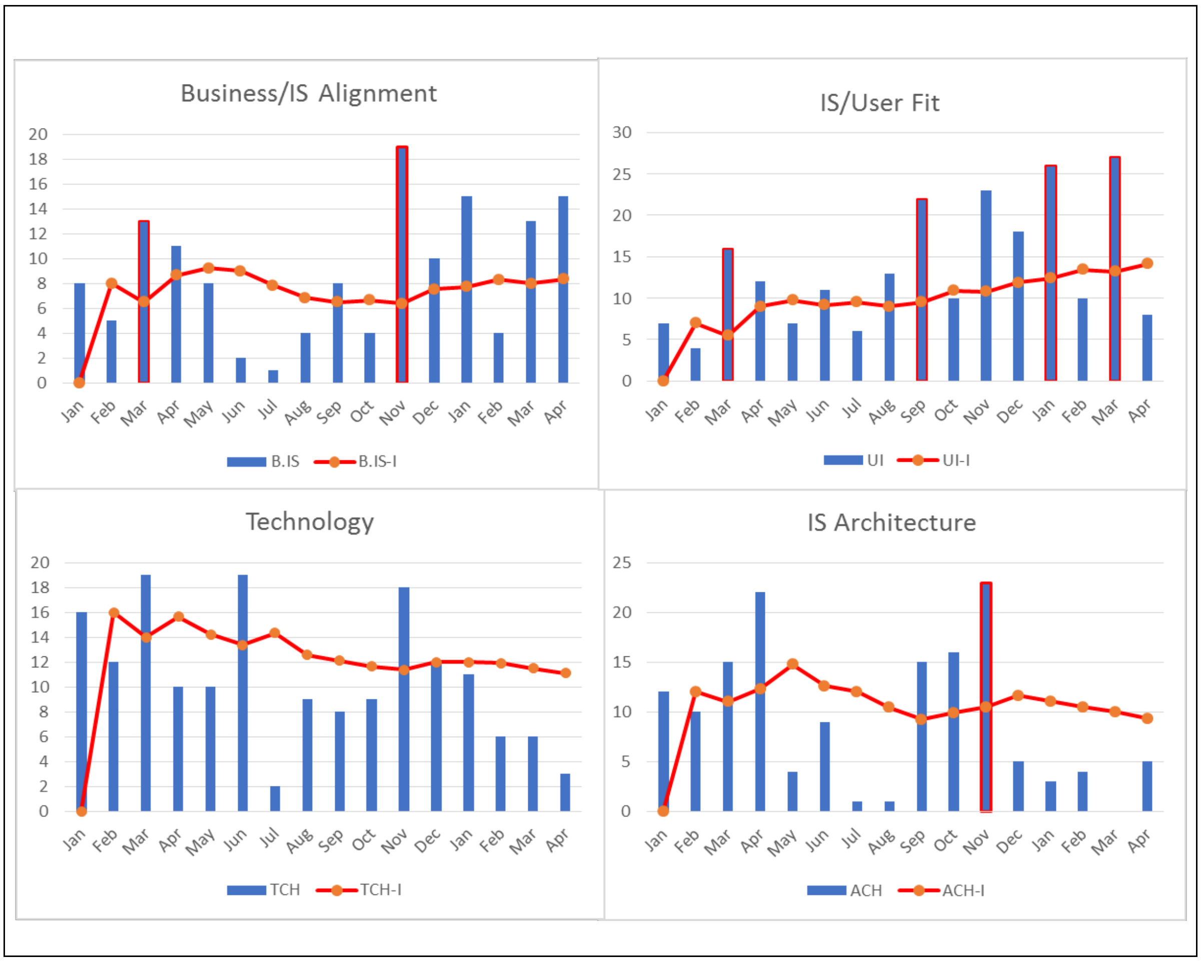


Fig. 4.2 Application of change indicator on system A

(ACH-I stands for IS architecture Indicator, B.IS-I for Business/IS alignment Indicator, TCH-I for Technology Indicator, and UI-I for IS/user fit Indicator).

4.2.2.5 Implementation of the change indicator on system B

The results obtained by applying the indicator to the system B are summarized in Fig. 4.3 and also presented in Appendix 6.2

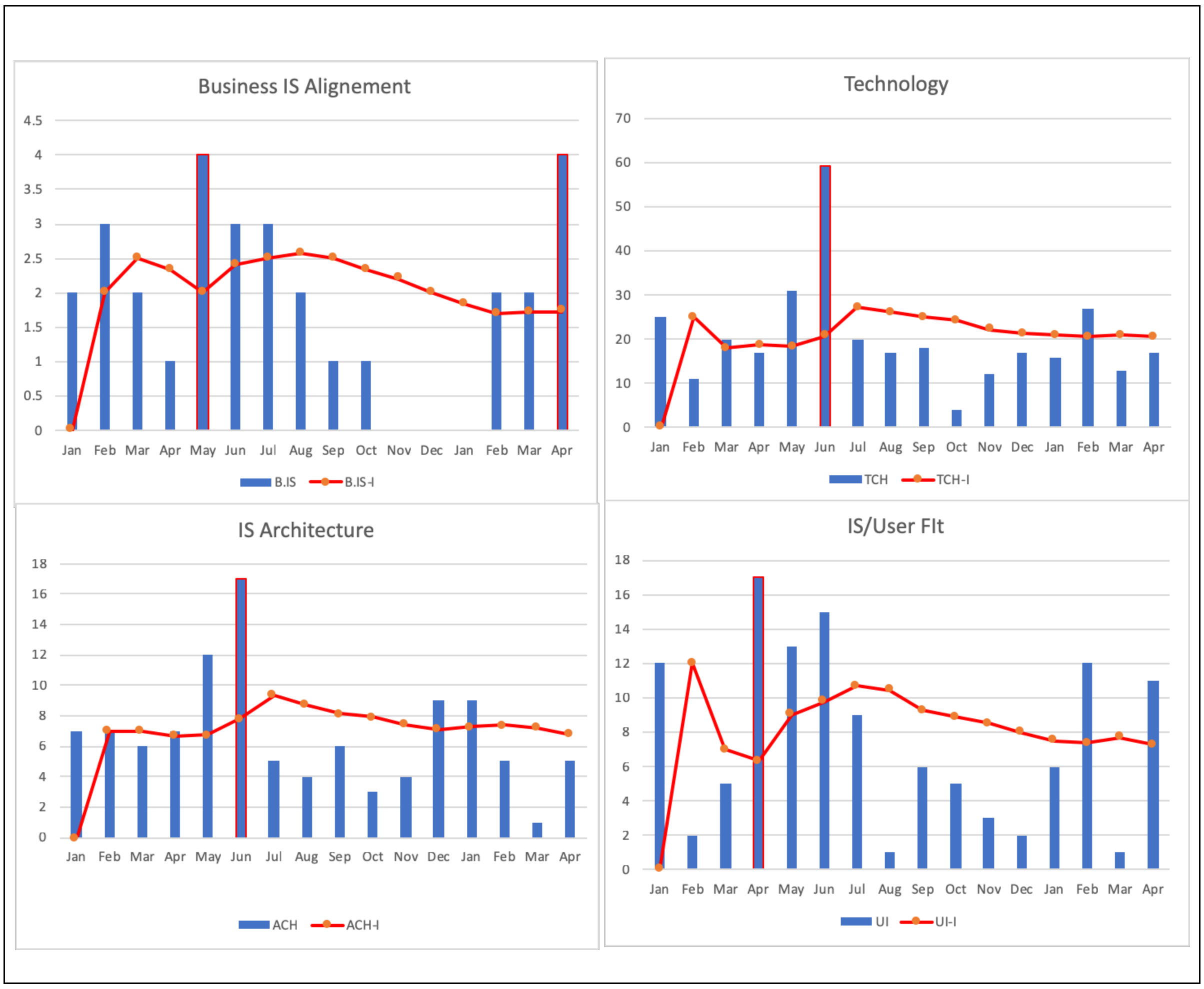


Fig. 4.3 Application of change indicator on system B

(ACH-I stands for IS architecture Indicator, B.IS-I for Business/IS alignment Indicator, TCH-I for Technology Indicator and UI-I for IS/user fit Indicator).

4.2.3 Discussion

Months with SDs that pass the set indicators are marked in red in both figures. Concerning system A, in the category of IS/user fit SDs we have four months (March 2015, September 2016, January 2016, and March 2016) for which the total number of SDs is two times greater than the average of previous SDs. Consequently, we investigated, and we found that those months correspond to a time period during which a special event took place—such as the start of a school period—where the system users demanded and reported a lot of system failures and defects they encountered when using the system. Both the IS architecture block and the Business/IS alignment block show the month of November 2015 as being a problematic month. When investigating this case, it was revealed that a module has been added to system A; this module had affected some processes as well as the interaction this system A had with another governmental system where student records were stored.

This implies that the interaction between system A and the other systems are crucial for the users to perform their activities. A possible set of actions to put in place in this situation would be the allocation of enough human resources to face the peak of these types of demands.

For system B, only the Business/IS alignment block has two months with SDs two times greater than the average. They are the month of May 2015 and April 2016. These months correspond to an introduction of new modules in this system. Both the Technology and the IS architecture block show the month of June as being problematic. Investigations reveal that a new version of the system was deployed in order to prepare the schools reopening in August that year.

4.2.4 Change Indicator Conceptual Tool

We proposed a conceptual tool as a summary to describe the main steps we followed in this first classification [29]. This conceptual tool describes actions to perform in order to mine SDs with the application of the change indicator.

The process to follow to classify SDs and set SDs indicators is summarized in these four steps:

1. Collection of the SDs into a repository: For this purpose, there are existing tools such as Jira [56].
2. Classification or triage of these SDs into categories: The triage features are integrated into the existing SDs' repositories cited in the previous step. In case this incorporated feature does not satisfy the software team or the project manager, there are other frameworks and models, or the traditional data mining techniques such as clustering or classification algorithms [82] with which this classification can be done. In our case, we used EVOLIS [9] as a CR framework to classify the SDs.
3. Set a limit for each type of SD over a defined period: At this level, one or more indicator must be defined to track the evolution of each type of SDs' category. In addition, in this step, the indicator definition depends on the objectives set by the organization [47] or the project team. Indicators can be defined as a ratio or as a target number to reach [81], [83]. In defining these parameters, the team must also consider the population size of stakeholders who can report a SD.

4. Investigation step: Define a set of actions to undertake when an indicator is reached, for instance, organize a user training section in case the category related to the user-fit indicators is reached.

Following, Fig. 4.4 where we present the diagram that describes our proposed four-step process of setting the change indicators**.**

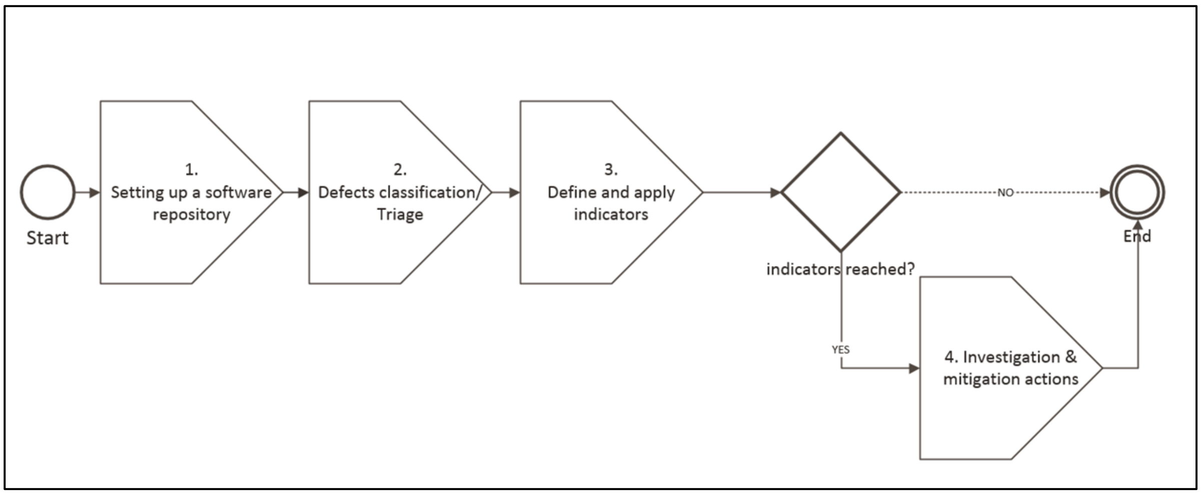

Start
1. Setting up a software repository
2. Defects classification/ Triage
3. Define and apply indicators
indicators reached?
NO
End
YES
4. Investigation & mitigation actions

Fig. 4.4 The steps in mining SDs and setting change indicator and mitigating actions [29]

As answer to the first sub-question “which SD factors trigger most SDs?”, we found that the factors triggering most SDs of an E-type system vary according to the system. In fact, for our studied systems, we found that IS/user fit for system A and the Technology trigger factors for system B are responsible for most of SDs. In the next section, we will present our second classification.

4.3 Second Classification: Severity Classification (Step 2)

Similarly, to the first classification, we obtained these results by focusing on a fixed period ranging from January 2015 to April 2016 for both systems. We set this fixed period in order to evaluate the impact SDs over this period have on each system. It also allowed us to compare the evolution of both systems during this time. Having this defined period helped us identifying the type of impacts these systems had over this period.

In fact, with this second classification, our main goal is to identify the number of SDs for each severity level. In overall, the SDs having a Major severity impact on the system are the highest for system A. System B has the Minor type as the highest. For each system, the ranking goes as follows: for system A, the Major type of SDs comes first with 330 SDs, followed by the

Minor type (170 SDs), the Critical type (84 SDs), then the Blocking type (76 SDs), and finally the Inconsequential with only 15 SDs (see Fig. 4.5). As stated in chapter 3, we validated the results of this severity classification in our second paper published in *Business Modeling and Software Design – 8th International Symposium, BMSD 2018* [30] (see Appendix 2).

For system B, the Minor type comes first with 210 SDs, followed by the Major type (164 SDs), the Critical type (130 SDs), the Blocking type with 75 SDs, and finally the Inconsequential type with only 2 SDs (see Fig. 4.5).

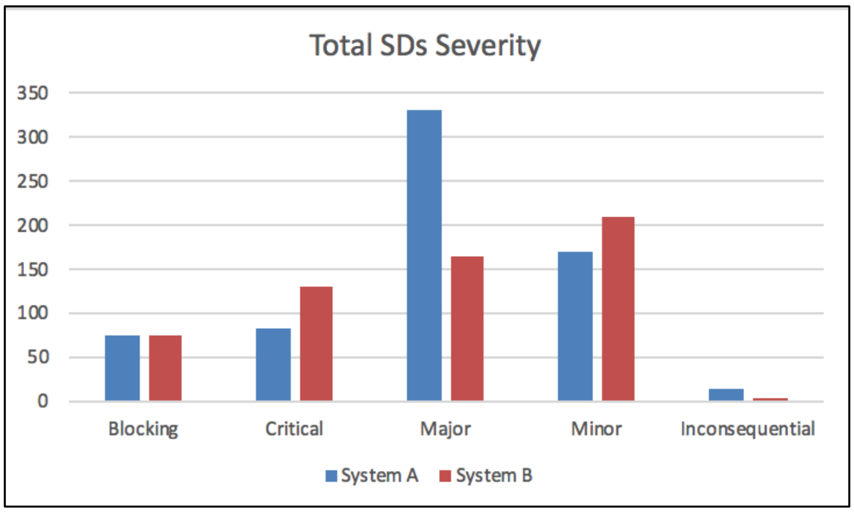


Fig. 4.5 SDs of system A and B classified based on their severity

4.3.1 Weighting Model

As stated in chapter three, we will present the weighting model in detail in this section. Doing the previous analysis, we realized that limiting the results only to the number of SDs for each severity level group raises an ambiguity. In fact, counting only the number of SDs per severity level does not give us the clear response on which level of impact has the highest severity damage on the system. E.g., how can we determine if five Blocking SDs have more effect on a system than eight Critical SDs? In order to clear this ambiguity, we associated a weighting factor to each level of severity according to their impact on the system's operation (see Table 4.1.). In fact, with this severity scale, we attributed the highest weight to the Blocking type, because they completely stopped the system operations, and we gave the least weight to the Inconsequential type, because they do not affect in any way the system's operations.

Table. 4.1 The weighting factors for the severity levels [31].

Severity level	Weighting Factor
Blocking	40%
Critical	30%
Major	20%
Minor	8%
Inconsequential	2%
Total	100%

We then calculated the weighted score (W) for each system based on the severity weight model (see table 4.2).

Table. 4.2 Weighted score for system A and B

	Weight	Number of SDs of System A	*Weighted A*	Number of SDs of System B	*Weighted B*
Blocking (B)	0.4	76	*30.4*	75	*30*
Critical (C)	0.3	84	*25.2*	130	*39*
Major (Maj)	0.2	330	*66*	164	*32.8*
Minor (Min)	0.08	170	*13.6*	210	*16.8*
Inconsequential (Inc)	0.02	15	*0.3*	2	*0.04*

Applying the weighting model showed us the following: for system A, the Major type has the highest weighted score with 66 followed by the Blocking type with a 30.4-weighted score and the Critical type with a 25.2 weighted score. The Minor type only scores 13.6 even though their number was twice as big as the Blocking SDs' number.

For system B, the Critical type is the highest with a 39-weighted score, followed by the Major type with a 32.8 weighted score, and the Blocking with a weighted score of 30. Even though the number of Minor SDs is the highest, their weighted score is only 16.8. Meaning, their impact on the system are half as many as the Critical SDs with 130 SDs initially. These results show us that the impact of SDs depends not on their number, but rather on the level of effect they can have on the system operations. Thus, counting only their number is inefficient to identify their effect on systems.

Similarly, we applied this weighting model on the monthly results to appreciate the real impact these SDs have on the systems on a monthly basis.

Table. 4.3 Monthly weighted scores for system A

Year	**Month**	**B**	***W-B***	**C**	***W-C***	**Maj**	***W-Maj***	**Min**	***W-Min***	**Inc**	***W-Inc***
2015	Jan	2	*0.8*	3	*0.9*	28	*5.6*	10	*0.8*	0	*0*
	Feb	1	*0.4*	5	*1.5*	14	*2.8*	11	*0.88*	0	*0*
	Mar	7	*2.8*	8	*2.4*	34	*6.8*	14	*1.12*	0	*0*
	Apr	2	*0.8*	5	*1.5*	37	*7.4*	9	*0.72*	2	*0.04*
	May	3	*1.2*	1	*0.3*	20	*4*	5	*0.4*	0	*0*
	Jun	0	*0*	2	*0.6*	25	*5*	14	*1.12*	0	*0*
	Jul	0	*0*	1	*0.3*	5	*1*	4	*0.32*	0	*0*
	Aug	2	*0.8*	7	*2.1*	8	*1.6*	7	*0.56*	3	*0.06*
	Sep	5	*2*	11	*3.3*	18	*3.6*	15	*1.2*	4	*0.08*
	Oct	7	*2.8*	4	*1.2*	22	*4.4*	6	*0.48*	0	*0*
	Nov	15	*6*	8	*2.4*	37	*7.4*	20	*1.6*	3	*0.06*
	Dec	7	*2.8*	5	*1.5*	18	*3.6*	12	*0.96*	3	*0.06*
2016	Jan	8	*3.2*	9	*2.7*	25	*5*	13	*1.04*	0	*0*
	Feb	5	*2*	3	*0.9*	10	*2*	6	*0.48*	0	*0*
	Mar	5	*2*	9	*2.7*	15	*3*	17	*1.36*	0	*0*
	Apr	7	*2.8*	3	*0.9*	14	*2.8*	7	*0.56*	0	*0*

Considering the month of September for system A, we can see that although the number of SDs of the Minor type are three times higher than the Blocking ones, their impact on the system is lesser than the one of the Blocking type. In fact, the Minor SDs are 15 and their weighted score is 1.2, while the Blocking type number is five, but their calculated weighted score is two (see Table 4.3). Same observation goes for the month of March 2015 where the number for Major SDs is five times higher than the one for Blocking, but the Blocking SDs have more impact on the system than the Major SDs during this month.

Table. 4.4 Monthly weighted scores for system B

Year	Month	B	*W-B*	C	*W-C*	Maj	*W-Maj*	Min	*W-Min*	Inc	*W-Inc*
2015	Jan	3	*1.2*	13	*3.9*	12	*2.4*	18	*1.44*	0	*0*
	Feb	2	*0.8*	4	*1.2*	9	*1.8*	8	*0.64*	0	*0*
	Mar	4	*1.6*	6	*1.8*	14	*2.8*	8	*0.64*	1	*0.02*
	Apr	5	*2*	9	*2.7*	7	*1.4*	21	*1.68*	0	*0*
	May	16	*6.4*	13	*3.9*	15	*3*	16	*1.28*	0	*0*
	Jun	6	*2.4*	32	*9.6*	29	*5.8*	27	*2.16*	0	*0*
	Jul	7	*2.8*	3	*0.9*	16	*3.2*	11	*0.88*	0	*0*
	Aug	4	*1.6*	8	*2.4*	10	*2*	2	*0.16*	0	*0*
	Sep	4	*1.6*	10	*3*	8	*1.6*	9	*0.72*	0	*0*
	Oct	3	*1.2*	2	*0.6*	3	*0.6*	5	*0.4*	0	*0*
	Nov	0	*0*	4	*1.2*	5	*1*	10	*0.8*	0	*0*
	Dec	3	*1.2*	8	*2.4*	7	*1.4*	10	*0.8*	0	*0*
2016	Jan	3	*1.2*	7	*2.1*	7	*1.4*	14	*1.12*	0	*0*
	Feb	4	*1.6*	3	*0.9*	13	*2.6*	26	*2.08*	0	*0*
	Mar	4	*1.6*	2	*0.6*	2	*0.4*	9	*0.72*	0	*0*
	Apr	7	*2.8*	6	*1.8*	7	*1.4*	16	*1.28*	1	*0.02*

For system B, we have similar observations as for system A. In fact, for the month of April 2015, the number of Minor SDs type is 21, four times higher than the ones of the Blocking type (5) but the impact of these Blocking SDs (2 weighted score) on the system is higher than the ones of the Minor ones (1.68) (see Table 4.4.). We have a similar observation for April 2016, where the number of Minor SDs type (11) is almost four time higher than the ones of Critical (3) but the impact of Critical SDs (0.9) on the system is higher than the ones of Minor (0.88).

4.3.2 Discussion

These different results demonstrate that the impact of a SD depends not only on their number but also on the type of severity category under which it falls. Thus, efficient management of SDs required their classification based on a severity scale in order to evaluate their real impact on a given system. This proves that to efficiently manage these SDs, it is necessary not only to mine them, but also to classify them based on their severity. Furthermore, this second classification demonstrates that—in some situations—mining SDs could provide uncompleted or insignificant results for software teams and to system owners. These situations happened when the objectives, the mining techniques, and the analysis were not well defined or the right interpretation was not put in place. To avoid such situations, we propose a conceptual tool to guide practitioners in elaborating SDs strategy and objectives. In fact, we named this conceptual tool *"the SD managerial conceptual tool"* [30]. The SD managerial conceptual tool is a combination of a refinement of the change indicator conceptual tool we presented in the EVOLIS classification, and *"the strategic management model"* proposed by Wheelen and Hunger in the business field [84]. In difference to the strategic management model, our conceptual tool is designed to target the field of SDs mining management. The aim of this conceptual tool is to help software portfolio managers as well as software maintenance teams to define their SDs mining management strategy and to specify concrete actions to put in place with this strategy in mining SDs. We will present this conceptual tool in the next section.

4.3.3 SD Managerial Conceptual Tool

4.3.3.1 The motivation and the role of the SD managerial conceptual tool

The SDs mining falls under the software evolution and maintenance phase. In fact, mining SDs is a complex set of activities; it goes from selecting a technique to mine the SDs, interpreting the obtained results, to taking a decision based on the obtained results. Moreover, each software system is unique, thus needs a specific SDs mining management strategy, e.g., the SDs of the system Waterfox [85] are not the same for Firefox [86], even though they have similar functionalities and purpose. Due to this complexity, inefficient SDs mining can lead to situations such as:

1. the results obtained from the SDs' mining are inaccurate for the SDs team as well as for the maintenance team and consequently, irrelevant for the SD product owner;
2. the SDs mining is requiring much more resources than planned and software portfolio manager, the maintenance team, and the mining team failed to take appropriate decisions in order to improve the quality of the software system based on the mining results;
3. the mining goals are poorly aligned with the strategy and the objectives of SDs management and the product owner's business needs, and;
4. control and evaluation measures for obtained results are missing. To avoid these problems, we are proposing this conceptual tool to guide SDs miners wishing to improve their mining project.

In order to provide a solution that not only addresses these types of situations, but also guides software maintenance teams to manage accurately the mining of their software system, we proposed this conceptual tool.

With this tool, we empower the software portfolio manager to manage efficiently and effectively the mining of the SDs of their systems. It is used when the system is in its maintenance phase, and only if there is a gathering of SDs or CRs on the system overtime. It also empowers any researcher or practitioner to conduct the mining of the SDs of any E-type system, which reach a maintenance phase. This tool has two main levels: the strategy level and the operational level.

At the strategy level, the application of this tool suggests to both the software portfolio managers and the SDs mining team leader to adopt a SDs mining strategy that is aligned with the objectives set by the software owners concerning the quality of their software system (stage 1).

At the operational level, this tool suggests to the SDs mining team to break down the selected mining strategy into concrete SDs mining goals (stage 2). At the same level, it also recommends that, for each objective or goal, a number of actions or activities need to be performed in order to reach the set objectives (stage 3). Finally, this tool recommends the implementation of corresponding control measures in order to evaluate the success of their actions (stage 4). We present this tool in form of a procedure to follow. In the next section, we present the four stages of this conceptual tool with a concrete example for each stage.

4.3.3.2 Presentation of the SD managerial conceptual tool

The conceptual tool is defined in four stages [30]:

1. The first stage consists of defining the SDs mining management strategy in alignment with the needs of the software product owner. The strategy must be broken down into short- or medium-term goals to achieve. The software team as well as the product owner must approve these goals, e.g., a defect mining management strategy may improve the software quality with the development of programs with few SDs for each software version released. The approval of these goals will lead to the second stage.
2. The second stage, which happens on the operational level, consists of converting this strategy into concrete objectives. Referring to the previous example, the set of objectives will be to improve the detection of the defect modules and predict SDs.
3. Following this, each objective must be broken down into terms of specific actions to be performed, e.g., classifying SDs according to their priorities. In addition, members of the SDs mining team are responsible for implementing each of these actions.

4. Following this and depending on the actions put in place, software teams must carefully select control measures to evaluate the state of the actions, e.g., the ratio of the corrected high level prioritized SDs over the total number of SDs received.

Finally, the software team must define a list of actions to establish in order to correct cases where the set objectives have not been reached, e.g. reorganization the process to detect SDs. Fig. 4.6 presents the process to follow to implement the proposed conceptual tool. In addition, we illustrate the application of this concept tool on system B's data.

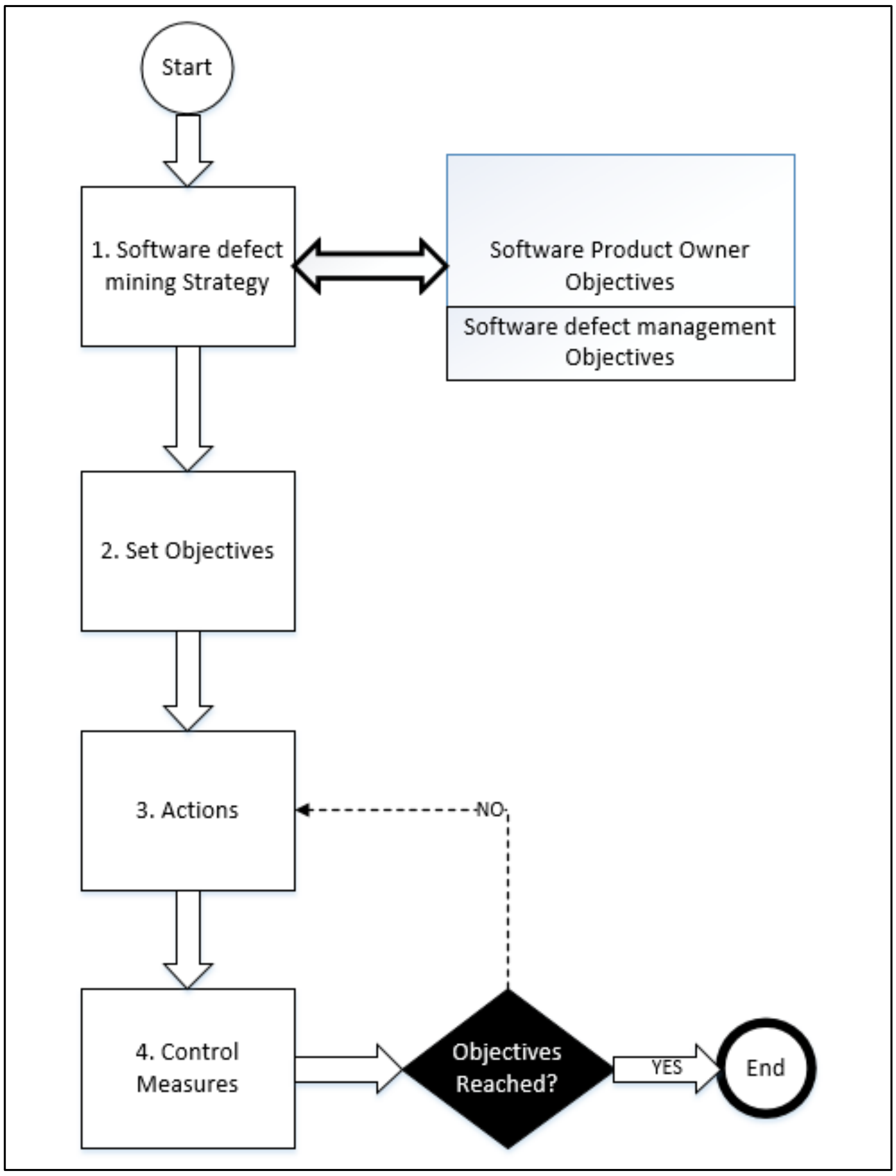

Start
1. Software defect mining Strategy
Software Product Owner Objectives
Software defect management Objectives
2. Set Objectives
3. Actions
NO
4. Control Measures
Objectives Reached?
YES
End

Fig. 4.6 Software defect managerial conceptual tool [30]

In order to apply the proposed conceptual tool to improve and control the SDs mining management in practice, we decided to conduct a proof of concept of our software system B in the next section.

4.3.4 Applying the Managerial Conceptual Tool on System B

- **Stage 1 and 2: Strategy definition and set of objectives**

In alignment with the owner's objective, our strategy was to mine SDs in order to reduce the impact of SDs on the system to limit the system's unavailability time (stage 1). In the next step (stage 2), we cascaded the defined strategy into different objectives such as reducing the impact of defects on system B, and possibly improving the correcting process of the SDs. In the next step, we defined a set of actions to implement the objective of reducing the defects' impact on the system (stage 3).

- **Stage 3: The Classification of SDs of system B based on their severity**

As a concrete action to reduce the effect of SDs on this system, we decided to evaluate the severity impact by classifying them based on the IEEE 1044-2009 [19] severity attribute. We present the results of this classification as follows:

- Results of System B Severity classification

Severity	**Blocking**	**Critical**	**Major**	**Minor**	**Inconsequential**	**Total**
2015	***57***	***112***	***135***	***145***	***1***	***450***
Jan	3	13	12	18	0	46
Feb	2	4	9	8	0	23
Mar	4	6	14	8	1	33
Apr	5	9	7	21	0	42
May	16	13	15	16	0	60
Jun	6	32	29	27	0	94
Jul	7	3	16	11	0	37
Aug	4	8	10	2	0	24
Sep	4	10	8	9	0	31
Oct	3	2	3	5	0	13
Nov	0	4	5	10	0	19
Dec	3	8	7	10	0	28
2016	***18***	***18***	***29***	***65***	***1***	***131***
Jan	3	7	7	14	0	31
Feb	4	3	13	26	0	46
Mar	4	2	2	9	0	17
Apr	7	6	7	16	1	37
Total	***75***	***130***	***164***	***210***	***2***	***581***

- **Stage 4: The selection of control measures**

This stage is similar to stage 4 of our previous change indicators conceptual tool. There are two important aspects to consider when selecting the evaluation metrics at the fourth stage of this conceptual tool. The first one is to choose metrics based on the objective or action to evaluate, e.g., a ratio of the corrected SDs over the total number of SDs received to evaluate the SDs' correction process. The second one is to take into consideration the Critical level [87] of the system being managed. This Critical level can relate to its business, security, and safety aspect. In addition, to determine the Critical level of the system, software teams must consult and get the approval of the product owner.

To track and evaluate the success of our objective, we selected a metric as an indicator (stage 4). In this regard, we defined the SDs indicator as a ratio of the weighted value of a type of SDs over the total weighted value of this type calculated for a month. This ratio informs us about the type of defects that is problematic during the month. We define a problematic case as follows: when the weighted value of a certain type of SDs is higher or equal to one-third of the total SDs weighted value in a month. One-third of the total weighted SDs is an agreed upon limited number a type of SDs may have during a month. The selection of this metric was based on system B's critical mission, which is its availability during the exam periods. We defined the indicator as follows [30]:

$$n \text{ expresses the severity type of SDs to evaluate}$$

$$X, \text{ the weighted value of SDs and } t \text{ being a time period}$$

$$If\ X_n t \geq \frac{\sum(X)t}{3}$$

$$then\ investigate$$

Following this, we defined a list of possible actions to undertake in order to correct problematic cases:

- Investigate within the problematic type of SDs to identify poor uncorrected defects;
- Reorganize the process of correcting the problematic type of SDs;
- Check other indicators such as the number of correcting defects over the total SDs within this category.

We must clarify that the choice of action depends on the investigation results. In this regard, an investigation must be conducted when an indicator is reached, in order to identify the problem and to provide the right fix on time. In Fig. 4.7, we present the result of applying our proposed indicator on SDs of system B.

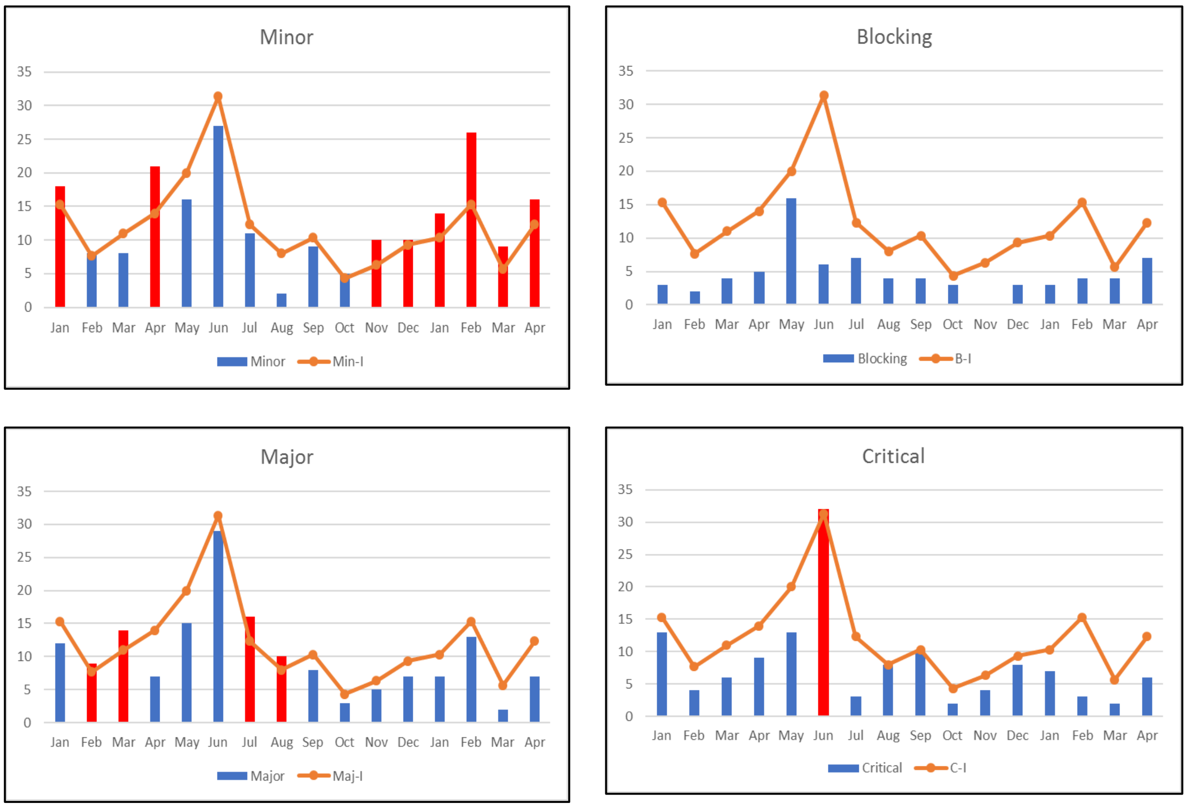


Fig. 4.7 Application of the software indicator on the system B's SDs classification based on their severity

These results show us that the Blocking type of SDs counted none of their months as being problematic, while the Major type has four months being problematic and the Minor type has eight. This implies that although the number of Major type SDs is the highest, it also has been constant and not causing that many problems in managing SDs; meanwhile, although the number of Minor type SDs is small, it has caused more problems than the other types. Going further, we recommend software-mining teams to investigate with the aim of identifying the reasons for these problematic cases.

As answer to the second sub-question, we found that the impact of SDs on an E-type system must be evaluated based on a well-defined severity scale. Thus, we found that system A was mostly affected by the Major type of SDs (49%) followed by the Blocking type (22%), then the Critical (19%) and finally the Minor type (10%) (see Appendix 5.6).

For system B, we found that it was mostly impacted by the Critical type of SDs (33%) followed by the Major type (28%), then the Blocking (25%), and finally the Minor type (14%) (see Appendix 6.7). In the next section, we will analyze the results of our third classification to identify which trigger factors generate the most severe SDs impact on our systems.

4.4 Third Classification: EVOLIS & Severity Classifications (Step 3)

As mentioned in chapter 3, the results of this classification are presented in our third paper published in *Digital Science 2018, Advances in Intelligent Systems and Computing, vol 850* [31], and in our fourth paper published in *ICTS19 Advances in Intelligent Systems and Computing, vol 918* [32] (see Appendix 3 and 4).

After having provided an answer to the two sub-questions in the previous sections, we will now answer the main question of our research, which is: "Which SD trigger factors cause the most severe SDs?", by integrating both of the answers obtained above.

4.4.1 Definition of Severe Software Defect

Based on their definition, we separated the severity level into two groups: the first group we called "severe SDs" and the second group we named "no severe SDs". The severe SDs are any SD which impacts the system in a way that prevents it from being operational, and may cause financial loss or any considerable resource loss to the system owner as well as system users. They are Blocking, Critical, and Major severity types.

The "no severe SDs" are any SD which has an impact level that does not affect the system operations: they are of Minor and Inconsequential type. Thus, we only consider the first group of severity level to be authentic severe SDs.

For the rest of our analysis, we will only consider the severe SDs group. Thus, even though we will present the results of the Minor as well as the Inconsequential type, they are not part of our severe SDs.

4.4.2 Discussion

In this section, we combine the results of classification based on the EVOLIS framework, and the second one based on the IEEE Standard 1044-2009 severity [19] attribute for each system studied. The combination of both classifications shows that:

4.4.2.1 System A

For system A, the Technology factors are respectively responsible for 11%, 18%, 55%, and 16% of Blocking SDs, Critical SDs, Major SDs, and Minor SDs (see Fig. 4.8). Similarly, the IS architecture triggers represent, respectively 13%, 8%, 69%, and 10% of Blocking SDs, Critical SDs, Major SDs, and Minor SDs (see Fig. 4.8). The Business/IS alignment factors trigger respectively 17%, 15%, 47%, 20%, and 1% of Blocking SDs, Critical SDs, and Major SDs, Minors, and Inconsequential SDs. Finally, the IS/user fit triggers represent, respectively, 7%, 10%, 32%, 45%, and 6% of Blocking SDs, Critical SDs, Major SDs, Minor SDs, and Inconsequential SDs (see Fig. 4.8).

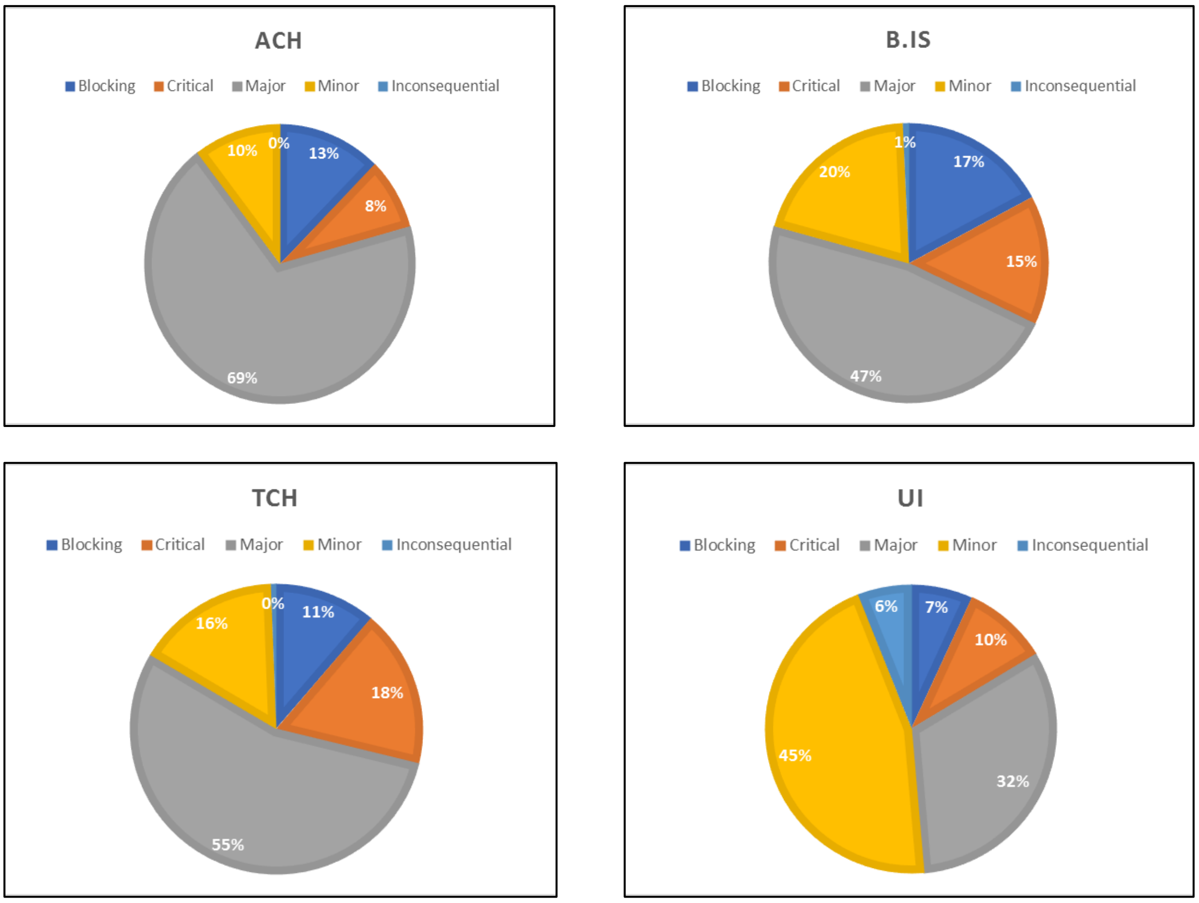


Fig. 4.8 Trigger factors and severity of system A's SDs

As we mentioned earlier in the severity classification section, limiting the severity SDs classification to only their number does not provide the right insight for understanding and interpreting the data. Therefore, we introduced our proposed weighting model into this third classification. We then calculated the weighted score (W) for each trigger factor group based on the severity weight. We then proceed to analyze only the data of severe SDs (see Table 4.5).

Table. 4.5 Severe SDs triggers of system A

	Weight	**ACH**	*W-ACH*	**TCH**	*W-TCH*	**UI**	*W-UI*	**B.IS**	*W-B.IS*
Blocking	*0.4*	18	*7.2*	19	*7.6*	15	*6*	24	*9.6*
Critical	*0.3*	12	*3.6*	30	*9*	21	*6.3*	21	*6.3*
Major	*0.2*	100	*20*	93	*18.6*	71	*14.2*	66	*13.2*
Total	*0.9*	130	*30.8*	142	*35.2*	107	*26.5*	111	*29.1*

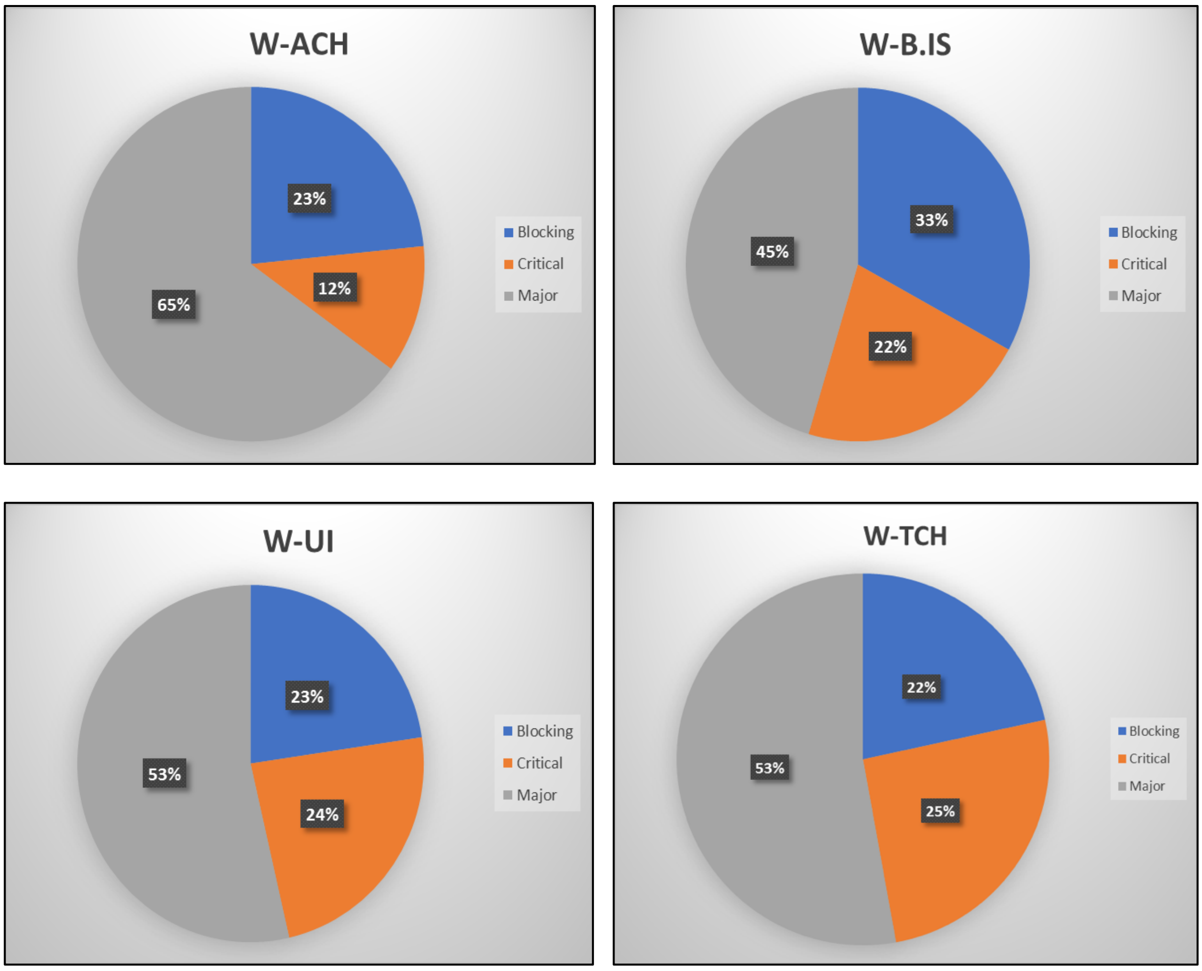


Fig. 4.9 Severe SDs Triggers of System A

Introducing the weighted model shows that for system A, the Technology trigger factors cause in the majority 53% of Major SDs followed by 25% of Critical SDs, and finally 22% of Blocking SDs (see Fig. 4.9). Similarly, the IS/user fit factors cause 53% of Major SDs, followed by 24% of Critical SDs, and finally 23% of Blocking SDs (see Fig. 4.9). The third group of trigger factors is the IS architecture trigger factors group. The IS architecture factors cause in majority 65% of Major SDs followed by 23% of Blocking SDs, and finally 12% of Critical SDs (see Fig. 4.9). Finally, the Business/IS alignment trigger factors cause 45% of the Major SDs, 33% Blocking SDs, and 22% of Critical SDs (see Fig. 4.9).

In overall, the results show for system A that the Technology trigger factors with the highest weighted score of 35.2 are responsible for most of the severe SDs followed by the IS architecture factors, with a weighted score of 30.8. Then the Business/IS alignment, with 29.1 and finally the IS/user fit trigger factors, with a weighted score of 26.5 (see Fig. 4.10).

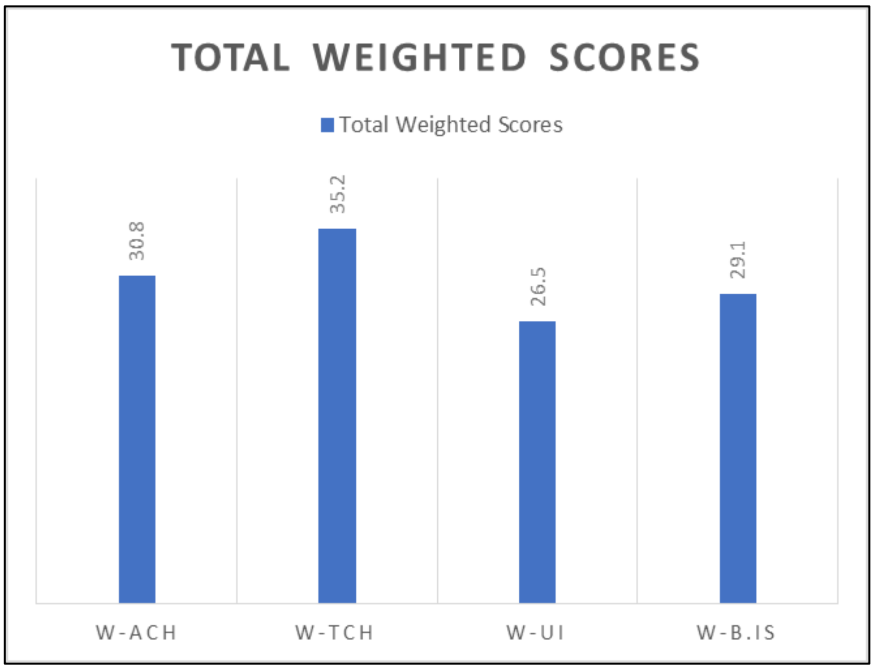


Fig. 4.10 System A's total weighted score for Severity and EVOLIS classifications

4.4.2.2 System B

We performed the same analysis we did on system A on system B. The combination of the EVOLIS framework and the severity results for system B show that the Technology triggers represent 16% of Blocking SDs, 23% Critical SDs, 28% of Major SDs, and 33% of Minor SDs (see Fig. 4.11). The Inconsequential SDs do not have any Technology factor as a trigger. The second group of trigger factors, the IS architecture triggers, represent, respectively 15%, 28%, 32%, and 25% for Blocking SDs, Critical SDs, Major SDs, and Minor SDs (see Fig. 4.11). The Business/IS alignment factor triggers shows the first place being at 50% of Minor SDs, in second place 27% with Major SDs followed by 17% of Critical SDs, and 6% of Blocking SDs (see Fig. 4.11). There are no Inconsequential SDs in this group. More than the half of the SDs triggers by IS/user fit factors are of the Minor type (51%), then of Major type (25%), Critical type (17%), Blocking (5%), and finally of Inconsequential type (2%) (see Fig. 4.11).

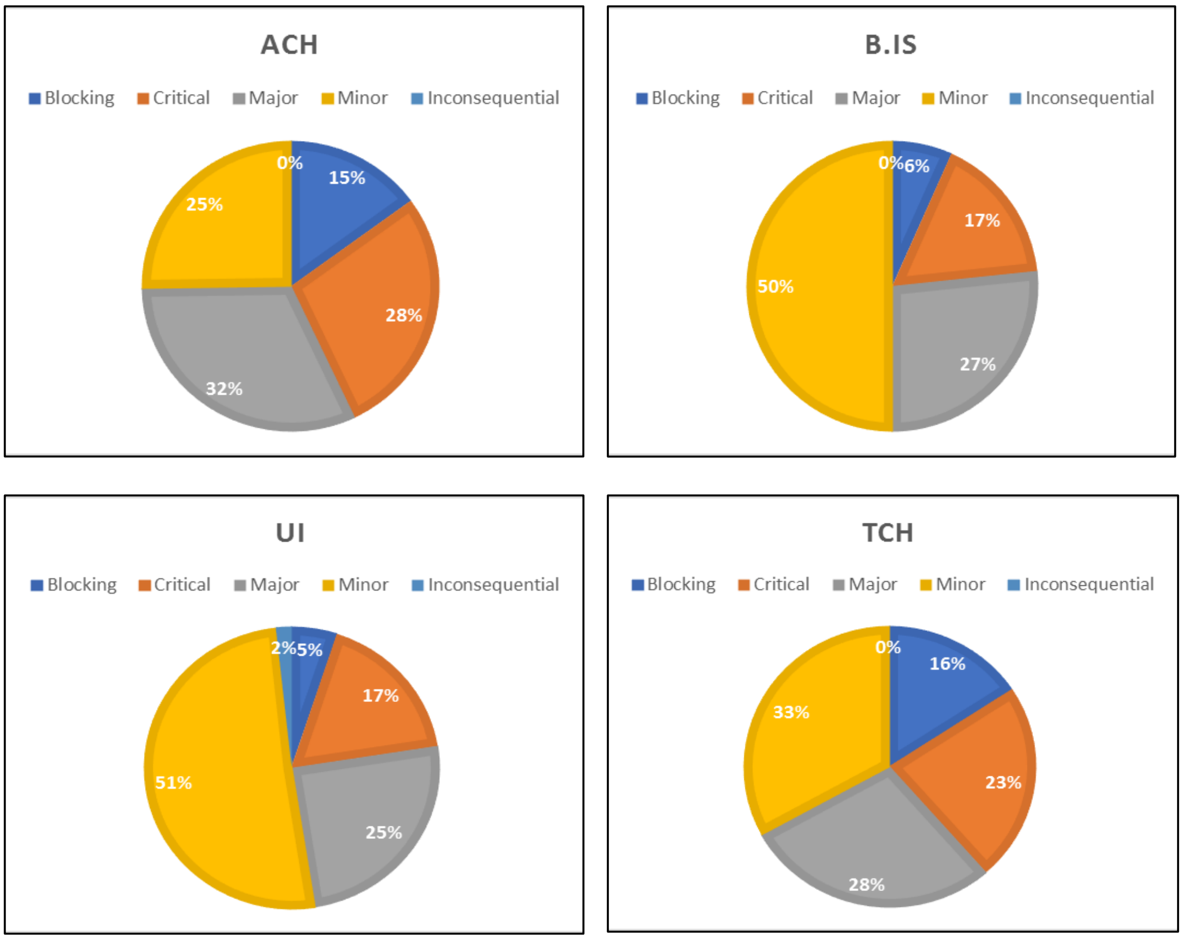


Fig. 4.11 Trigger factors and severity of system B's SDs

Similarly, as in the case of system A, we also calculated the weighted score (W) for each trigger factor group based on the severity weight for system B. We then proceeded to analyze only the data of severe SDs of this system (see Table 4.6).

Table. 4.6 Severe SDs triggers of system B

	Weight	ACH	*W-ACH*	TCH	*W-TCH*	UI	*W-UI*	B.IS	*W-B.IS*
Blocking	0.4	16	***6.4***	51	***20.4***	6	***2.4***	2	***0.8***
Critical	0.3	30	***9***	74	***22.2***	21	***6.3***	5	***1.5***
Major	0.2	34	***6.8***	92	***18.4***	30	***6***	8	***1.6***
Total	0.9	80	***22.2***	217	***61***	57	***14.7***	15	***3.9***

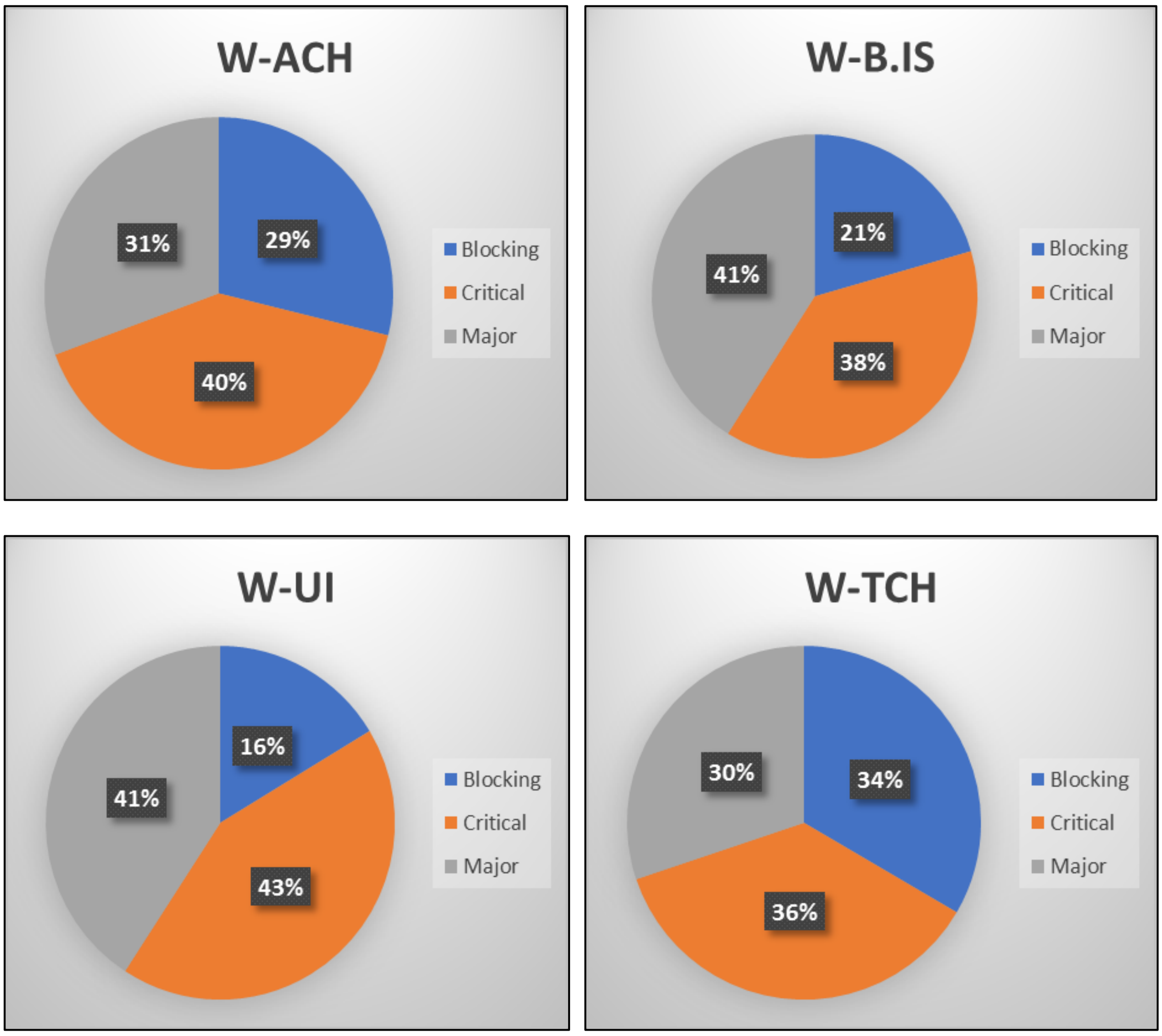


Fig. 4.12 Severe SDs triggers of system B

Similarly, looking at system A's results, we can see that 40% of the severe SDs caused by the IS architecture trigger factors are of Critical type, followed by 31% that are of Major type, and 29% that are of Blocking type (see Fig. 4.12). There is only a slight difference between the Major SDs group and the Blocking SDs group (see Fig. 4.12). The Business/IS alignment trigger factors cause 41% of Major SDs, 38% Critical, and 21% of Blocking SDs. The

Technology trigger factors cause in majority 36% of Critical SDs followed by 34% of Blocking SDs, and finally 30% of Major SDs (see Fig. 4.12). Finally, the IS/user fit factors cause 43% of Critical SDs, followed by 41% of Major SDs, and finally 16% of Blocking SDs (see Fig. 4.12).

In overall, the results show for system B that the Technology trigger factors with the highest weighted score of 61 are responsible for most of the severe SDs, followed by the IS architecture trigger factors, with a weighted score of 22.2. Contrary to system A, the third position is held by the IS/user fit triggers with a weighted score of 14.7 and finally the Business/IS alignment coming in last with a weighted score of 3.9 (see Fig. 4.13).

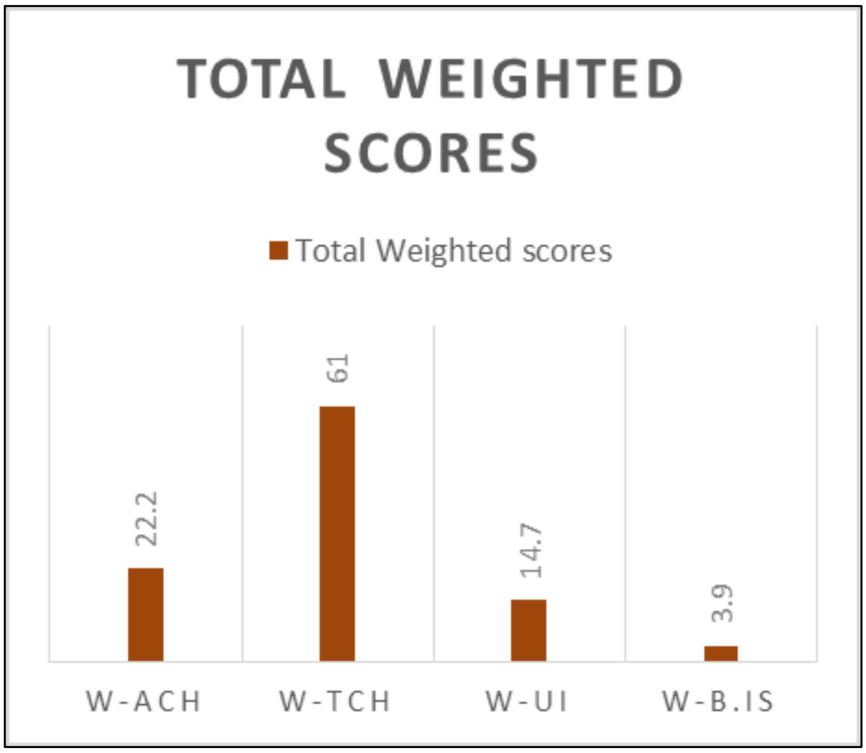


Fig. 4.13 System B total weighted score for Severity and EVOLIS classifications

In summary, as answer to our main research question, *“Which types of trigger factors generate the most severe SDs on a given E-type software system?”,* we found that:

Ranking first is the Technology trigger factor group with a weighted score of 35.2 for system A and a weighted score of 61 for system B. It is followed by the IS architecture trigger factors group with a weighted score of 30.8 for system A and a weighted score of 22.2 for system B. In the third position is the Business/IS alignment with 29.1 for the system A and IS/user fit with a weighted score of 14.7 for system B. In the last position are the Business/IS alignment with a weighted score of 3.9 for system B, as well as the IS/user fit alignment with a weighted score of 26.5 for system A. With these results, we can conclude that in our case the Technology and the IS architecture trigger factors are the leading couple in causing severe SDs. We emphasize on the fact that we consider as a severe SD any SD having a Blocking, Critical, or Major severe impact on a system.

4.4.3 The Origins of Severe Software Defects Method

Based on the results of this third classification, we propose a method in order to identify the origins of severe SDs on any E-type system. This method is addressed to people responsible for managing and controlling software evolution such as software portfolio manager, software maintenance teams, software development teams, and to researchers studying software evolution and conducting studies in the SD/CR mining field. This method empowers these stakeholders to identify the origins or sources of severe SDs on any E-type system. We named this method "The origins of severe software defects method".

This method consists of three stages: SDs collection or acquisition stage, SDs analysis stage, and the SDs classification stage. We define each of these stages as follows:

- 1st stage: The collection stage consists of collecting SDs into a software repository.
- 2nd stage: The analysis stage consists of identifying the trigger factors (origins) of the SDs and evaluating their severity impact on the system. We named a SD having its origin and severity impact identified "Analyzed SD" (ASD).
- 3rd stage: The classification stage consists of grouping the analyzed SDs (ASDs) at the precedent level into two groups: the group of severe SDs and the group of nonsevere SDs. Any ASD that causes a partial or total disruption on the system's essential operations falls under the second group. The severity seriousness of the SDs is defined by the person responsible of software engineering in this context. Any other ASD having no impact on the system's essential operations must be categorized into the group of nonsevere SDs. Performing this classification will lead to identify the origins of severe SDs of the E-type system being studied.

We represent this method in pyramid form. The first stage being the initial phase, we place it at the base of the pyramid. It is followed by the second stage in the middle of the pyramid and then the third stage at the top of the pyramid (see Fig 4.14). In summary, the activities at the three stages are presented as follows:

At the first stage (the collection stage), the activity to perform is:

- The collection of SDs.

At the second stage (the analysis stage), the activities to perform are:

- For each SD, identifying its trigger factor or origin, and identifying the severity of its impact on the system.

At the third stage (the classification stage) the activity to perform is:

- Classification of the ASDs based on their severity impact into severe and nonsevere groups.

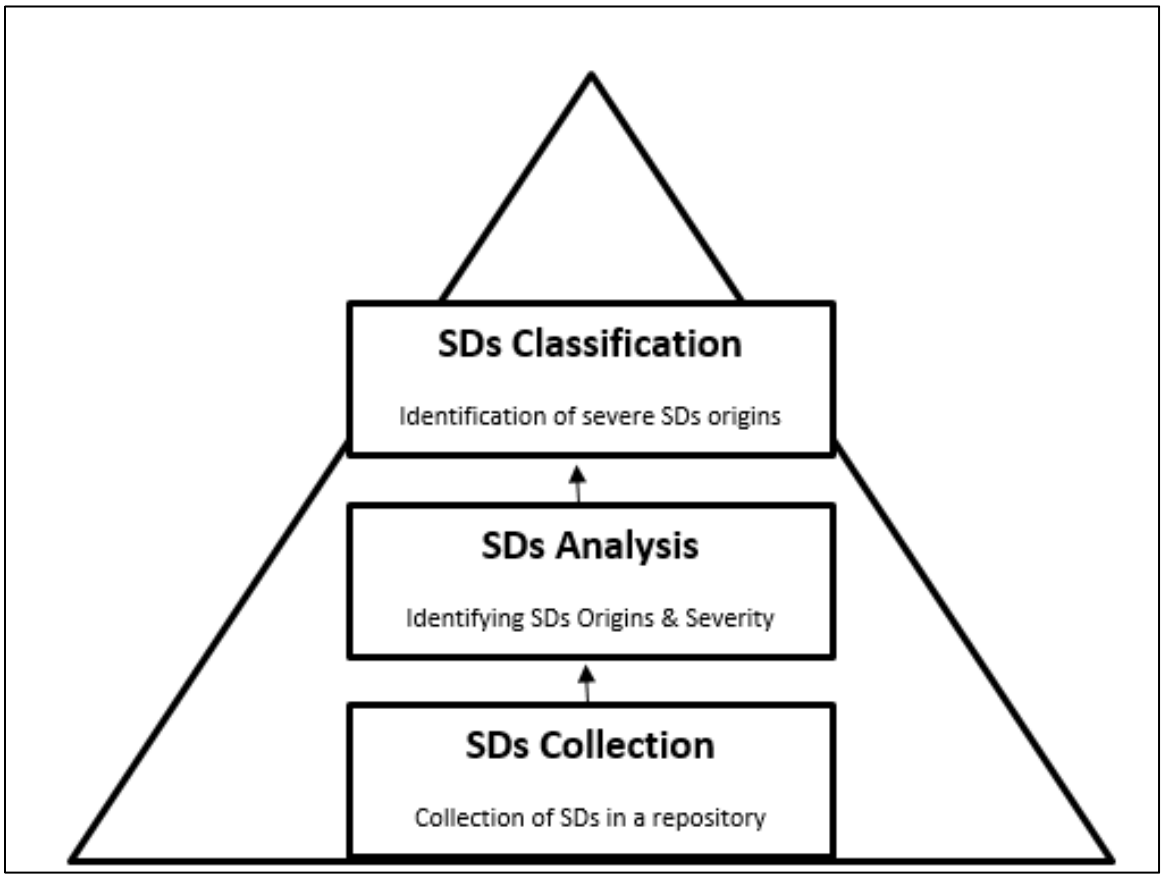

SDs Classification
Identification of severe SDs origins
SDs Analysis
Identifying SDs Origins & Severity
SDs Collection
Collection of SDs in a repository

Fig. 4.14 The origins of severe software defects method.

To answer the question *"How to identify the origins of severe defects on evolving information systems?",* we provide *"the origins of severe software defects method"*. Applying this method will help researchers as well as practitioners in software management and controlling fields to identify the trigger factors of severe SDs of any E-type system.

5 Conclusion

Not only do SDs cause huge financial loss to system owners, but their management also causes up to 80% of the total cost of a system during its life cycle [23]. In this context, identifying the ones which may potentially generate high financial damage was the goal of our research. To reach this goal, we conducted two case studies on two systems in the field of education. We studied the SDs of these systems over several months. We classified them based on two SD classification concepts: the EVOLIS [9] and their severity [19]. Each of these classifications was done to answer the two sub-questions necessary to be able to answer our main question. In the next part, we will present the answer we have for each question, as well as our contributions to and the limitations of this research.

5.1 Contributions

- **The main practical and theoretical contributions of this research**

This thesis provides practical and theoretical contributions on identifying severe SD trigger factors of a given E-type system. The contributions of this research are represented in the following three points:

1. In answering the two sub-questions, we proposed two conceptual tools in order to improve the management of SDs.
2. In answering our main research question, we proposed a method in order to identify the origins of severe SDs on an E-type system. We named it “the origins of severe software defects method” (see Fig. 5.1).
3. We identified the existing relationship between SD trigger factors and the impact of SDs they may cause to a given E-type system in terms of severity.

- **The role of time in these contributions**

These previous contributions could not be possible if we had not considered the notion of time. Time is a central aspect of this research. In fact, in order to study the evolution of these E-type systems and to propose tools to manage and control this evolution efficiently, we studied the dynamism of these two systems. These studies cover a long period from January 2014 to December 2016 in our four published papers. We analyzed in detail over a thousand SDs resulting in change requests. These change requests are the main drivers of the system change, thus leading to a system evolution over time. E.g., considering our first classification, we observed the monthly evolution of SDs. This observation led us to propose a conceptual tool that will help people concerned with the evolution of E-type systems such as software portfolio managers and software maintenance teams in controlling the SDs evolution of their E-type systems.

Furthermore, this specific (monthly) period analysis allowed us to observe the impact some events in the studied system's environment have on the evolution of the system. E.g. reopening of schools trigger an unusually high amount of SDs of IS user-fit type for system B.

5.1.1 Conceptual Tools

- **Software change indicator conceptual tool [29]**

We presented a conceptual tool as a summary of the steps we followed in this first study. We did this in order to provide a tool to software teams and researchers to be able to conduct similar studies on different systems. In addition, this conceptual tool provides guidelines on how to identify a sudden rise of SDs of a particular category. Furthermore, it provides recommendations on how to get prepared in order to handle a problematic rise of SDs of E-type systems. We named this tool "software change indicator conceptual tool" [29].

Furthermore, to the first sub-question—"which SD factors trigger most of SDs on our studied systems?"—we find that for system A, the IS/user fit trigger factors block is in the lead with 33% of the total SDs, followed by the Technology block with 25%, and finally the Business/IS alignment and the IS architecture with 21% each (see Appendix 5.1).

For system B, the Technology block is leading with 56% of the total SDs, followed by the IS/user fit 21%, then the IS architecture 18%, and finally the Business/IS alignment with 5% (see Appendix 6.1).

- **Software defect managerial conceptual tool [30]**

At this point, we presented a refined version of our precedent conceptual tool. This new version incorporated the alignment between the strategy and objectives of software mining teams with the strategy and objectives of software owners. This tool is to help both parties in engaging in mining projects that are most useful for them. We named it "the software defect managerial concept tool" [30].

We also provided an answer to our second sub-question while conducting this study. For the second sub-question, which was to evaluate the impact that a SD has on a given E-type system, we found that:

For system A, the Major SDs group is in the lead position with a 66-weighted score, followed by the Blocking SDs group with a 30.4-weighted score, and then the Critical group with 25.2, the Minor with 13.6, and finally the Inconsequential with a 0.3-weighted score (see Table 4.2).

For system B, the Critical severe impacts are in the lead with a weighted score of 39 followed by the Major type with 32.8, then the Blocking type with a weighted score of 30, the Minor group with a weighted score of 16.8, and finally the Inconsequential type with a weighted score of 0.04 (see Table 4.2).

- **Integration of results [31]**

Finally, in order to reach our main goal, we integrated both results of the classification by EVOLIS and severity. Doing this, we found that the factors triggering most of severe SDs are of the Technology type followed by the ones of the IS architecture type [31]. The third position is occupied by the factors of Business/IS alignment for system A and by the IS/user fit factors for system B. Furthermore, we also proposed a method in order to identify the sources or origins of severe SDs on any given E-type system. Researchers as well as practitioners may use this method to conduct similar studies on any given E-type system. This method represents a major contribution of this research and we will present it once again in more detail in the next section.

5.1.2 The Method to Identify the Origins of Severe SDs of E-type Systems

To make it possible for other practitioners and other researchers to identify the trigger factors (origins) of severe SDs of E-type systems, we propose "*the origins of severe software defects method"*.

This method consists of three stages: SDs collection stage, SDs analysis stage, and the SDs classification stage (see Fig. 5.1). We define each of these stages as follows:

- The collection stage consists of collecting SDs into a software repository.
- The analysis stage consists of identifying the trigger factors (origins) of the SDs and evaluating their severity impact on the system. We named a SD having its origin and severity impact identified “Analyzed SD” (ASD).
- The classification stage consists of grouping the analyzed SDs (ASDs) at the analysis stage into two groups: the group of severe SDs and the group of nonsevere SDs. Any ASD that causes a partial or total disruption on the system’s essential operations falls under the group of severe SDs. Any other ASD having no impact on the system’s essential operations must be categorized into the group of nonsevere SDs. Performing this classification will lead to identify the origins of severe SDs of the E-type system being studied.

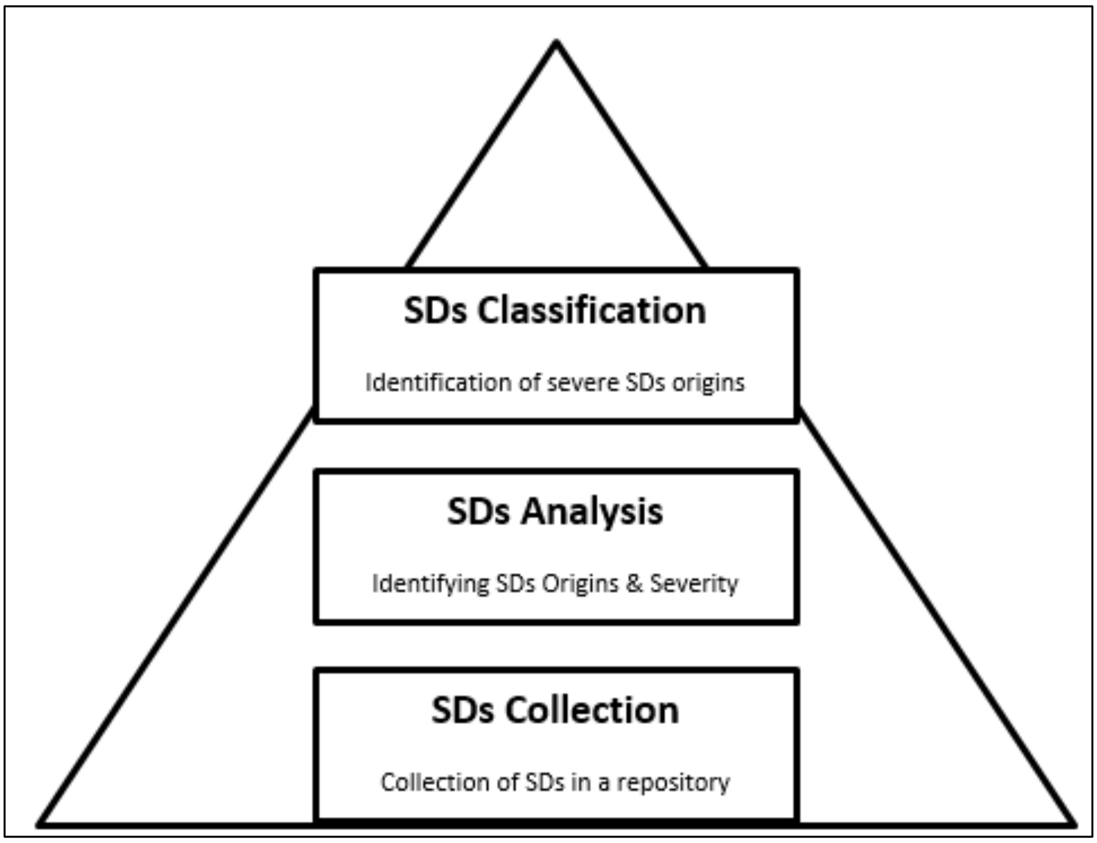

SDs Classification
Identification of severe SDs origins
SDs Analysis
Identifying SDs Origins & Severity
SDs Collection
Collection of SDs in a repository

Fig. 5.1 The origins of severe software defects method.

5.1.3 Relationships between SD Trigger Factors and the Severity Impact they have on E-type systems

To the question of which SD factors trigger the most severe SDs, we found that:

In the leading position come the Technology trigger factors with 29% of the total weighted score for system A and 60 % for system B. They are followed by the IS architecture trigger factors, with 25 % for system A and 22% for system B. We can see that in both cases the same type of SD trigger factors occupies the first and the second position. Opposed to this, the Business/IS alignment occupied third place with 24% for system A, and IS/user fit occupied the same rank with 14% for system B. The last position for system A is occupied by IS/user fit with 22%, and for system B by Business/IS alignment with 4% (see Appendices 5.5 and 6.5).

In addition, our analysis portrays that there are two main couples of SDs trigger factor groups, one being composed of the Business/IS and the IS/user fit alignment, and the leading couple being composed of the Technology and the IS architecture trigger factors.

Concerning the no severe SDs (we defined as no severe SD any SD having a Minor or Inconsequential severity impact on a system), we observed that the majority of the Inconsequential SDs were triggered by the IS/user fit factors. This implies that the probability of an Inconsequential SD being triggered by either the IS architecture, the Business/IS alignment, or the Technology factors is very low or barely existent. In addition, the Minor SDs are in majority triggered by the Technology factors for system B (107) followed by the IS/user fit triggers (61). For system B, the Minor SD types are triggered by the IS/user fit (100), followed by both the Business/IS alignment (28) and the Technology (27) (see Appendix 6.4).

5.2 Limitations and Future Works

5.2.1 Limitations

A common criticism the case study methodology faces in the literature is that its findings are not representative enough to draw a general theory [88]. Similarly, the number of cases used was a bit limited to generate a general theory concerning the factors that trigger severe SDs for E-type systems. Nevertheless, both results show us that the Technology factors in particular have the highest calculated weighted score to be at the source of severe SD. Thus, we argue that a special attention must be paid to the system component presenting vulnerabilities towards the technology trigger factors. In addition, it makes sense to pay particular attention to this type of

SDs because they can spread and affect other systems as well. Our results in fact confirm the bug propagation concept [89], [90] because the technology triggers formed a couple with the architecture triggers: one drives the other one. Furthermore, these findings can serve as provisional truth and can be considered as informative knowledge to other software teams in regards to the management of their E-type systems, more precisely for managing SDs of E-type systems.

In addition, during this research, we only focused on the study of SDs without looking at the data of real activities or actions performed by the software team members when correcting them. A possible perspective would have been to analyze both the SDs and their change response using our method in order to compare their results. This may be addressed in another research project.

5.2.2 Future Works

Further exploration of *the origins of severe software defects method* opens an opportunity to develop an artifact, which can automatically identify a SD's source based on its description. We will present this artifact in the next section.

5.2.2.1 Practical perspective: A tool to manage software defects

From a practical perspective, a possible future work will be to build an automatized SD management system (SDM) based on our proposed method. Just like the method, this SDM system will inform in real time on the real impact of SDs and their sources. This system will help software maintenance teams and software portfolio managers in managing their system's SDs by deploying the right resource at the right moment on the right instance. It also allows them to have an overview of the actual state of their systems. More precisely, to have an insight of their system portfolio, with information such as which systems are requiring more resources for maintenance. Having such knowledge will improve their decision-making in managing a system's life cycle. Thus, it will help them to pilot the evolution of their systems and surely reduce the cost of the system maintenance.

The SDM system will be made up of four main components. Each component represents one of the stages of the origins of severe software defects method.

- **Component 1: SD collection**

The first component is a data store component. Its role is to extract, transform, and load (ETL) SDs data from software repositories and other software bug-reporting platforms (e.g., helpdesk, emails, Jira [56]) into the SDM system. This component corresponds to the SDs collection stage. This component will gather all the necessary information on SDs in one place for analysis, and thus, allowing an easy access to SDs information for the software development and maintenance teams.

- **Component 2: SD trigger factors identification**

The second component will be either a natural language processing (NLP) solution or an artificial intelligence system. This component will have a double role. First, it will analyze each SD description in order to retrieve the semantics of it. Second, based on the semantic of the SD description it will identify the source of this SD. To assume these double roles, this component will integrate notions in the domains of linguistic analysis, machine learning, and ontology engineering. This component corresponds to the second stage of our method. It will be connected to the first component in order to access SDs information. It will perform a classification similar to the first classification we did for this research.

- **Component 3: Evaluation of SD impact**

The third component will evaluate the severity impact of each SD, as we did with our second classification. To perform this evaluation, this component will consider on one side the type and the amount of resources used to solve the SDs. On the other side, it will also consider the financial loss a SD causes the system owner and system users. Based on these two evaluations, this component may classify SDs according to their severity impact. This component also falls under the second stage of the origins of severe SDs method. As well as the second component, this third component will be also connected to the first component in order to have access to SDs' information.

- **Component 4: A dashboard for displaying E-type systems' SDs characteristics**

Finally, a fourth component will integrate the results of the 2nd and 3rd components in order to inform software teams on types of SDs having more impact on their E-type systems. This last component will be a dashboard where software managers can consult the state of each system within their software portfolio. It will present results such as the results we obtained doing our third classification.

The actual state of existing knowledge does not provide enough tools in order to build such system accurately. In fact, the existing semantic analysis tools are limited to a system to draw accurately the semantic knowledge from a sentence [91], [92]. In the future, we hope there will be enough improvement in this area of research in order to apply those novelties to design our proposed SDM system.

5.2.2.2 Research perspective

The results we obtained from analyzing these systems correspond to events that those systems went through during their life cycle. In fact, the system projects' managers confirmed to us that the peak of certain types of SDs at certain periods of time corresponded to special events that happened to the systems during these periods of their life cycle. Based on these feedbacks, we argue that applying our origins of severe SDs method to analyze more E-type systems over their entire life will help to identify the evolving characteristics of E-type systems. These characteristics can be classified into groups to represent different stages of evolution of E-type systems. Thus, it will present as an artifact and its application will help both practitioners and researchers to determine the life stages of E-type systems.

Finally, another future possibility will be to conduct more case studies on other E-type systems of different domains. This not only in order to confirm our results, but also to compare these results among themselves and possibly generate a theory from these future results as well as our results.

6 References

[1] C. Barker, “The top 10 IT disasters of all time,” *ZDNet*. [Online]. Available: https://www.zdnet.com/article/the-top-10-it-disasters-of-all-time/. [Accessed: 27-Sep-2018].

[2] “77 Million Edmodo Users Are Hacked as Widespread Cyberattacks Hit the Ed Tech World.” .

[3] “Knight Shows How to Lose $440 Million in 30 Minutes,” *Bloomberg.com*, 02-Aug-2012.

[4] S. Neville, “NHS cyber attack far more extensive than thought, says report,” *Financial Times*, 26-Oct-2017. [Online]. Available: https://www.ft.com/content/4110069a-ba3d-11e7-8c12-5661783e5589. [Accessed: 01-Oct-2018].

[5] S. Gallagher, “50 million Facebook accounts breached by access-token-harvesting attack,” *Ars Technica*, 28-Sep-2018. [Online]. Available: https://arstechnica.com/information-technology/2018/09/50-million-facebook-accounts-breached-by-an-access-token-harvesting-attack/. [Accessed: 01-Oct-2018].

[6] M. M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[7] S. Cook, R. Harrison, M. M. Lehman, and P. Wernick, “Evolution in software systems: foundations of the SPE classification scheme,” *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 1, pp. 1–35, Jan. 2006.

[8] G. S. Walia and J. C. Carver, “A systematic literature review to identify and classify software requirement errors,” *Information and Software Technology*, vol. 51, no. 7, pp. 1087–1109, Jul. 2009.

[9] A. Métrailler and T. Estier, “EVOLIS Framework: A Method to Study Information Systems Evolution Records,” in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, 2014, pp. 3798–3807.

[10] I. Benbasat, D. K. Goldstein, and M. Mead, “The Case Research Strategy in Studies of Information Systems,” *MIS Quarterly*, vol. 11, no. 3, p. 369, Sep. 1987.

[11] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?,” in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

[12] M. N. Li, Y. K. Malaiya, and J. Denton, “Estimating the number of defects: A simple and intuitive approach,” in *Proc. 7th Int’l Symposium on Software Reliability Engineering (ISSRE)*, 1998, pp. 307–315.

[13] S. Wagner, “Defect classification and defect types revisited,” in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 39–40.

[14] Z. Li and Y. Zhou, “PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code,” p. 10.

[15] T. Xie, J. Pei, and A. E. Hassan, “Mining Software Engineering Data,” in *29th International Conference on Software Engineering (ICSE’07 Companion)*, Minneapolis, MN, 2007, pp. 172–173.

[16] R. Binder, *Testing object-oriented systems: models, patterns, and tools*. Reading, Mass: Addison-Wesley, 2000.

[17] D. Vallespir, F. Grazioli, and J. Herbert, “A framework to evaluate defect taxonomies,” in *XV Congreso Argentino de Ciencias de La Computación*, 2009.

[18] M. Leszak, P. Dewayne E., and D. Stoll, “Classification and evaluation of defects in a project retrospective,” *The Journal of Systems and Software*, no. 61, pp. 173–187, 2002.

[19] *1044-2009 IEEE Standard Classification for Software Anomalies.* 2009.

[20] N. Mellegãrd, *Improving Defect Management in Automotive Software Development, LiDeC—A Light-weight Defect Classification Scheme*. Chalmers University of Technology, 2013.

[21] Beizer, Boris, *Software testing techniques*, 2nd Edition. Thomson Learning, 1990.

[22] R. G. Mays, C. L. Jones, G. J. Holloway, D. P., and D.P. Studinski, “Experiences with Defect Prevention.” IBM Systems Journal, 29(1):4, 32, 1990.

[23] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The impact of using regression models to build defect classifiers,” in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 135–145.

[24] R. Chillarege *et al.*, “Orthogonal defect classification-a concept for in-process measurements,” *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[25] J. T. Huber, “A Comparison of IBM’s Orthogonal Defect Classification to Hewlett Packard’s Defect Origins, Types, and Modes.” Hewlett Packard Company, 1999.

[26] L. Yu and S. R. Schach, “Applying association mining to change propagation,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 18, no. 08, pp. 1043–1061, 2008.

[27] G. Murphy and D. Cubranic, “Automatic bug triage using text categorization,” in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

[28] W. Dickinson, D. Leon, and A. Fodgurski, “Finding failures by cluster analysis of execution profiles,” in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, Toronto, Ont., Canada, 2001, pp. 339–348.

[29] N. Hillah and T. Estier, “The Application of Change Indicators in Mining Software Repositories,” in *Trends and Advances in Information Systems and Technologies*, vol. 746, Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo, Eds. Cham: Springer International Publishing, 2018, pp. 418–428.

[30] N. Hillah, “A Conceptual Tool to Improve the Management of Software Defects,” in *Business Modeling and Software Design*, vol. 319, B. Shishkov, Ed. Cham: Springer International Publishing, 2018, pp. 443–451.

[31] N. Hillah, “Severe Software Defects Trigger Factors: A Case Study of a School Management System,” in *Digital Science*, vol. 850, T. Antipova and A. Rocha, Eds. Cham: Springer International Publishing, 2019, pp. 389–396.

[32] N. Hillah, “Classification of Software Defects Triggers: A Case Study of School Resource Management System,” in *Information Technology and Systems: Proceedings of ICITS 2019*, vol. 918, Á. Rocha et al., Ed. Cham: Springer International Publishing, 2019, p. 10.

[33] “evolution | Definition of evolution in English by Oxford Dictionaries,” *Oxford Dictionaries | English*. [Online]. Available: https://en.oxforddictionaries.com/definition/evolution. [Accessed: 03-Oct-2018].

[34] G. E. Moore, “Cramming More Components Onto Integrated Circuits,” *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan. 1998.

[35] M. Doyle, “The Project Gutenberg EBook of On the Origin of Species, by Charles Darwin,” p. 362.

[36] Joseph Valacich and Christoph Schneider, *Information Systems Today: Managing in the Digital World*, 4th Edition. Pearson, 2010.

[37] D. P. Truex, R. Baskerville, and H. Klein, “Growing systems in emergent organizations,” *Communications of the ACM*, vol. 42, no. 8, pp. 117–123, Aug. 1999.

[38] M. M. Lehman and J. C. Fernáandez-Ramil, “Rules and Tools for Software Evolution Planning and Management,” in *Software Evolution and Feedback*, N. H. Madhavji, J. C. Fernández-Ramil, and D. E. Perry, Eds. Chichester, UK: John Wiley & Sons, Ltd, 2006, pp. 539–563.

[39] J. H. Saleh, D. E. Hastings, and D. J. Newman, “Flexibility in system design and implications for aerospace systems,” *Acta Astronautica*, vol. 53, no. 12, pp. 927–944, Dec. 2003.

[40] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, “Types of software evolution and software maintenance,” *Journal of software maintenance and evolution: Research and Practice*, vol. 13, no. 1, pp. 3–30, 2001.

[41] R. Phaal, C. J. P. Farrukh, and D. R. Probert, “Technology roadmapping—A planning framework for evolution and revolution,” *Technological Forecasting and Social Change*, vol. 71, no. 1–2, pp. 5–26, Jan. 2004.

[42] “Software and Systems Engineering Vocabulary.” [Online]. Available: https://pascal.computer.org/sev_display/search.action;jsessionid=de3d8aeebd96a8bc6415bfff1750. [Accessed: 04-Oct-2018].

[43] MM Lehman, J.F Ramil, P.D Wernick, D.E Perry, and W.M Turski, “Metrics and Laws of Software Evolution- The Nineties View,” *http://www.ifi.uzh.ch/seal/teaching/courses/archive/hs10-1/evolution/LehmanRamil97-Metrics-of-swevol.pdf*, 1997. [Online]. Available: http://www.ifi.uzh.ch/seal/teaching/courses/archive/hs10-1/evolution/LehmanRamil97-Metrics-of-swevol.pdf. [Accessed: 07-Sep-2018].

[44] B. A. Kitchenham *et al.*, “Towards an ontology of software maintenance,” *Journal of Software Maintenance: Research and Practice*, vol. 11, no. 6, pp. 365–389, Nov. 1999.

[45] M. Fowler, K. Beck, and J. Brant, “Refactoring - Improving the Design of Existing Code,” p. 337.

[46] *IEEE Std 1219-1998: IEEE Standard for Software Maintenance*, IEEE. .

[47] E. B. Swanson, “The dimensions of maintenance,” in *Proceedings of the 2nd international conference on Software engineering*, 1976, pp. 492–497.

[48] K. H. Bennett and V. T. Rajlich, “The staged model of the software lifecycle: A new perspective on software evolution,” p. 14.

[49] B. Boehm, “IEEE Transactions on Software Engineering, Vol. SE-10 (1),1984 pp. 4-2,” p. 47.

[50] “anomaly | Definition of anomaly in English by Oxford Dictionaries,” *Oxford Dictionaries | English*. [Online]. Available: https://en.oxforddictionaries.com/definition/anomaly. [Accessed: 30-Jul-2018].

[51] X. Xia, D. Lo, X. Wang, and B. Zhou, “Dual analysis for recommending developers to resolve bugs,” *Journal of Software: Evolution and Process*, vol. 27, no. 3, pp. 195–220, Mar. 2015.

[52] B. Freimut, C. Denger, and M. Ketterer, “An industrial case study of implementing and validating defect classification for process improvement and quality management,” in *Software Metrics, 2005. 11th IEEE International Symposium*, 2005, pp. 10–pp.

[53] B. S. Dhillon and Y. Liu, “Human error in maintenance: a review,” *Journal of Quality in Maintenance Engineering*, vol. 12, no. 1, pp. 21–36, Jan. 2006.

[54] D. Greer and Y. Hamon, “Agile Software Development,” *Software: Practice and Experience*, vol. 41, no. 9, pp. 943–944, Aug. 2011.

[55] Y. Tian, D. Lo, and C. Sun, “DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis,” 2013, pp. 200–209.

[56] Atlassian, “Jira | Logiciel de suivi des tickets et des projets,” *Atlassian*. [Online]. Available: https://fr.atlassian.com/software/jira. [Accessed: 06-Apr-2018].

[57] “Home :: Bugzilla :: bugzilla.org.” [Online]. Available: https://www.bugzilla.org/. [Accessed: 06-Apr-2018].

[58] P. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in *Proceedings of the 29th international conference on Software Engineering*, 2007, pp. 499–510.

[59] P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, “The bug report duplication problem: an exploratory study,” *Software Quality Journal*, vol. 21, no. 1, pp. 39–66, Mar. 2013.

[60] G. Canfora and L. Cerulo, “Impact Analysis by Mining Software and Change Request Repositories,” 2005, pp. 29–29.

[61] G. A. Di Lucca, M. Di Penta, and S. Gradara, “An approach to classify software maintenance requests,” in *Software Maintenance, 2002. Proceedings. International Conference on*, 2002, pp. 93–102.

[62] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. do C. Machado, T. F. Vale, E. S. de Almeida, and S. R. de L. Meira, “Challenges and opportunities for software change request repositories: a systematic mapping study.,” *Journal of Software: Evolution and Process*, vol. 26, no. 7, pp. 620–653, Jul. 2014.

[63] K. M. Eisenhardt, “Building Theories from Case Study Research,” p. 20.

[64] B. Flyvbjerg, “Five Misunderstandings About Case-Study Research,” *Qualitative Inquiry*, vol. 12, no. 2, pp. 219–245, Apr. 2006.

[65] R. K. Yin, *Case Study Research, Design and Methods (2nd ed.)*, Sage Publications. cations, Beverly Hills, CA, 1994.

[66] H. Simons, *Case Study Research in Practice*. SAGE, 2009.

[67] J. Gerring, *Case Study Research: Principles and Practices*. Cambridge University Press, 2006.

[68] J. Gerring, “What Is a Case Study and What Is It Good for?,” *American Political Science Review*, vol. 98, no. 02, pp. 341–354, May 2004.

[69] R. K. Yin, *Case Study Research: Design and Methods*. SAGE, 2003.

[70] R. E. Stake, *The Art of Case Study Research*. SAGE, 1995.

[71] P. Baxter and S. Jack, “Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers,” p. 18.

[72] “Bienvenue sur le wiki EasyVista (Documentation.WebHome) - XWiki.” [Online]. Available: https://wiki.easyvista.com/xwiki/bin/view/Documentation/. [Accessed: 18-Sep-2018].

[73] Henderson-Sellers, Brian, and M. K. Serour, “Creating a dual-agility method: The value of method engineering.” Journal of database management 16.4, 2005.

[74] G. Lee and W. Xia, “Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility,” *Mis Quarterly*, vol. 34, no. 1, pp. 87–114, 2010.

[75] N. B. Moe, T. Dingsøyr, and T. Dybå, “A teamwork model for understanding an agile team: A case study of a Scrum project,” *Information and Software Technology*, vol. 52, no. 5, pp. 480–491, May 2010.

[76] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Vol. 1 vols. Upper Saddle River: Prentice Hall, 2002.

[77] V. Venkatesh and F. D. Davis, “A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies,” *Management Science*, vol. 46, no. 2, pp. 186–204, Feb. 2000.

[78] N. Venkatraman and V. Ramanujam, “Measurement of Business Performance in Strategy Research: A Comparison of Approaches,” *The Academy of Management Review*, vol. 11, no. 4, p. 801, Oct. 1986.

[79] A. Weber and R. Thomas, “Key performance indicators,” *Measuring and Managing the Maintenance Function, Ivara Corporation, Burlington*, 2005.

[80] R. S. Kaplan and D. P. Norton, “Transforming the balanced scorecard from performance measurement to strategic management: Part I,” *Accounting horizons*, vol. 15, no. 1, pp. 87–104, 2001.

[81] “A definition of indicator,” *https://en.oxforddictionaries.com*. [Online]. Available: https://en.oxforddictionaries.com/definition/indicator.

[82] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, “Is it a bug or an enhancement?: a text-based approach to classify change requests,” in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, p. 23.

[83] “Change Management: Best Practices,” *Cisco*. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-458050.html. [Accessed: 26-Nov-2017].

[84] T. Wheelen and D. Hunger, “A Descriptive Model of Strategic Management,” *Scribd*. [Online]. Available: https://www.scribd.com/document/29959620/A-Descriptive-Model-of-Strategic-Management-Wheelen-amp-Hunger. [Accessed: 26-Apr-2018].

[85] “Access the Internet, built just for you.,” *Waterfox*. [Online]. Available: https://www.waterfoxproject.org/en-US/. [Accessed: 26-Dec-2018].

[86] “The new, fast browser for Mac, PC and Linux | Firefox,” *Mozilla*. [Online]. Available: https://www.mozilla.org/en-US/firefox/. [Accessed: 26-Dec-2018].

[87] J. Rushby, “Critical system properties: survey and taxonomy,” *Reliability Engineering & System Safety*, vol. 43, no. 2, pp. 189–219, Jan. 1994.

[88] W. M. Tellis, “Application of a Case Study Methodology,” p. 21.

[89] D. Challet and A. Lombardoni, “Bug propagation and debugging in asymmetric software structures,” *Physical Review E*, vol. 70, no. 4, Oct. 2004.

[90] A. E. Hassan and R. C. Holt, “Predicting change propagation in software systems,” in *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.*, Chicago, IL, USA, 2004, pp. 284–293.

[91] E. Isaeva, V. Bakhtin, and A. Tararkov, “Collecting the Database for the Neural Network Deep Learning Implementation,” in *Digital Science*, vol. 850, T. Antipova and A. Rocha, Eds. Cham: Springer International Publishing, 2019, pp. 12–18.

[92] A. Stavrianou, P. Andritsos, and N. Nicoloyannis, “Overview and semantic issues of text mining,” *ACM SIGMOD Record*, vol. 36, no. 3, p. 23, Sep. 2007.

Part II: Synthesis Published Papers

Appendix 1. The Application of Change Indicators in Mining Software Repositories

Published in Trends and Advances in Information Systems and Technologies

Volume 2 [WorldCIST'18, Naples, Italy, March 27-29, 2018]

Authors: Nico Hillah, Thibault Estier.

Pages: 418-428

Advances in Intelligent Systems and Computing 746,

Springer 2018, ISBN 978-3-319-77711-5

DOI: 10.1007/978-3-319-77712-2_40

Abstract. This paper presents a framework to identify a problematic or uncontrollable rise in the number of software change requests and to take right actions to fix it. With this work, we propose the use of an acceptable limit number of change requests as indicators to track the evolution of software change requests. The change indicators are used to identify a periodical sharp rise in demands of change requests fast enough and provide the right fix on time. Not only these indicators track the evolution of change request, but they also help to identify the right solution to address the triggers of these change requests.

Keywords: Change requests, Indicators, Maintenance

1 Introduction

Software maintenance is defined in IEEE Standard 1219-1998 [1] as *"The modification of a software product after delivery to correct faults, to improve the performance or other attributes, or to adapt the product to a modified environment"* [p1,2]. Software developed in the agile context is not an exception to this IEEE definition. These faults corrections and performance improvements are expressed in the form of change requests (CRs). These CRs are collected and saved in repositories such as JIRA, and Redmine. CRs are classified into different categories [3], [4] based on different methods. They are classified either by their assignments [5], [6], their types [3], their duplication [7]; or based on the type of activities performed to solve them [8] e.g. corrective maintenance, adaptive maintenance [9]. The maintenance team, as well as the development team, often implement these CRs without systematically checking the main original causes of these faults. This lack of attention may be encouraged by either a large and overwhelming number of CRs [10],[11] or by the limited resources to find, solve and implement these changes [12]. In fact, the sharp rise in the number of change requests on software becomes problematic in situations where the number of requests is extremely high and their correction is poorly done [7]. The probability that those poorly corrected bugs and implemented CRs lead to catastrophic failure, consequently huge cost in the near future are extremely high [13]. In addition, different studies have proved that the implementing software CRs activity cost up to 90% of the total cost of software product [14],[15]and [16]. For these reasons, the question we address in this paper is "*How can a dramatic rise in the number of CRs be identified and solved?"*

We conduct this study in three parts: (1) we classify change reports using the "*EVOLution of Information System (EVOLIS)*" [4] conceptual framework based on the factors that trigger software evolution. In our case, an evolution is as a group of CRs addressed to a system over its lifetime. (2) For each category of CRs, we propose to define a maximum number of CRs above which the software maintenance team must be alerted. This maximum is referred as the *CR indicator*. We set indicators for each category in order to track the evolution of these CRs and (3) we suggest possible actions to consider in some problematic cases.

In the following sections of the paper, we first introduce the definition of a software change request and its characteristics. Second, we present the methodology we use and explain the steps in classifying the CRs before presenting our results. Finally, we present a four-step process to implement the indicators and the benefits they bring as a contribution of the paper.

2 Literature review

2.1 Software Change Request

Before a bug report or software problem becomes a change request (CR), it goes through a process similar to the incident management process proposed by ITIL [17]. The process proposes different system support levels. At level one or help desk, the user reports problems she faced using the software system. When a satisfactory answer could not be provided to the user at this level of support, the problem becomes a change request. From this point, each CR enters the CR cycle where an analysis determines whether it will be accepted or not, then to whom it will be assigned, and finally how it will be solved and tested to ensure that it has been solved [18]. This is called a Generic change request workflow [19] (see Fig. 1). At this stage of the cycle, the CRs and the associated decisions are saved in a repository.

CRs are managed with the help of platforms where stakeholders such as managers, developers, and customers coordinate activities and share information [19]. These platforms support change request repositories. A CRs repository helps to conduct studies in different fields such as software evolution, change propagation in relation to the software coding, fault analysis, software complexity, and software reuse [20]. A CRs repository not only contributes to other software fields, but it is also a field of studies in itself. Mining CRs repository treats subjects such as how to eliminate duplication of CRs, how to choose a CRs repository [19]. It also presents solutions on how to manage CRs, from their classification to the choice of the developer who will solve them [8], [21]. However, analyzing the literature, we found that the question of how to manage these CRs, more precisely when and how to access their overload and which measures must be put in place to mitigate this overload, is not explored in detail [7],[10].We address this question in this paper by providing indicators to gauge the actual state of CRs. In the next section, we present the framework EVOLIS used to classify the CRs.

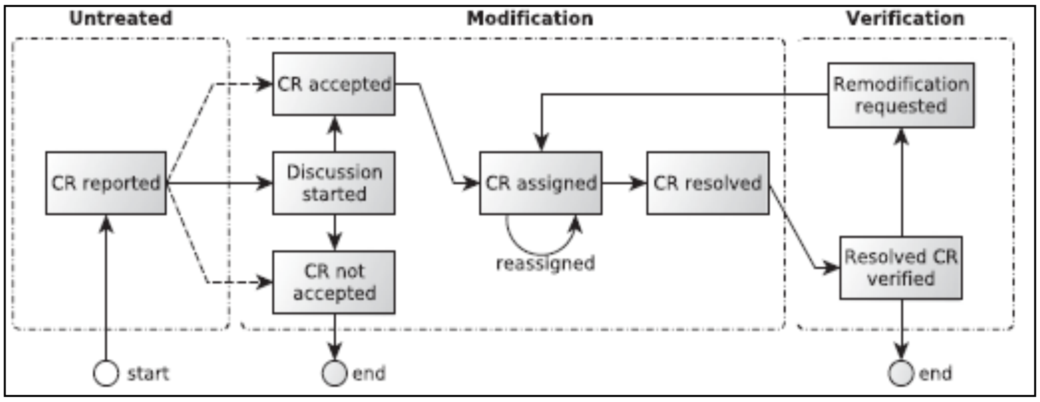


Fig. 1. Generic change request workflow [19].

2.2 Presentation of EVOLIS framework

According to the authors, EVOLIS [4] framework classified CRs as evolution based on factors that trigger them. "*EVOLIS can be caused by a large variety of factors: bugs that need to be fixed, users that wish to have new functionalities, new market opportunities that require new software features, performance standards that the system must reach, technical changes in the environment with which the system must interact, obsolescence of applications and so on*" [p2,4]. EVOLIS presents four categories or blocks of CRs: (1) IS/Users fit change requests (U.F-CRs), defined as any request related to the user interface, the user documentation and aptitude to use the system. Simply, the authors "*classify as IS/user fit each activity during an evolution regarding directly users or when the evolution only alters the fit between IS and users without altering business functionalities*" [p4,4]. (2) The Technology change requests (T-CRs) are related to changes that concerns the software as well as the hardware platforms as information system components. As example, they stated that *"when reason like performance, updates, preventive maintenance and so on motivate evolutions of the software or hardware"* [p5,4]. (3) IS architecture change requests category (A-CRs), according to the authors this type of change request concerns "*different types of integration evolution, namely an evolution of integration among components of the system, among business functionalities, or an integration with systems outside of the company*." [p4,4].(4) The Business-IS alignment change requests (B.IS-CR) *"addresses the co-alignment between business and information systems"* [p3,4]. There are two type of alignment under this category: company external environment alignment and evolution-oriented alignment. For this study, we classified each change requests that is related to align IS to the actual or future business scenario and to the external organization environment alignment as the B.IS CRs.

3 Methodology

To answer the question of this paper, we conducted two case studies. We chose this methodology to be able to study software CRs in a real life, in the organizational context [22]. This methodology allows us to study software CRs in their natural settings [23] and also to generate a framework from this observation. Thus, we collected the data from an educational organization CRs repository. Two software development and maintenance teams provided the data.

Firstly, we conducted the case on software A. Secondly, we evaluated and confirmed our findings in case A by testing them on system B. The two systems are developed in-house using the scrum agile method. Both systems are school management systems. System A has been developed to help schools in managing grades, courses and posting of their students. The system B has been developed to manage the hiring process of teachers and their assignment to classes. The change repository tool used by this organization is JIRA. We analyzed the CRs of the two systems A and B, over a period of 16 months (January 2015-April 2016). System A has nine released versions over this period. The first version of the system A had been released middle 2012. System B has 10 released versions over the same period. System A had 629 CRs and system B had 1450 CRs. We analyzed the CRs of these systems by classifying them with EVOLIS framework [4].

3.1 Data Analysis

Each incident report (IR) is characterized by a source, a description and a help desk person handling the incident. Incidents that could not be solved by the help desk team become CRs and are saved in the repository. The classification of the IR is done in two parts. First, we group the change reports into seven groups based on their descriptions. Then we classify these groups into a four-trigger factor of CRs from EVOLIS. Based on the description of the change reports, we identified seven main groups:

1. Change reports related to the user ability to manipulate the system
2. Change reports related to the user interface
3. Change reports related to system error or system bug
4. Change reports related to another system different from the system in use
5. Change reports related to the business and processing rules
6. Change reports related to the system database and mainly on user access privileges
7. Change reports related to testing of the system done by the user.

The seven groups have been classified into the four CRs groups of EVOLIS framework. Table 1, classification of change reports into EVOLIS CR trigger factors categories, presents this result.

Table 1. Classification of the changes reports into EVOLIS CR trigger factors categories:

			User	*User Interface*	*User test*	*System Bug*	*Another system*	*Rules/ Process*	*User Privilege/ Database*
EVOLIS	Business–IS Alignment	(B.IS-CR)						●	
	IS/Users Fit	(U.F-CR)	●	●	●				
	Technology	(T-CR)				●			●
	IS/Architecture	(A-CR)					●		

4 Change Indicator

4.1 Data Analysis of System A

The results of analyzing CRs on system A based on EVOLIS [4] framework are shown in Table 2, System A change request categories. This table presents the number of CRs for each EVOLIS CRs category from January 2015 until April 2016 on system A.

Table 2. System A change request categories

Month	*A-CRs*	*B.IS-CRs*	*T-CRs*	*U.F-CRs*	*Total CRs*
January	5	15	3	7	30
February	0	12	6	27	45
March	3	4	6	10	23
April	2	14	9	25	50
May	5	8	10	16	39
June	19	20	17	25	81
July	13	4	7	10	34
August	9	8	6	20	43
September	1	4	9	13	27
October	1	1	2	6	10
November	9	2	19	11	41
December	4	8	10	6	28
January	21	11	9	13	54
February	16	13	20	16	65
March	4	5	12	4	25
April	10	8	10	6	34
Total	122	137	155	215	629

During the classification of CRs of system A based on their description or factors that trigger them (EVOLIS), we observe that the number of CRs varies among different categories and from one month to another. This raised two questions: (1) How to evaluate the level status of CRs? And (2) on which bases one can prove that CRs of one month are overloaded as against another month, to justify the need of allocating enough resources to address this excess of CRs?

4.2 Definition and Implementation of the Change Indicator on System A

The absence of standards to evaluate which month has a CRs overload compared to the others has prompted us to introduce an indicator as the accepted limit of CRs over a fixed period of time, in our case monthly. The purpose of this indicator is to alert software maintenance team on the status of software CRs in each category. These categories or types of CRs depend on the repository mining techniques [19]. CR indicator is set as a certain limit for each category of change above which, CRs must not only be solved but the category concerned must be investigated. Due to the variance of the level of CRs month-to-month as well as category-to-category, we could not base the limit on a fixed amount of change requests. Nevertheless, we decided to introduce CRs number limit as an indicator that will help us to identify cases where the rise in the number of changes is unusual. The indicator helps to control the evolution of the change request. Moreover, it will trigger corrective actions.

The Oxford English dictionary defines indicator as *"A thing that indicates the state or level of something."* [24]. Based on this definition, we set the indicator artifact as an arithmetic value that informs us on the CRs level in each category. We defined the condition of this artifact indicator as follows: "If the number of change requests for a particular month in a category is greater than twice the average number of all previous change requests up to this month in the same category; then an investigation must be conducted for this month, within this category". The condition of our indicators is expressed as follows:

$$\begin{gathered}\mathrm{k} = 1..\mathrm{n}, \qquad \text{k being a month and} \\ \text{x, the change requests number to test} \\ If\ \ x_n > 2\bar{x}_{n-1}\ (1) \\ \text{then investigate \& adopt mitigation actions}\end{gathered}$$

We must precise that the limit setting of the measures to observe depends on the goal set by each software team or each organization [9]. In our case, our emphasis is on the sudden increase in CRs for a month compared to the number of CRs of the previous months. The application of this formula implies that the change indicator calculated for the first month is irrelevant. In case of a progressive rise in CRs, another formula will be more appropriate for setting the indicators. Our indicator rule is only an example of the form a CR indicator may have. The results obtained in applying the indicator to the system A are summarized in Table 3. Months with CRs that pass the indicators as limit are marked with gray color in the table. Considering the system A and its architecture CRs category, we have three months (May, June, January) for which their total number of CRs is twice greater than the average of previous CRs. Consequently, we conducted an investigation and we found that those months correspond to a period of a special event such as examination as well as the reopen class sessions. This implies that the interaction between the system A and the other systems is crucial for the users to perform their activities. A possible set of actions to put in place in this situation will be to have a checking of the interacting systems before these major events (checking of APIs, of updates, etc.). In Fig.2, we present the result of applying CR indicator on system A classified CRs.

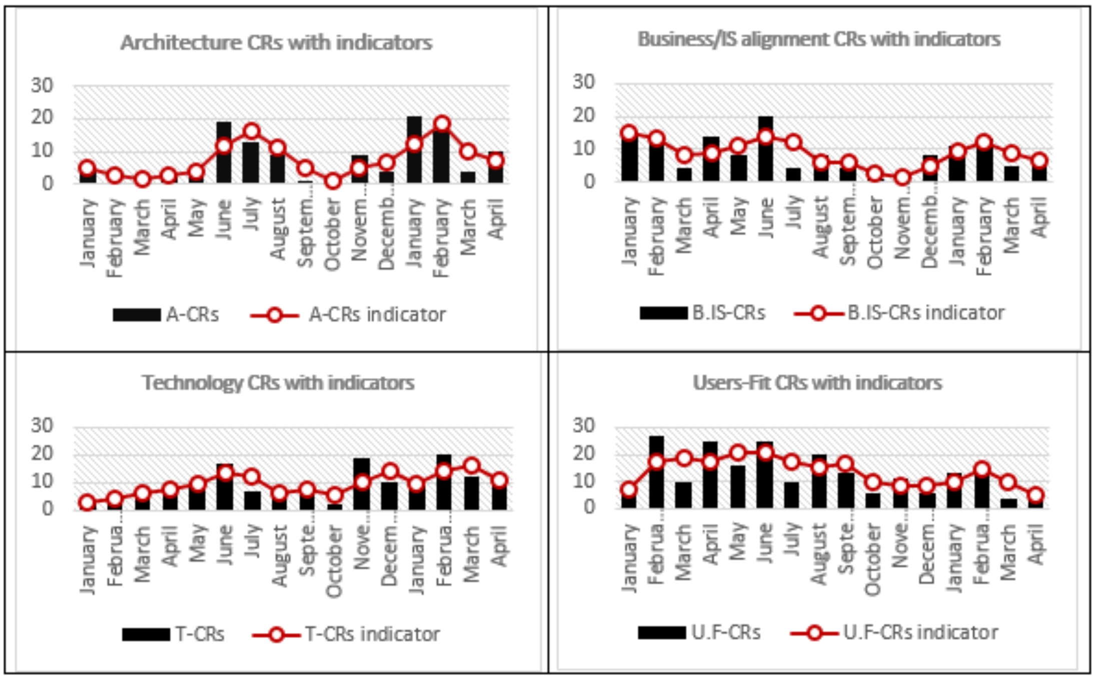


Fig. 2. System A change requests categories with the indicators

Table 3. Applying the indicator to categories of CRs system A

Month	A-CRs	*A-CRs Indicator*	B.IS-CRs	*B.IS-CRs Indicator*	T-CRs	*T-CRs Indicator*	U.F-CRs	*U.F-CRs Indicator*	Total CRs
January	5	*5.0*	15	*15*	3	*3*	7	*7*	30
February	0	*2.5*	12	*13.5*	6	*4.5*	**27**	*17*	45
March	3	*1.5*	4	*8*	6	*6*	10	*18.5*	23
April	2	*2.5*	14	*9*	9	*7.5*	25	*17.5*	50
May	**5**	*3.5*	8	*11*	10	*9.5*	16	*20.5*	39
June	**19**	*12.0*	**20**	*14*	17	*13.5*	25	*20.5*	81
July	13	*16.0*	4	*12*	7	*12*	10	*17.5*	34
August	9	*11.0*	8	*6*	6	*6.5*	20	*15*	43
September	1	*5.0*	4	*6*	9	*7.5*	13	*16.5*	27
October	1	*1.0*	1	*2.5*	2	*5.5*	6	*9.5*	10
November	**9**	*5.0*	2	*1.5*	**19**	*10.5*	11	*8.5*	41
December	4	*6.5*	8	*5*	10	*14.5*	6	*8.5*	28
January	**21**	*12.5*	**11**	*9.5*	9	*9.5*	13	*9.5*	54
February	16	*18.5*	13	*12*	**20**	*14.5*	16	*14.5*	65
March	4	*10.0*	5	*9*	12	*16*	4	*10*	25
April	10	*7.0*	8	*6.5*	10	*11*	6	*5*	34
Total	122		137		155		215		629

4.3 Evaluation of the Change Indicator with System B

In this second case, we applied the same classification and setting up indicator process to the system B CRs. Similarly, in the first case, there are some months where the CRs are problematic. The results are present in Fig 3, System B change requests with indicators. For instance, in the category of user-fit, the month of August has fallen under the investigating criteria. Conducting this investigation helped us to identify a problem related to a change done on the user interface; change that prevents users to correct manually the name of participants in certain activities.

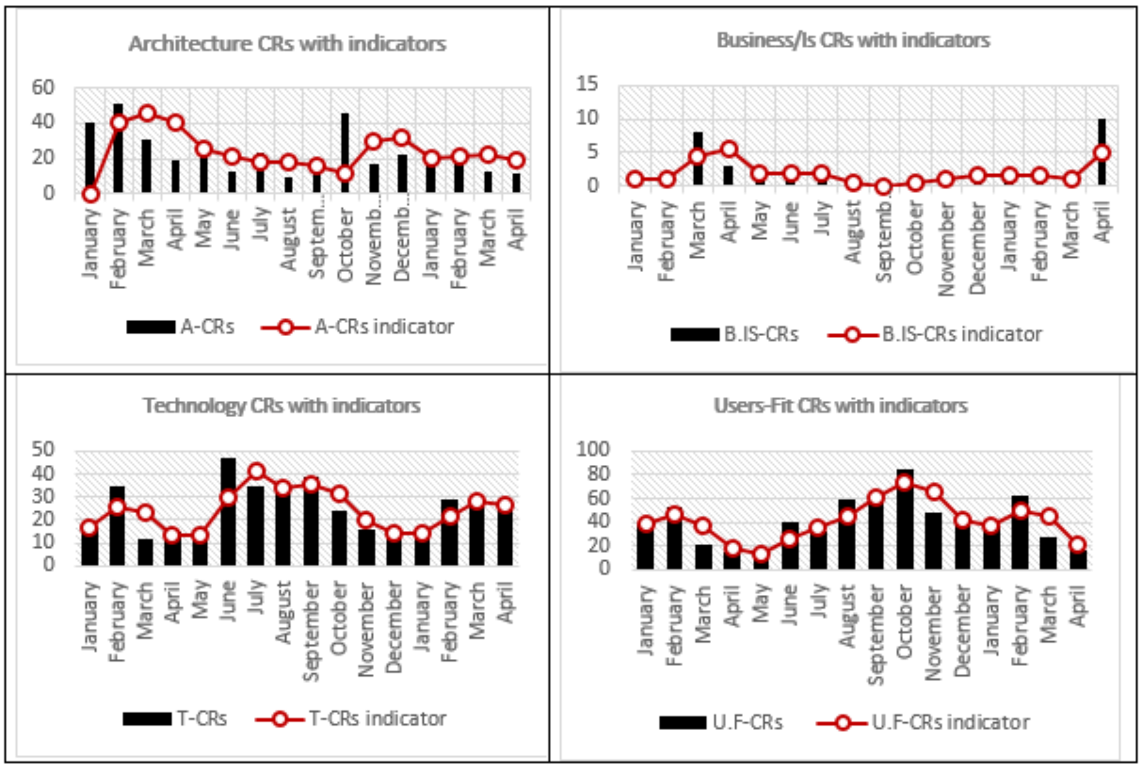


Fig. 3. System B change requests categories with the indicators

5 Contributions

In this section, we describe the four-step process organizations must implement to track the evolution of each type of CRs and must be able to identify the excessive ones. We argue that implementing our proposed process will help software maintenance and development team to track the evolution of CRs, their state and triggers factors that generate excessive CRs. By adopting this process, organizations gain time and resources advantage. In fact, it allows software team to point to the exact modules or parts of the software system responsible for the sharp rise in CRs. Knowing these affected modules allows taking measures to stop the rise in CRs and provides the right solutions in fixing them. Consequently, it reduces the cost of software maintenance in general.

The process to follow to classify CRs and set CR indicators is summarized in these four steps:

1. Collection of the CRs into a repository. For this purpose, there are existing tools such as JIRA and Redmine.

2. Classification or triage of these CRs into categories. The triage features are integrated into the existing CR repositories cited in the previous step. In case this incorporated feature does not satisfy the software team or the project manager, there are other frameworks and models or the traditional data mining techniques such as clustering or classification algorithms [3]; with which this classification can be done. In our case, we use EVOLIS [4] as a CR framework to classify the CRs.
3. Set a limit for each type of CR over a defined period. At this level, one or more indicators must be defined to track the evolution of each type of CRs category. In addition, at this step, the indicator definition depends on the objectives set by the organization [9] or the project team. Indicators can be defined as a ratio or as a target number to reach [24],[25]. In defining these parameters, the team must also consider the population size of stakeholders who can report a CR.
4. Investigation step: Define a set of actions to undertake when an indicator is reached. For instance, organize a user training section in case the category related to the user-fit indicators is reached.

Fig. 4 presents the diagram that describes our proposed four-step process in setting the CR indicators.

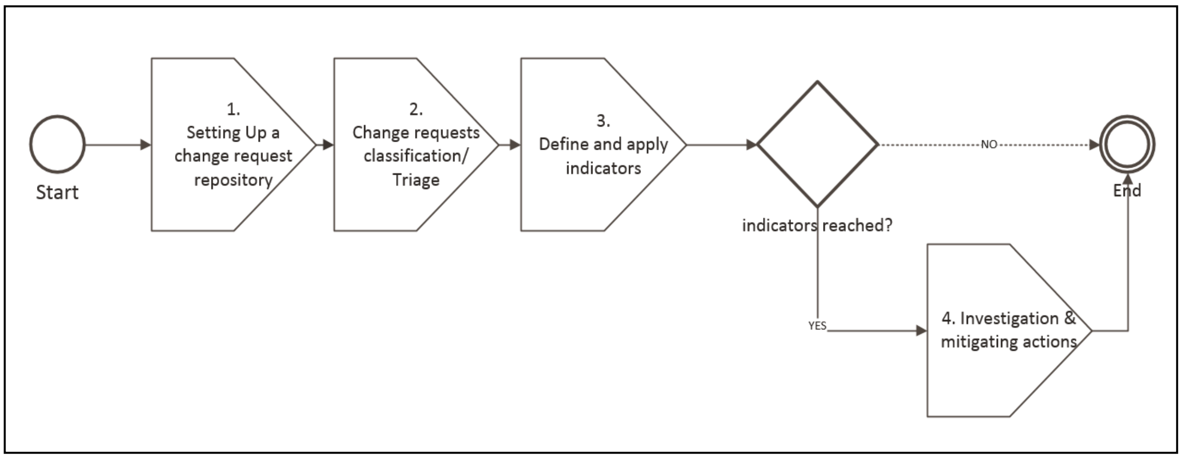

Start
1. Setting Up a change request repository
2. Change requests classification/ Triage
3. Define and apply indicators
indicators reached?
NO
End
YES
4. Investigation & mitigating actions

Fig. 4.The steps in setting change request indicators and mitigating actions.

In case of software emergency maintenance, the question on how to prioritize and manage these emergency CRs have already been studied [26],[27]. Different solutions emerged and suggested that the software maintenance team may put in place special means for reporting and tracking these emergency CRs. The team must also conduct a frequent investigation on these CRs by applying our proposed CR indicators concept. This will ensure that possible hidden problems and malfunctions related to the IS are identified, so that the right fix is provided on time.

6 Conclusion

Tracking the CR evolution by classifying them based on factors that trigger them provides a means for decision-making in software maintenance. We analyzed the CRs of two systems using the conceptual framework EVOLIS [4]. As result, we found that tracking CRs only is not sufficient to propose the right actions to tackle their triggers, but it is also necessary to set indicators to monitor their evolution. In addition, a list of actions must be prepared to handle each problematic case. We summarized this result in the four-step process diagram. Implementing this four-step process diagram helps to identify which type of CRs trigger factors have most solicited the organization resources in maintaining and changing the software systems.

In our future work, we will extend this analysis to the entire software ecosystem to identify all potential actors and factors that trigger software changes. We will also identify how to address the ones that cause the higher cost to the software development and maintenance.

7 References

[1] IEEE Std 1219-1998: IEEE Standard for Software Maintenance, IEEE. .

[2] S. Brand, How buildings learn: what happens after they're built, Rev. pbk. ed. London: Phoenix, 1997.

[3] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," in Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds, 2008, p. 23.

[4] A. Métrailler and T. Estier, "EVOLIS Framework: A Method to Study Information Systems Evolution Records," in System Sciences (HICSS), 2014 47th Hawaii International Conference on, 2014, pp. 3798–3807.

[5] G. A. Di Lucca, M. Di Penta, and S. Gradara, "An approach to classify software maintenance requests," in Software Maintenance, 2002. Proceedings. International Conference on, 2002, pp. 93–102.

[6] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in Proceedings of the 28th international conference on Software engineering, 2006, pp. 361–370.

[7] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," 2005, pp. 35–39.

[8] N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil, and W.-G. Tan, "Types of software evolution and software maintenance," J. Softw. Maint. Evol. Res. Pract., vol. 13, no. 1, pp. 3–30, 2001.

[9] E. B. Swanson, “The dimensions of maintenance,” in Proceedings of the 2nd international conference on Software engineering, 1976, pp. 492–497.

[10] X. Xia, D. Lo, X. Wang, and B. Zhou, “Dual analysis for recommending developers to resolve bugs,” J. Softw. Evol. Process, vol. 27, no. 3, pp. 195–220, Mar. 2015.

[11] P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, “The bug report duplication problem: an exploratory study,” Softw. Qual. J., vol. 21, no. 1, pp. 39–66, Mar. 2013.

[12] R. W. Butler and G. B. Finelli, “The infeasibility of quantifying the reliability of life-critical real-time software,” IEEE Trans. Softw. Eng., vol. 19, no. 1, pp. 3–12, 1993.

[13] K. T. Ryan, “Software processes for a changing world: Software processes for a changing world,” J. Softw. Evol. Process, vol. 28, no. 4, pp. 236–240, Apr. 2016.

[14] B. Ulziit, Z. A. Warraich, C. Gencel, and K. Petersen, “A conceptual framework of challenges and solutions for managing global software maintenance.,” J. Softw. Evol. Process, vol. 27, no. 10, pp. 763–792, Oct. 2015.

[15] M. M. Lehman, “Programs, life cycles, and laws of software evolution,” Proc. IEEE, vol. 68, no. 9, pp. 1060–1076, 1980.

[16] F. Jaafar, Y.-G. Guéhéneuc, S. Hamel, and G. Antoniol, “Detecting asynchrony and dephase change patterns by mining software repositories.,” J. Softw. Evol. Process, vol. 26, no. 1, pp. 77–106, Jan. 2014.

[17] A. Hochstein, Z. Rüdiger, and B. Walter, “ITIL as common practice reference model for IT service management: formal assessment and implications for practice,” In e-Technology, e-Commerce and e-Service, 2005. EEE’05. Proceedings.

[18] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, “Assigning change requests to software developers,” J. Softw. Evol. Process, vol. 24, no. 1, pp. 3–33, Jan. 2012.

[19] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. do C. Machado, T. F. Vale, E. S. de Almeida, and S. R. de L. Meira, “Challenges and opportunities for software change request repositories: a systematic mapping study.,” J. Softw. Evol. Process, vol. 26, no. 7, pp. 620–653, Jul. 2014.

[20] G. Canfora and L. Cerulo, “Impact Analysis by Mining Software and Change Request Repositories,” 2005, pp. 29–29.

[21] I. Aljarah, S. Banitaan, S. Abufardeh, W. Jin, and S. Salem, “Selecting discriminating terms for bug assignment: a formal analysis,” 2011, pp. 1–7.

[22] R. K. Yin, Case Study Research, Design and Methods (2nd ed.), Sage Publications. cations, Beverly Hills, CA, 1994.

[23] I. Benbasat, D. K. Goldstein, and M. Mead, “The Case Research Strategy in Studies of Information Systems,” MIS Q., vol. 11, no. 3, p. 369, Sep. 1987.

[24] “A definition of indicator,” *https://en.oxforddictionaries.com*. [Online]. Available: https://en.oxforddictionaries.com/definition/indicator.

[25] “Change Management: Best Practices - Cisco.” [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-458050.html. [Accessed: 26-Nov-2017].

[26] J. Kanwal and O. Maqbool, “Bug Prioritization to Facilitate Bug Report Triage,” *J. Comput. Sci. Technol.*, vol. 27, no. 2, pp. 397–412, Mar. 2012.

[27] Y. Tian, D. Lo, and C. Sun, “DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis,” 2013, pp. 200–209.

Appendix 2. A Conceptual Tool to Improve the Management of Software Defects

Published in Business Modeling and Software Design – 8th International Symposium, BMSD 2018, Vienna, Austria, July 2-4, 2018, Proceedings.
Author: Hillah Nico
Pages: 443-451,
Lecture Notes in Business Information Processing 319,
Springer 2018, ISBN 978-3-319-94213-1
DOI: 10.1007/978-3-319-94214-8_35
Postprint Version.

Abstract. Software teams address software defect problems in a simple way: they identify them, assign them and resolve them. Nevertheless, studies have proven that having only these activities as approaches to handle a large and increasing number of software defects is inefficient. As a solution to this, we propose in this study a managerial conceptual tool for mining software defects in order to improve the management of SDs. With our proof of concept, we demonstrate how SDs mining management can be enhanced from a strategic and operational view. This is done through the precise definition of software defects' management objectives in line with the objectives of the software product owner.

Keywords: Defects mining, Software defect management, Control measures.

1 Introduction

IEEE standard 1044-2009 [1] defines a defect as: *"An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced".*

Not only the software defects (SDs) are present in the whole life cycle of a software product, but different studies also proved that 80% of the total cost of the software life cycle is associated with the management of the SDs [2]. Having this high impact on the software product, SDs management must be crucial to software teams as well as to organizations. Nowadays, the management of SDs does not only consists of identifying, assigning, and correcting them, but also in mining them. The purpose of this study is to focus on the mining aspect of SDs management.

In fact, there are different studies which propose solutions on how to mine SDs [2], [3], [4]. However, most of these existing techniques are limited to the collection, the classification, and the assignment of the SDs. In addition, these techniques do not cover the question of how to define specific SDs mining management objectives that are aligned with first, the SDs management objective, and second, with the objectives of the software product owner. This results in a poor resource allocation in mining SDs as well as in the absence of control over the SDs management in a software life cycle. In this regard, the problem we address in this paper is *how to improve and control SDs mining management in alignment with the business objectives of the software owner?*

As a solution to this problem, we are proposing the use of our conceptual tool to control the mining management of the SDs. This conceptual tool is a guideline with four stages.
The paper will proceed as follows: first, we will define the software defect and its management approaches. Secondly, we will present the conceptual tool that we used to conduct the proof of concept. Finally, we will present the results and the advantages that we gained from applying it.

2 Related Works

The defects are the source of software failures and problems. Software failures are defined as *"Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits*" [p5, 3].

In the last decade, SDs management has received a considerable amount of attention from researchers. In fact, SDs management has been the center of interest for many studies in different software studies subdomains such as software project management, software engineering and evolution [6], [7]. Due to the diversity of these studies, we group them into branches based on their interest in SDs management.

The first branch deals with questions such as how to collect and store these SDs. Studies related to this branch provide answers to questions such as how to collect SDs or which SDs characteristics must be documented [8]. These studies propose solution tools named bug-tracking systems to help collect SDs. They take the form of a central hub accessible by project managers and software developers to manage the software products. Some of these online tools are Jira [9] and Bugzilla [10].

The second branch deals with questions such as how to assign SDs to developers or how to deal with the problem of an SDs duplication [11]. The research in this branch proposes techniques and methods such as algorithms to automatically assign SDs to the right developer [12], [13], [14] and also techniques to eliminate the duplication of SDs [15].

The third branch deals with the triage and the mining of SDs. In the software life cycle, the mining of defects presents many advantages [2]. Researchers as well as practitioners in this branch have proposed schemas and taxonomies for mining SDs. The best-known schemas are (1) The Orthogonal Defect Classification (ODC) of IBM [16], the root cause analysis [1], (2) the HP Defect origins, types and modes [17] and standards like the IEEE standard 1044-2009 [5]. In the same context, they also apply data mining methods such as the Naïve Bayes Model [13] or the regression model [2] to classify SDs. In fact, the classification of defects helps the software development teams to reduce the cost of correcting SDs and helps them detect defective modules. This study is conducted as part of this last branch. In fact, our goal is to propose a conceptual tool to improve the quality of software mining results in an organization, since these results will lead to decision making concerning the quality of the software systems. The need to assure that the mining is rightly performed with defined targets will improve decision making concerning the state of software quality.

3 Presentation of the Conceptual Tool

In order to provide a means to avoid the insignificant SDs mining results to software product owners, we decided to propose this conceptual tool.

Mining SDs is a complex set of activities; it goes from selecting a technique to mine the SDs, interpreting the obtained results, to taking a decision based on the obtained results. Moreover, each software system is unique, thus needs a specific SDs mining management strategy, e.g., the SDs of the system Waterfox are not the same for Firefox, even though they have similar functionalities and purpose. Due to this complexity, inefficient SDs mining can lead to situations such as:

(1) the results obtained from the SDs mining are irrelevant for the product owner;
(2) the SDs mining is requiring much more resources than planned and software teams failed to take decisions in order to improve the quality of the software system based on the mining results;
(3) the mining goals are poorly aligned with the strategy and the objectives of SDs management and the product owner's business needs and;
(4) control and evaluation measures for obtained results are missing. To avoid these problems, we are proposing this conceptual tool to guide SDs miners wishing to improve their mining project.

Although there are similar existing conceptual tools in the literature for business domain [18], our conceptual tool is designed to target the SDs mining management field. The aim of this conceptual tool is to help software teams to define their SDs mining management strategy and to specify concrete actions to put in place this strategy in mining SDs. The conceptual tool is defined fourfold:

(1) The first step consists of defining the SDs mining management strategy in alignment with the needs of the software product owner. The strategy must be broken down into short or medium term goals to achieve. The software team, as well as the product owner, must approve these goals, e.g., a defect mining management strategy may be the improvement of the software quality by developing error-free programs for each software version released. The approval of these goals will lead to the second stage.
(2) The next step, which is the operational level, is to convert this strategy into concrete objectives. Referring to the previous example, the set of objectives will be to improve the detection of the defect modules and predict SDs.
(3) Following this, each objective must be broken down into terms of specific actions to be performed, e.g., classifying SDs according to their priorities. In addition, members of the SDs mining are responsible to implement each of these actions.

(4) Following this and depending on the actions put in place, software teams must carefully select control measures to evaluate the state of the actions, e.g., the ratio of the corrected high level prioritized SDs over the total number of SDs received.

Finally, the software team must define a list of actions to establish in order to correct cases where the set objectives have not been reached, e.g. reorganization of the process to detect SDs. Fig 1 presents the process to follow to implement the proposed conceptual tool.

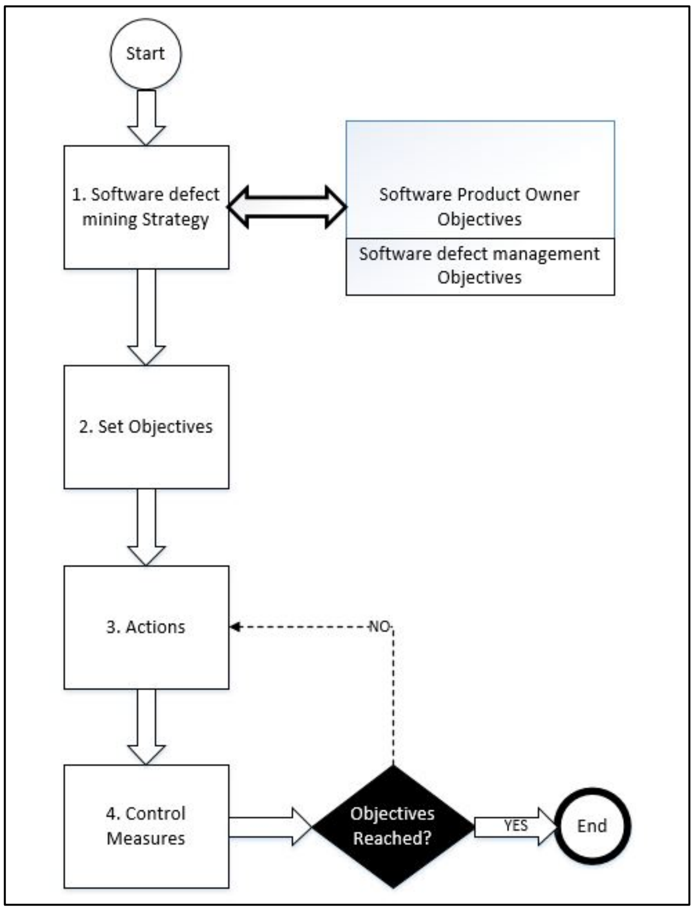

Start
1. Software defect mining Strategy
Software Product Owner Objectives
Software defect management Objectives
2. Set Objectives
3. Actions
NO
4. Control Measures
Objectives Reached?
YES
End

Fig. 1. The process of the conceptual tool

4 The Application of the Software Defects Managerial Conceptual Tool

In order to apply the proposed conceptual tool to improve and control the SDs mining management in practice, we decided to conduct a proof of concept of a software system that we will name system A. This system is developed using the scrum method. The owner of this system A is an education company. Its purpose is to help schools in managing the grades of their students. The overall objectives set by the owner of the system A is to have a software system with a considerable high quality, with emphasis on its availability to the users, especially during the exam period. In line with this objective, the software team objective aims to improve the assignment and the correction of the SDs.

4.1 Stage 1 and 2: Strategy Definition and Set of Objectives

In alignment with the owner's objective, our strategy would be to mine SDs in order to reduce the impact of SDs on the system to limit the system's unavailability time (stage 1). In the next step (stage 2), we cascade the defined strategy in different objectives such as to reduce the impact of defects on system A and possibly to improve the correcting process of the SDs. In the next step, we defined a set of actions to implement the objective of reducing the impact of defects on the system (stage 3). We first need to know the actual number of defects of this system A and then classify them according to their impact. To do this, we classify SDs according to their severity in order to analyze the different impact that they are having on the system availability. In the next session, we will present how we classified the SDs of system A, and then we will present the application of the final stage on this system A.

4.2 Stage 3: The Classification of SDs of System A

We analyzed the SDs of system A over a period of a year, from January 2015 to December 2015. System A has 522 SDs. We analyzed the SDs of this system by classifying them according to the defect severity attribute of IEEE 1044-2009 standards (see Table 2). This severity attribute is one of the most used attributes in SDs classification in practice [19]. The main advantage of choosing the severity attribute is the possibility for managers to identify which defect should be first corrected [19]. The IEEE's standard defines this attribute as *"The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering."* [5]. There are five values of severity. They are classified from the most significant to the least significant (see Table 1).

Table 1. Severity values [5]

Attribute	Value	Definition
Severity	Blocking (B)	Testing is inhibited or suspended pending correction or identification of suitable workaround.
	Critical (C)	Essential operations are unavoidably disrupted, safety is jeopardized, and security is compromised.
	Major (Mj)	Essential operations are affected but can proceed.
	Minor (Mn)	Nonessential operations are disrupted.
	Inconsequential (I)	No significant impact on operations.

Table 2. System A software defects classification

	Severity					
	B	C	Mj	Mn	I	Total
Jan	2	3	28	10	0	43
Feb	1	5	15	11	0	32
Mar	7	8	34	14	0	63
Apr	2	5	38	9	2	56
May	3	1	20	5	0	29
June	0	2	25	15	0	42
July	0	1	5	4	0	10
Aug	2	7	8	7	3	27
Sept	5	11	18	15	4	53
Oct	7	4	22	6	0	39
Nov	15	8	37	20	3	83
Dec	7	5	18	12	3	45
Total	51	60	268	128	15	522

4.3 Stage 4: The Section of Control Measures

There are two important aspects to consider when selecting the evaluation metrics at the fourth stage of this conceptual tool. The first one is to choose metrics based on the objective or action to evaluate, e.g. a ratio of the corrected SDs over the total number of SDs received to evaluate the SDs' correction process. The second one is to take into consideration the critical level [20] of the system being managed. This critical level can relate to its business, security, and safety aspect. In addition, to determine the critical level of the system, software teams must consult and get the approval of the product owner.

To track and evaluate the success of our objective, we selected a metric as an indicator (stage 4). In this regard, we defined the SDs indicator as a ratio of the number of a type of SDs over the total SDs number received within a month. This ratio informs us about the type of defects that is problematic during the month. We define a problematic case as follows: when the number of a certain type of SDs is higher or equal to one-third of the total SDs number of a month. One-third of the total SDs is an agreed upon limited number a type of SDs may have during a month. The selection of this metric was based on system A's critical mission, which is its availability during the exam periods. We defined the indicator as follows:

$$n \ \ expresses\ the\ type\ of\ SDs\ to\ evaluate$$

$$X,\ \ the\ value\ of\ SDs\ and\ t\ being\ a\ time\ period$$

$$If\ X_n t \geq \frac{\sum(X)t}{3} \tag{1}$$

$$then\ investigate$$

Following this, we defined a list of actions to undertake in order to correct problematic cases. These actions are:

- to investigate within the problematic type of SDs to identify miscorrected defects;
- to reorganize the process of correcting the problematic type of SDs;
- to check other indicators such as the number of correcting defects over the total SDs within this category.

Our choice of action depends on the investigation results. In this regard, an investigation must be conducted when an indicator is reached, in order to identify the problem and to provide the right fix on time. In Fig. 2, we present the application of our indicator on SDs of system A in 2015.

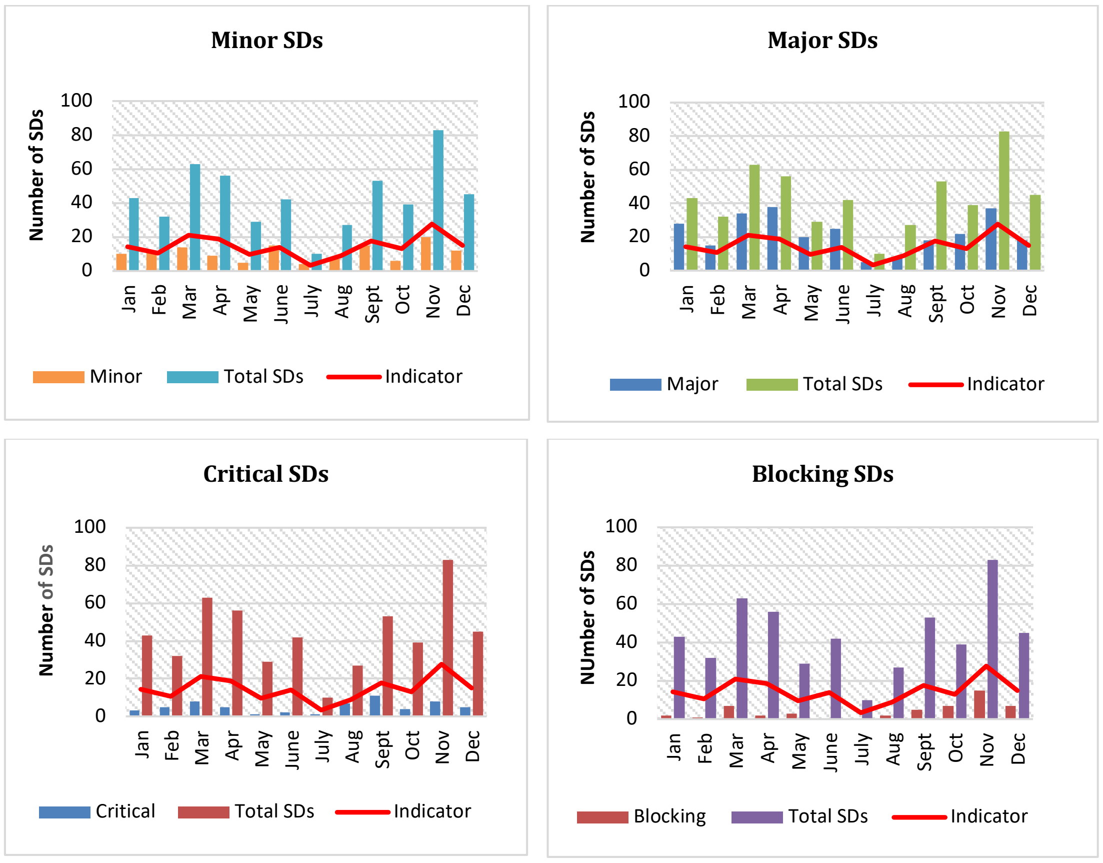


Fig. 2. Classified SDs of system A with the indicators.

5 Discussion and Contribution

The results clearly show us that the group of minor, blocking, and critical SDs management has reached the objective set in relation to the organization's objectives. In opposition, the major type of SDs failed to reach the fixed goal. In fact, from January until June, the number of the major SDs was considerably high. After investigating those months, we found that the high number of SDs of January was due to the duplication of SDs. Similarly, the month of March inherited some of the SDs of January that were incorrect. E.g., mistakes found in the names of some of the students were related to the use of ACSII format in system A and corrected in the system in January; but the same mistakes reappeared in the month of March, due to the use of an API to connect system A to an external system B. This information leads to the assignment process reorganization for the major type of SDs.

Applying this conceptual tool gives not only the insight of the SDs mining management, but also of the entire SDs management. In fact, knowing the status of each type of SDs will guide the SDs manager to focus on the problematic group of SDs and to reorganize the resource allocation in handling these groups of SDs. This improves the decision-making in managing SDs. Consequently, it improves the SDs management altogether.

The application of this conceptual tool is a manner not only to improve the management of SDs but also to align this management with the objectives of the software product owner. Its implementation is also flexible concerning the objectives set by each organization and its software department. In addition, the selection of control measures to evaluate the management must be customized for each software product.

This conceptual tool alerted us to bring the management of SDs into line. Most of all, it did not demand many interventions from us once we set it up. We propose this conceptual tool not only as contribution, but we also demonstrate its application in a real case.

6 Conclusion

Our proof of concept presents some of the advantages that software teams can gain from implementing our conceptual tool. This tool not only helps to define the precise objectives in line with the objectives of software owners in the context of SDs mining management but also guides the owners to evaluate the state of their SDs management. Indeed, the defined control measures will alert them to possible existing problems related to the management of their SDs and, therefore, of their software products. Knowing this, they will be able to take the right actions to handle the SDs. Herewith they will, on the one hand, considerably achieve the set goals, on the other hand, improve the quality of the software product, and reduce the cost of its development or maintenance. In our future work, we will provide a deep insight into the process of defining and implementing appropriate SDs management strategies by looking at the interdependence among the SDs management branches.

7 References

1. M. Leszak, P. Dewayne E., and D. Stoll, “Classification and evaluation of defects in a project retrospective,” *Elsevier*, no. 61, pp. 173–187, 2002.

2. G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The impact of using regression models to build defect classifiers,” in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 135–145.

3. H. Kagdi, M. L. Collard, and J. I. Maletic, “A survey and taxonomy of approaches for mining software repositories in the context of software evolution,” *J. Softw. Maint. Evol. Res. Pract.*, vol. 19, no. 2, pp. 77–131, Mar. 2007.

4. S. Davies, M. Roper, and M. Wood, "Comparing text-based and dependence-based approaches for determining the origins of bugs: COMPARING APPROACHES FOR DETERMINING BUG ORIGINS," *J. Softw. Evol. Process*, vol. 26, no. 1, pp. 107–139, Jan. 2014.

5. *1044-2009 IEEE Standard Classification for Software Anomalies*. 2009.

6. M. Fischer, M. Pinzger, and H. Gall, "Analyzing and relating bug report data for feature tracking," in *WCRE*, 2003, vol. 3, p. 90.

7. Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. do C. Machado, T. F. Vale, E. S. de Almeida, and S. R. de L. Meira, "Challenges and opportunities for software change request repositories: a systematic mapping study.," *J. Softw. Evol. Process*, vol. 26, no. 7, pp. 620–653, Jul. 2014.

8. Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis," 2013, pp. 200–209.

9. Atlassian, "Jira | Logiciel de suivi des tickets et des projets," *Atlassian*. [Online]. Available: https://fr.atlassian.com/software/jira. [Accessed: 06-Apr-2018].

10. "Home :: Bugzilla :: bugzilla.org." [Online]. Available: https://www.bugzilla.org/. [Accessed: 06-Apr-2018].

11. P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Proceedings of the 29th international conference on Software Engineering*, 2007, pp. 499–510.

12. J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?," in *Proceedings of the 28th international conference on Software engineering*, 2006, pp. 361–370.

13. G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

14. I. Aljarah, S. Banitaan, S. Abufardeh, W. Jin, and S. Salem, "Selecting discriminating terms for bug assignment: a formal analysis," 2011, pp. 1–7.

15. P. A. da Mota Silveira Neto, D. Lucrédio, T. Vale, E. S. de Almeida, and S. R. de Lemos Meira, "The bug report duplication problem: an exploratory study," *Softw. Qual. J.*, vol. 21, no. 1, pp. 39–66, Mar. 2013.

16. R. Chillarege *et al.*, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, 1992.

17. J. T. Huber, "A Comparison of IBM's Orthogonal Defect Classification to Hewlett Packard's Defect Origins, Types, and Modes." Hewlett Packard Company, 1999.

18. T. Wheelen and D. Hunger, "A Descriptive Model of Strategic Management," *Scribd*. [Online]. Available: https://www.scribd.com/document/29959620/A-Descriptive-Model-of-Strategic-Management-Wheelen-amp-Hunger. [Accessed: 26-Apr-2018].

19. S. Wagner, “Defect classification and defect types revisited,” in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 39–40.

20. J. Rushby, “Critical system properties: survey and taxonomy,” *Reliab. Eng. Syst. Saf.*, vol. 43, no. 2, pp. 189–219, Jan. 1994.

Appendix 3. Severe Software Defects Trigger Factors: A Case Study of a School Management System

Published in Digital Science. DSIC18 2018.
Advances in Intelligent Systems and Computing,
vol 850. Springer, Cham

Author: Nico Hillah

DOI: 10.1007/978-3-030-02351-5_44
Print ISBN 978-3-030-02350-8
Postprint Version.

Abstract. In this paper, we identify the groups of triggers that are responsible for severe software failures. These failures prevent any essential operation or activity to be conducted through the concerned system or other systems connected to it. In fact, the occurrence of these failures causes a double financial cost to organizations: one in fixing them and the other one because of the unavailability of the system or systems. We targeted three types of software defects as sources of these failures. We conducted this study by classifying 665 software defects of a school management system and we found that the top two trigger groups are the technology and the IS architecture groups.

Keywords: Software defect severity, Software defect triggers, Software defect classification.

1 Introduction

In this age of information, every organization uses software systems to perform all types of activities in different domains. Unfortunately, most of the time, they are subject to failures [1], [2]. Software failures are defined as "*Termination of the ability of a product to perform a required function or its inability to perform within previously specified limits*" [p5, 3].

In fact, depending on their severity, these failures induce not only financial loss to organizations, but also time and resource loss in correcting them. Software defects (SDs) are the sources of these failures. IEEE standard 1044-2009 [p5, 3] defines a defect as: *"An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced".* Different studies have investigated the sources and factors triggering SDs [4], [5]. Nevertheless, there is no existing literature on the types of triggers associated with the level of SDs severity that they generate. Knowing which types of triggers generate which level of SDs severity will help systems administrators in particular and organizations in general to better allocate their resources in order to address software failures. In this regard, the question we address in this paper is *which types of trigger factors generate the most severe SDs?*

To answer this question, we conducted a case study on a software system. In fact, we performed two main classifications of its SDs. The first classification was to identify the severity of SDs; then we classified the same SDs based on the trigger factors using EVOLIS framework [4].
The paper is structured as follows: first, we will introduce existing techniques software defects classification; secondly, we will present the methodology that we used in conducting this study; then, we will show the results that we obtained; and finally, we will present our contribution.

2 Related Works

In the software life cycle, the classification of defects presents many advantages [6]. The classification of defects helps the software development teams to reduce the cost of correcting them, to detect defective modules and to have efficient resource planning. Various studies have proposed and evaluated different approaches to collect and to analyze these SDs. The main approaches are (1) taxonomies [7], [8], root cause analysis [9], schemes [3] and the classification of these SDs [10].

There are different existing schemes in classifying SDs [10]. (1) The Orthogonal Defect Classification (ODC) of IBM [11] was developed in 1992 by R. Chillarege et al. [11] and it classifies defects across *"the dimensions (1) defect type, (2) source, (3) impact, (4) trigger, (5) phase found, and (6) severity"* [12]. (2) The HP Defect Origins, Types and Modes, the approach of Hewlett Packard, was developed by the HP software metrics in 1986 [13] and this scheme classifies the defects according to their types, their origins and their mode [12]. (3) The IEEE standard 1044-2009 is the scheme we retain for our first classification project. We selected this approach because it proposes the most complete definition of the SDs severity types among the three schemes. Moreover, this severity attribute is one of the most used attributes in SDs classification in practice [12]. The main advantage of choosing the severity attribute is the possibility for managers to identify which defect to correct first [12]. We retain the severity attribute for our first classification.

3 Methodology

To be able to understand the relation existing between triggering factors for SDs and the severity of SDs, we conducted a case study of a system that we will name system A. This system is developed by an educational organization and it is a school management system. Its purpose is to help schools in managing the grades of their students. It is used for managing more than 90000 students' grades. The first version of the system A had been released middle 2012. We classified 665 SDs of system A. The collection of SDs covers a period of one year and four months from January 2015 to April 2016. System A has nine released versions over this period. The defects repository tool used by this organization is Jira [14].

Our objective is to classify SDs according to their severity and then classify these same SDs according to the factors that trigger them. In fact, we analyzed the SDs of this system A by classifying them according to the defect severity attribute of IEEE 1044-2009 standards [3] and then by classifying them with EVOLIS framework [4]. The software team in charge of maintenance of the system A and a member of our research team had conducted both classification.

3.1 The Classification of SDs Based on Severity

Our first classification is done based on the severity attribute of the IEEE standard (see Table 2.). The IEEE's standard defines this attribute as "*The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering."* [3]. The five values of severity are classified from the most significant to the least significant ones (see Table 1). For the purpose of this study, we define any software defect (SD) as severe as long as it belongs to one of these severity levels: (1) Blocking (B), (2) Critical (C) and (3) Major (Maj). The Minor (Min) and the Inconsequential (Inc) are not considered as severe SDs.

Table 1. Severity values [3]

Attribute	Value	Definition
Severity	Blocking (B)	Testing is inhibited or suspended pending, correction or identification of suitable workaround.
	Critical (C)	Essential operations are unavoidably disrupted, safety is jeopardized, and security is compromised.
	Major (Maj)	Essential operations are affected but can proceed.
	Minor (Min)	Nonessential operations are disrupted.
	Inconsequential (Inc)	No significant impact on operations.

Table 2. Classification of System A's SDs based on their severity

	B	C	Maj	Min	Inc	Total
Jan	10	12	46	22	0	90
Feb	5	8	24	17	0	54
March	12	17	49	31	0	109
April	9	8	50	16	2	85
May	3	1	20	5	0	29
June	0	2	25	14	0	41
July	0	1	5	4	0	10
Aug	2	7	8	7	3	27
Sept	5	11	18	15	4	53
Oct	7	4	22	6	0	39
Nov	15	8	37	20	3	83
Dec	7	5	18	12	3	45
Total	75	84	322	169	15	665

3.2 The Classification of SDs Based on the EVOLIS Framework

For our second classification project, we chose the EVOLIS framework [4] (see Table 3.). This framework proposes a technique to classify SDs according to the factors that trigger them. "*EVOLIS can be caused by a large variety of factors: bugs that need to be fixed, users that wish to have new functionalities, new market opportunities that require new software features, performance standards that the system must reach, technical changes in the environment with which the system must interact, obsolescence of applications and so on*" [3]. EVOLIS identifies four main groups of factors that triggers SDs: (1) IS/users fit (U.F) triggers that are defined as any failure related to the user interface, the user documentation and aptitude to use the system. (2) The technology (TCH) triggers are related to defects that concern the software as well as the hardware platforms as information system components. (3) According to the authors, the IS architecture (ACH) triggers concern "different types of integration evolution, *namely an evolution of integration among components of the system, among business functionalities, or an integration with systems outside of the company*." [3]; and finally (4) the Business-IS (Bs-IS) alignment triggers that *"address the co-alignment between business and information systems"* [3].

Table 3. Classification of System A's SDs based on their trigger factors

	ACH	Bs-IS	TCH	U.F	Total
Jan	13	23	21	33	90
Feb	13	9	18	14	54
March	15	26	25	43	109
April	27	26	13	19	85
May	4	8	10	7	29
June	9	2	19	11	41
July	1	1	2	6	10
Aug	1	4	9	13	27
Sept	15	8	8	22	53
Oct	16	4	9	10	39
Nov	23	19	18	23	83
Dec	5	10	12	18	45
Total	142	140	164	219	665

Subsequently, we grouped these results into a two-dimensional table (see Table 4.). Each dimension represents the results obtained for each classification project.

Table 4. Two-dimensional classification of SDs of system A

	B	C	Maj	Min	Inc	Total
ACH	17	12	99	14	0	142
Bs-IS	24	21	66	28	1	140
TCH	19	30	87	27	1	164
U.F	15	21	70	100	13	219
Total	75	84	322	169	15	665

4 Discussion and Contribution

We analyzed the results threefold: the results of severity classification, followed by the results of EVOLIS and then we combined and analyzed both results together. First, the severity classification showed us that the top three high types of SDs are respectively the major type of SDs with 322 SDs, followed by the minor type with 169 and the critical type with 84 SDs (see Table 2.). Second, the EVOLIS classification showed us that the top three groups of factors that trigger SDs are respectively the IS/users factors with 219, followed by the technology factors with 164 and then the factors related to the IS architecture and business-IS alignment (see Table. 3). These last trigger groups have almost the same number of SDs: 142 SDs for the IS architecture and 140 SDs for the business-IS alignment SDs. Further, the analysis of both combined results showed us that technology triggers represent respectively 12%, 18% and 53% for blocking SDs, critical SDs and major SDs (see Fig. 1). In total, the technology triggers are responsible for 83% of the severe SDs. Similarly, the architecture triggers represent respectively 12%, 8% and 70% for blocking SDs, critical SDs and major SDs (see Fig. 1). The business-IS alignment represents respectively 17%, 15% and 47% for blocking SDs, critical SDs and major SDs. Finally, the IS/users triggers represent 20% of the total of severe SDs (see Fig. 1).

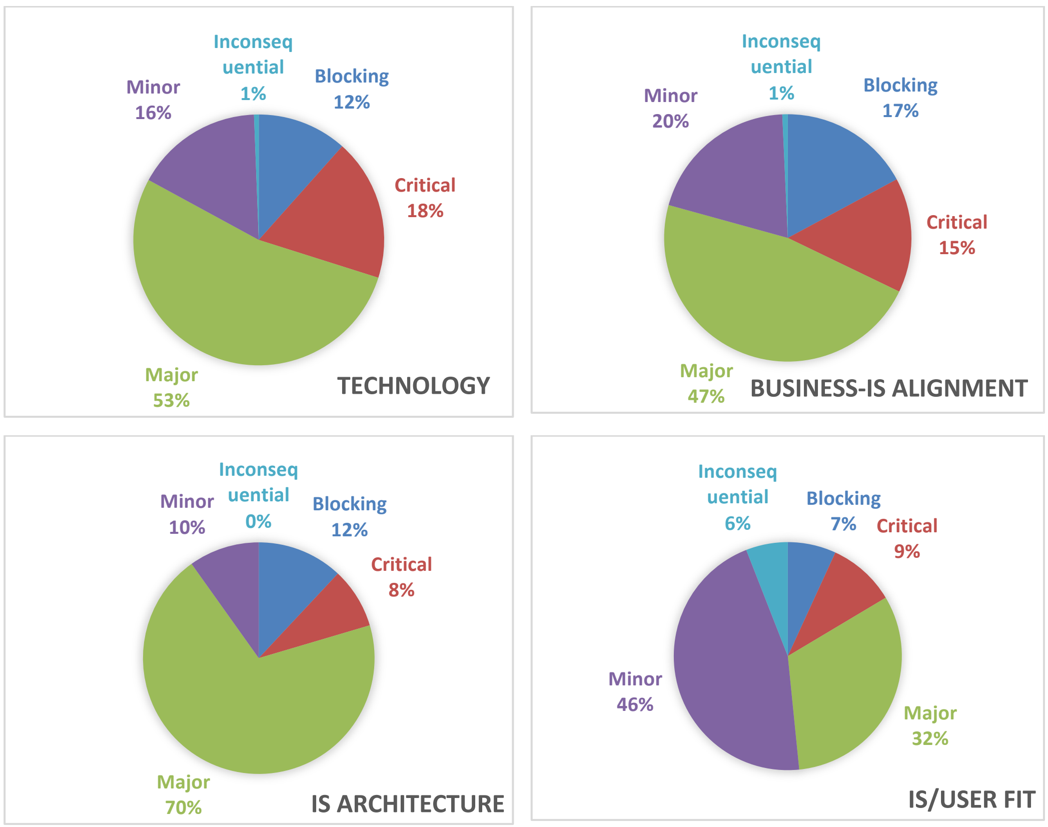


Fig. 1. Trigger factors and severity of SDs of System A

Analyzing these results separately did not give so much information to organize the SDs management. However, when we put them together, we found that some type of SDs trigger factors are the source of some specific severe groups of SDs. In fact, we observed that the majority of the inconsequential SDs are triggered by the IS/users fit factors.

This implies that the probability of an inconsequential SDs to be triggered by either the IS architecture, the business-IS alignment or the technology factors are very low or barely existent. Furthermore, we assigned weighting factors to each severity level according to their importance as followed (see Table 5.):

Table 5. The weighting factors for the severity level.

Severity level	Weighting factor
Blocking	40%
Critical	30%
Major	20%
Minor	8%
Inconsequential	2%

We then apply this weighted scoring model to our two-dimension table to calculate the weighted scores (W) for severe SDs per trigger factor (see Table 6.).

Table 6. Severe SDs weighted score.

Severe SD	Weight (W)	Bs-IS	W-Bs-IS	ACH	W-ARC	TCH	W-TCH	U.F	W-U.F
Blocking	0.4	24	9.6	17	6.8	19	7.6	15	6
Critical	0.3	21	6.3	12	3.6	30	9	21	6.3
Major	0.2	66	13.2	99	19.8	87	17.4	70	14
Total	0.9	111	**29.1**	128	**30.2**	136	**34**	106	**26.3**

Looking at these results, we can conclude that the technology trigger factors, with the highest weighted score 34, are responsible for most of the severe SDs followed by the IS architecture factors, with 30 weighted score. Then the business-IS alignment, with 29.1, and finally the IS/users fit triggers, with 26.3 (see Fig. 2.). We can also notice that there is a considerable gap between the number of SDs of the first two groups of triggers (IS architecture and technology and the last two of them (business-IS alignment and IS/user fit).

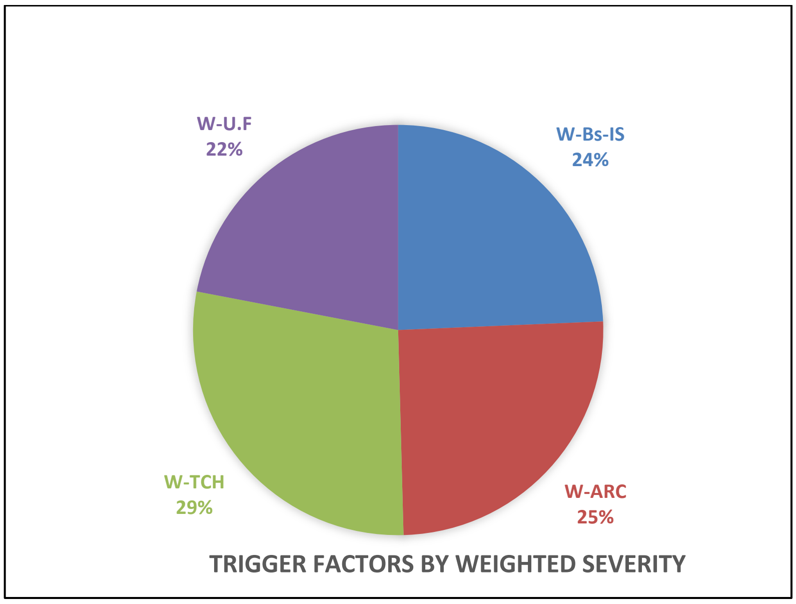


Fig. 2. Trigger factors by weighted severity of system A

5 Conclusion

To the question of which groups of SDs triggers generate the most severe SDs, we answered that the technology triggers are at the head position with a total of 34 weighted score. In the second position is the IS architecture triggers which comes with 30.2 weighted score, and then followed by the business-IS alignment triggers with 29.1. The last position is occupied by the IS/users fit triggers with 26.3 weighted score of the total severe SDs analyzed. We also found that the majority of the defects triggered by IS/user factors are either minor or inconsequent types of SDs.

The results obtained from this study will help software managers to improve the management of SDs by allocating the SDs correction resources more accurately thus reduce the cost of managing SDs. In our future work, we will analyze other software systems and compare their results to the ones we obtained in this study. We will also investigate in depth this close relation between the first couple of trigger groups (IS architecture and technology) and the last couple of trigger groups (business-IS alignment and IS/users fit).

6 References

[1] R. N. Charette, "Why software fails [software failure]," *Ieee Spectr.*, vol. 42.9, pp. 42–49, 2005.

[2] R. Kaur and D. J. Sengupta, "Software Process Models and Analysis on Failure of Software Development Projects," *Int. J. Sci. Eng. Res.*, vol. 2, no. 2, p. 4.

[3] *1044-2009 IEEE Standard Classification for Software Anomalies*. 2009.

[4] A. Métrailler and T. Estier, “EVOLIS Framework: A Method to Study Information Systems Evolution Records,” in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, 2014, pp. 3798–3807.

[5] A. A. Alshazly, A. M. Elfatatry, and M. S. Abougabal, “Detecting defects in software requirements specification,” *Alex. Eng. J.*, vol. 53, no. 3, pp. 513–527, Sep. 2014.

[6] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The impact of using regression models to build defect classifiers,” in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 135–145.

[7] R. Binder, *Testing object-oriented systems: models, patterns, and tools*. Reading, Mass: Addison-Wesley, 2000.

[8] D. Vallespir, F. Grazioli, and J. Herbert, “A framework to evaluate defect taxonomies,” in *XV Congreso Argentino de Ciencias de La Computación*, 2009.

[9] M. Leszak, P. Dewayne E., and D. Stoll, “Classification and evaluation of defects in a project retrospective,” *Elsevier*, no. 61, pp. 173–187, 2002.

[10] N. Mellegãrd, *Improving Defect Management in Automotive Software Development, LiDeC—A Light-weight Defect Classification Scheme*. Chalmers University of Technology, 2013.

[11] R. Chillarege *et al.*, “Orthogonal defect classification-a concept for in-process measurements,” *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, 1992.

[12] S. Wagner, “Defect classification and defect types revisited,” in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 39–40.

[13] J. T. Huber, “A Comparison of IBM’s Orthogonal Defect Classification to Hewlett Packard’s Defect Origins, Types, and Modes.” Hewlett Packard Company, 1999.

[14] Atlassian, “Jira | Logiciel de suivi des tickets et des projets,” *Atlassian*. [Online]. Available: https://fr.atlassian.com/software/jira. [Accessed: 06-Apr-2018].

Appendix 4. Classification of Software Defects Triggers: A Case Study of School Resource Management System

Published in the 2019 International Conference on Information Technology & Systems. ICITS 2019. Advances in Intelligent Systems and Computing, vol 918. Springer.

Author: Nico Hillah

ISBN 978-3-030-11889-1

DOI: 10.1007/978-3-030-11890-7_30

Postprint Version.

Abstract. In this work, we identify trigger factors of software defects that are responsible for severe defects. We conducted a case study on a system by classifying 842 defects according to their trigger factors and then identified the level of severity they have on this system. Knowing these types of triggers helps software maintenance teams improving the management of software defects by reducing the cost of maintaining the system, consequently the cost of software projects.

Keywords: Software Defect Triggers, Software Severity, Defects Classification.

1 Introduction

IEEE standard 1044-2009 [1] defines Software defects (SDs) as *“An imperfection or deficiency in a work product where that work product does not meet its requirements or specifications and needs to be either repaired or replaced.”* [1].

The classification of SDs helps the maintenance teams reducing the cost of correcting software bugs, detecting defective modules, and having efficient resource planning. Various studies have proposed and evaluated different approaches to collect and to analyze these SDs [1]–[4]. Other studies target the source of these defects by providing schemas and frameworks to help identifying these sources [5]–[8]. For our project, we have retained the EVOLIS framework [7] to identify the trigger factors that are at the source of the SDs.

To be able to know which factors among these trigger factors have a more severe impact on the system, we conducted a case study on a school resources management system. In fact, we studied the SDs of this system by identifying their trigger factors based on the EVOLIS framework [7] and then identified their severity weight on the system. The question we address in this paper is “*how to identify problematic SDs trigger groups using their severity weight?”*

The paper will proceed as follows; first, we will define the software defect and its classification approaches. Second, we will present the case study, the classification results, and our contribution.

2 Related Works

2.1 The EVOLIS Framework

For our first classification project, we chose the EVOLIS framework [7]. This framework proposes a technique to classify SDs according to the factors that trigger them. *“EVOLIS can be caused by a large variety of factors: bugs that need to be fixed, users that wish to have new functionalities, new market opportunities that require new software features, performance standards that the system must reach, technical changes in the environment with which the system must interact, obsolescence of applications and so on*” [7]. EVOLIS identifies four main groups of factors that trigger SDs: (1) IS/users fit triggers (U.F) that are defined as any defect related to the user interface, the user documentation, and aptitude to use the system. (2) The technology triggers (TCH) are related to defects that concern the software as well as the hardware platforms as information system components. (3) According to the authors, the IS architecture triggers (ACH) concern *“different types of integration evolution, namely an* *evolution of integration among components of the system, among business functionalities, or an integration with systems outside of the company.*" [7], and finally (4) the business-IS alignment triggers (B.IS) that "*address the co-alignment between business and information systems"* [7].

2.2 Software Defects

Previous research studies have proposed different approaches and schemas to classify SDs: the best-known schemas are (1) The Orthogonal Defect Classification (ODC) of IBM [6], the root cause analysis [5], (2) the HP Defect origins, types, and modes [8], and standards like the IEEE standard 1044-2009 [1]. In the same context, they also apply to data mining methods such as the Naïve Bayes Model [9], Clustering [10] or the regression model [11] to classify SDs. IEEE standard1044-2009 also proposed a SDs classification approach based on their severity, priority, and origins. The IEEE standard 1044-2009 is the classification approach we retain for our second classification project. In fact, our objective is to classify SDs according to their trigger factors and their severity [1].

The approach of IEEE standard 1044-2009 proposes a simple and complete definition of the SDs severity types. Moreover, this severity attribute is one of the most used attributes in SDs classification in practice [12]. Our second SDs classification project is based on this attribute. The main advantage of choosing the severity attribute is the possibility for managers to identify which defects to correct first [12]. The IEEE's standard defines this attribute as "*The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering."* [1]. There are five values of severity. They are classified by the most significant to the least significant ones in terms of the impact they have on the system (see Table 1).

Table 1. Severity values [1]

Attribute	Value	Definition
Severity	Blocking	Testing is inhibited or suspended pending correction or identification of suitable workaround.
	Critical	Essential operations are unavoidably disrupted, safety is jeopardized, and security is compromised.
	Major	Essential operations are affected but can proceed.
	Minor	Nonessential operations are disrupted.
	Inconsequential	No significant impact on operations.

3 Methodology

3.1 Case Presentation

In order to identify the trigger factors that generate defects with the highest severity impact on the system, we conducted a case study of a school resources management system that we will name system B. The software development method used to develop system B is the scrum agile method [13]. A government institution owns it. The system is used for the human and material resources management of public schools. We classified 842 SDs of system B. The collection of SDs covers a period of 23 months from June 2014 to May 2016. The bug repository tool used by this organization is Jira [14]. The first version of the system B had been deployed at the beginning of 2013.

Each of these defects has the following information: the identity of the failure reporter, the date of reporting and solving the software defect (SD). In addition, each SD contains its description, the person who reported the case and the person who treated it. We only take into consideration the description characteristic in classifying these SDs.

3.2 Classification Method

Overall, we did three main classifications. First, we analyzed the SDs of system B by classifying them with the EVOLIS framework [7] (see Table 2.). Second, we classified the same SDs according to the defect severity attribute of IEEE 1044-2009 standards [1] (see Table 3.). Third, we combined both classifications. In detail, our method of classifying these SDs consists of four main steps:

In the first step, we collected SDs of system B from the Jira repository [14].

In the second step, we took each SD and identified its trigger factor or source based on its description. At this stage, we used the EVOLIS framework. We named this step "EVOLIS Classification"

In the third step (Severity Classification), we took again each SD and evaluated its severity impact (cost) on the system. This classification is done based on our severity-weighting model (see table 4).

In the final step, we took each EVOLIS-Severity couple and ranked them according to the level of damage they may have on system operations (EVOLIS-severity classification).

Table 2. Classification of System B's SDs based on their trigger factors (EVOLIS)

Years	IS architecture	business-IS alignment	technolo gy	IS/users fit	Total
2014	61	11	118	57	247
2015	87	22	251	90	450
2016	23	10	78	34	145
Total	171	43	447	181	842

Table 3. Classification of System B's SDs based on their severity

Year	Blocking	Critical	Major	Minor	Inconsequential	Total
2014	31	57	71	85	3	247
2015	57	112	135	145	1	450
2016	21	19	33	71	1	145
Total	109	188	239	301	5	842

4 Discussion and Contribution

4.1 Discussion

In this section, we analyzed the classification results threefold: the results of EVOLIS classification, followed by the results of severity classification, and finally, we combined and analyzed both results together.

First, the EVOLIS classification showed us that the top three groups of factors that trigger SDs are respectively the technology factors with 447 SDs, followed by the IS/Users factors with 181 SDs, and then the factors related to the IS architecture with 171 SDs. In the last position is the business-IS alignment with 43 SDs (see Table 2.).

Second, the severity classification showed us that the top three high types of SDs are respectively the Minor type of SDs with 301 SDs, followed by the Major type with 239, and the Critical type with 188 SDs. The Blocking types are fourth with 109 SDs, and, in the last position, we find the Inconsequential SDs type with only five SDs (see Table 3.).

We combined the two results in order to identify the groups of triggering factors that cause severe SDs (see Fig. 1.). Doing so, we realized that limiting the results only to the number of SDs for each severity level group raises an ambiguity. In fact, counting only the number of SDs per trigger factor group does not give us the clear response on which trigger factors are responsible for severe SDs. E.g., how can we determine if five Blocking SDs have affected a system more than eight Critical SDs? In order to clear this ambiguity, we have associated a weighting factor to each level of severity according to their impact on the system (see table 4.). In addition, based on their definition, we separated the severity level into two groups: the first group we called "severe SDs" and the second group we named "no severe SDs". The severe SDs are any SD that has an impact preventing the system to be operational. This group of SDs usually causes financial loss or any considerable resource loss to the system owner and to the system users. They are Blocking, Critical, and Major severity types.

The "no severe SDs" are any SD that has an impact level that do not affect the system's operation: they are Minor and Inconsequential severity type. Thus, for the purpose of this study, we only considered the first group of severity level to be authentic severe SDs.

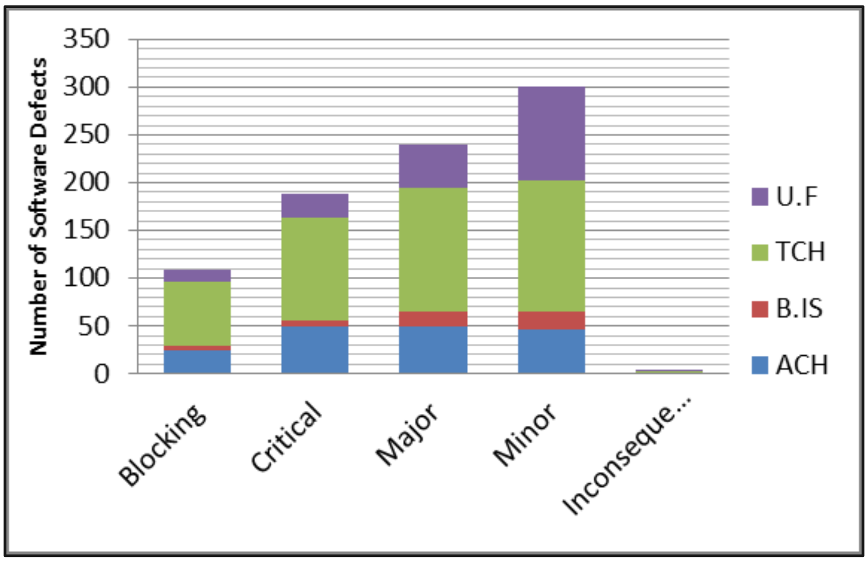


Fig. 1. Trigger factors by severity of system B

Table 4. The weighting factors for the severity level.

Severity level	Weighting Factor
Blocking	40%
Critical	30%
Major	20%
Minor	8%
Inconsequential	2%

We then calculated the weighted score (W) for each trigger factor group based on the severity weight (see table 5.).

Table 5. Severe SDs weighted score of system B

	Weight	ACH	W-ACH	B.IS	W-B.IS	TCH	W-TCH	U.F	W-U.F
Blocking	0.4	25	**10**	4	**1.6**	68	**27.2**	12	**4.8**
Critical	0.3	49	**14.7**	6	**1.8**	109	**32.7**	24	**7.2**
Major	0.2	50	**10**	15	**3**	129	**25.8**	45	**9**
Total	0.9	124	**34.7**	25	**6.4**	306	**85.7**	81	**21**

Integrating both, results allow us to identify SD triggers that are causing more severe impact to the system. These results show that the technology trigger factors with the highest weighted score of 85.7, are responsible for most of the severe SDs followed by the IS architecture factors, with a weighted score of 34.7. Then come the IS/users fit triggers with a 21, and finally the business-IS alignment, with 6.4 (see Fig. 2.).

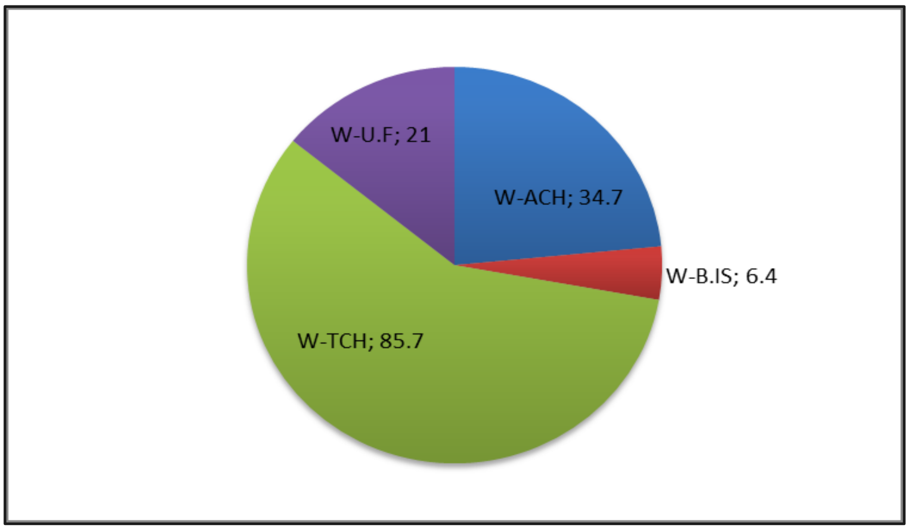


Fig. 2. Trigger factors by weighted severity of system B

4.2 Similar Case Study: Classifications of SDs of System A

Similarly, in our previous publication we conducted the same study on another system we named system A [15]. In this section, we will present this second case and compare its results to the one of system B.

System A is a school management system and it belongs to an educational institute. Its purpose is to help schools in managing the grades of their students. More than 1500 teachers use this system for managing more than 90000 student grades. The first version of the system A had been released mid- 2012. We classified in total 665 SDs of this system. The collection of SDs covered a period of 16 months, from January 2015 to April 2016. System A has released nine versions over this period [15]. The integrated classifications' results we obtained from this study are as follows (see Table 6):

Table 6. Severe Weighted score of system A

Severe SD	Weight	B.IS	W-B.IS	ACH	W-ARC	TCH	W-TCH	U.F	W-U.F
Blocking	0.4	24	9.6	17	6.8	19	7.6	15	6
Critical	0.3	21	6.3	12	3.6	30	9	21	6.3
Major	0.2	66	13.2	99	19.8	87	17.4	70	14
Total	0.9	111	**29.1**	128	**30.2**	136	**34**	106	**26.3**

The results of system A showed that the technology trigger factors, with the highest weighted score 34, are responsible for most of the severe SDs followed by the IS architecture factors, with a weighted score of 30.2. Then come the business-IS alignment, with a 29.1, and, finally the IS/users fit triggers, with 26.3 (see Fig. 3). We can also notice that there is a considerable gap between the number of SDs of the first two groups of triggers (IS architecture and technology) and the last two of them (business-IS alignment and IS/users fit). In the next section, we will present our contributions.

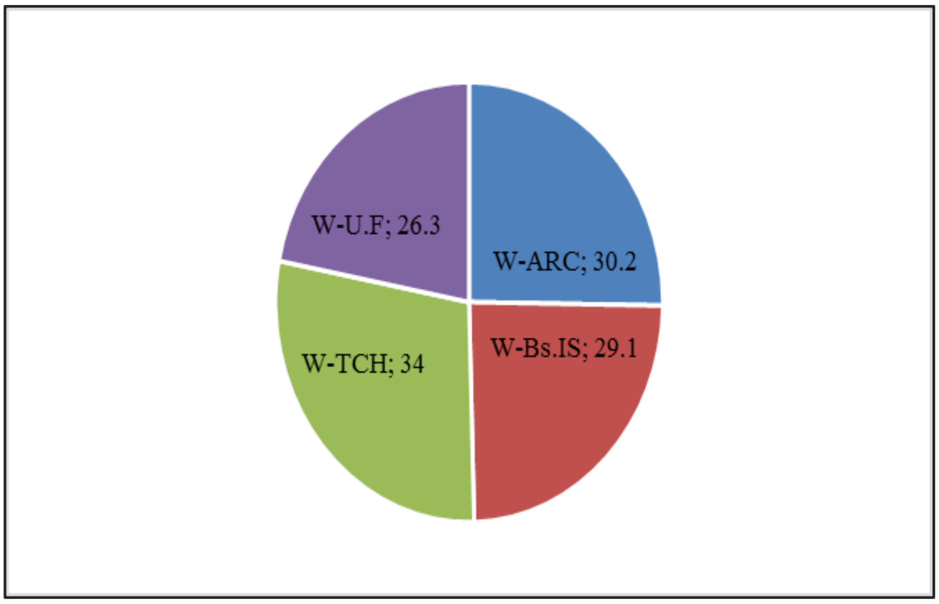


Fig. 3. Trigger factors by weighted severity of system A

4.3 Contribution

The contribution of this paper is twofold:

First, to the question, *"how to identify problematic SDs trigger groups using their severity weight?"* we proposed our 4-steps method.

In order to make possible for other practitioners and other researchers to conduct and possibly observe similar results on their systems we summarized our method as follows:

- Step 1. Data collection: this step consists of collecting SDs of the system to study.
- Step 2. Identification of each SD triggering factor: in this step, we classify the SDs based on the EVOLIS framework in order to identify their trigger factors (EVOLIS classification).
- Step 3. Weighting of the severity level of each SD on the system. Here we classify the same SDs based on the severity attribute of IEEE standard 1044-2009 (Severity classification).
- Step 4. Integrate results of steps 2 and 3: At this level, we classified the SDs based on both EVOLIS and IEEE 1044-2009 severity attribute in order to identify SDs having high severe impacts on the studied systems. This step is our EVOLIS and Severity classification.

We summarized these steps in Fig. 4. We must also point out that step 2 and 3 are interchangeable.

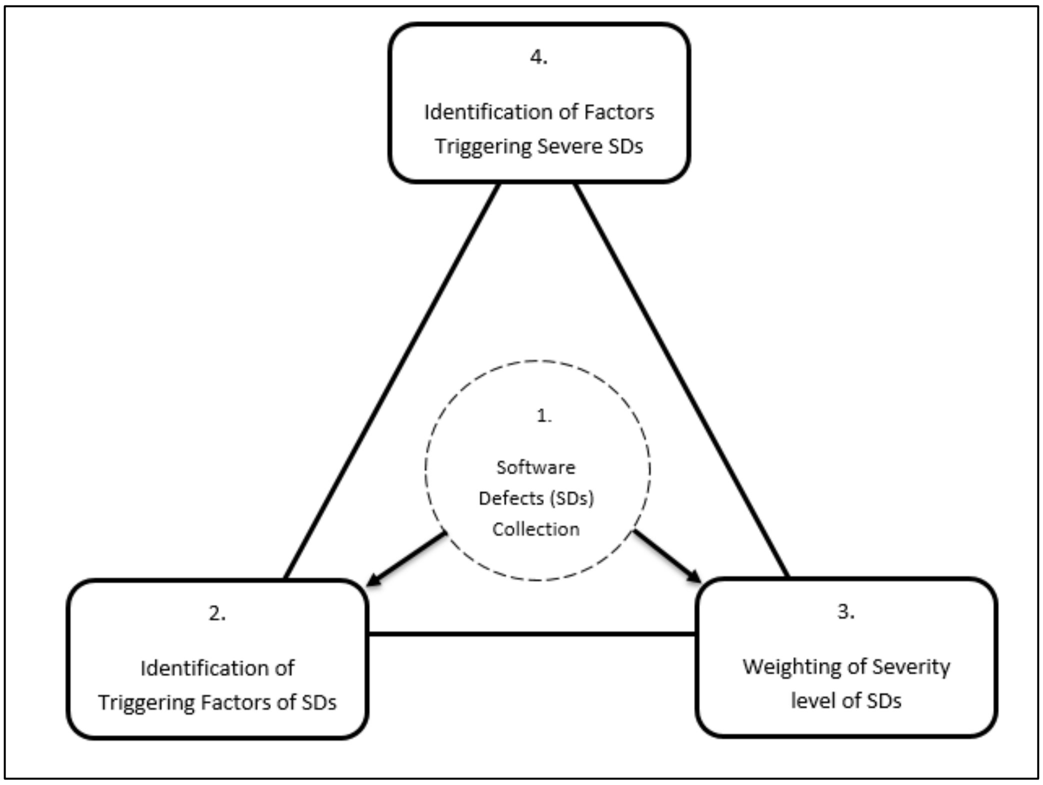

4.
Identification of Factors Triggering Severe SDs
1.
Software Defects (SDs) Collection
2.
Identification of Triggering Factors of SDs
3.
Weighting of Severity level of SDs

Fig. 4. 4-steps method to identify trigger factors causing most of severe SDs to a system

Second, to the question of which SD factors trigger most of severe SDs, we found that the following:

In the leading position are the technology trigger factors with 29% of the total weighted score for system A and 58% for system B. They are followed by the architecture trigger factors with 25% (30.8 weighted score) for system A and 24% for system B. We can see that in both cases, the same type of SD trigger factors occupies the first and the second position. In contrary, the third position is occupied by the business-IS alignment with 24% for system A while the IS/users fit occupied the same rank with 14% for system B. Finally, the business-IS alignment with 4% occupied the last place for system B, and IS/user fit with 22% occupied this position for system A.

5 Conclusion

With this case study, we have shown that severe SDs are mostly caused by technology and architecture trigger factors. We did so by classifying SDs according to four groups of trigger factors and matched them with the SDs severity level. The obtained results show us that the technology triggers are leading with 85.7 weighted scores. In the second position are the IS architecture triggers which come with 34.7 weighted scores, and are then followed by the IS/users fit triggers with 21. The last position is occupied by the business-IS alignment triggers with 6.4 weighted scores of the total SDs analyzed. As contribution, we also presented a method in order to identify problematic SDs trigger groups using their severity weight.

The results obtained from this study will help software teams to reallocate the resources of maintaining systems and help them to prioritize certain categories of SDs, thus reduce the cost of maintaining software systems.

In our future work, we will analyze other software systems and compare their results to the ones we obtained in this study. We will also investigate if there is any correlation between the first couple of trigger factor groups (IS architecture and technology) and the last couple of trigger factor groups (business-IS alignment and IS/users fit).

6 References

[1] *1044-2009 IEEE Standard Classification for Software Anomalies.* 2009.

[2] R. B. Grady, "Software Failure Analysis for High-Return Process Improvement Decisions," p. 12.

[3] M. Grottke and K. S. Trivedi, "A Classification of Software Faults," p. 3.

[4] Richard O. Duda, Peter E . Hart, and David G. Stork, "Pattern Classification." John Wiley & Sons, Inc, 2000.

[5] M. Leszak, P. Dewayne E., and D. Stoll, "Classification and evaluation of defects in a project retrospective," *The Journal of Systems and Software*, no. 61, pp. 173–187, 2002.

[6] R. Chillarege *et al.*, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Transactions on software Engineering*, vol. 18, no. 11, pp. 943–956, 1992.

[7] A. Métrailler and T. Estier, "EVOLIS Framework: A Method to Study Information Systems Evolution Records," in *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, 2014, pp. 3798–3807.

[8] J. T. Huber, “A Comparison of IBM’s Orthogonal Defect Classification to Hewlett Packard’s Defect Origins, Types, and Modes.” Hewlett Packard Company, 1999.

[9] G. Murphy and D. Cubranic, “Automatic bug triage using text categorization,” in *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering*, 2004.

[10] W. Dickinson, D. Leon, and A. Fodgurski, “Finding failures by cluster analysis of execution profiles,” in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, Toronto, Ont., Canada, 2001, pp. 339–348.

[11] G. K. Rajbahadur, S. Wang, Y. Kamei, and A. E. Hassan, “The impact of using regression models to build defect classifiers,” in *Proceedings of the 14th International Conference on Mining Software Repositories*, 2017, pp. 135–145.

[12] S. Wagner, “Defect classification and defect types revisited,” in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 39–40.

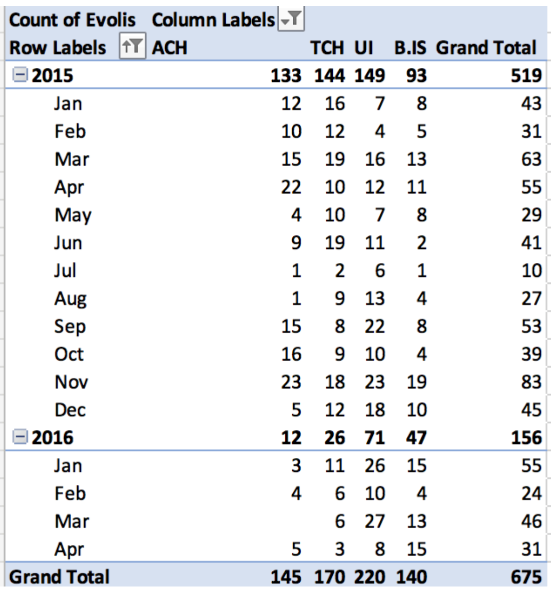
[13] E. Hossain, M. A. Babar, and H. Paik, “Using Scrum in Global Software Development: A Systematic Literature Review,” 2009, pp. 175–184.

[14] Atlassian, “Jira | Logiciel de suivi des tickets et des projets,” *Atlassian*. [Online]. Available: https://fr.atlassian.com/software/jira. [Accessed: 06-Apr-2018].

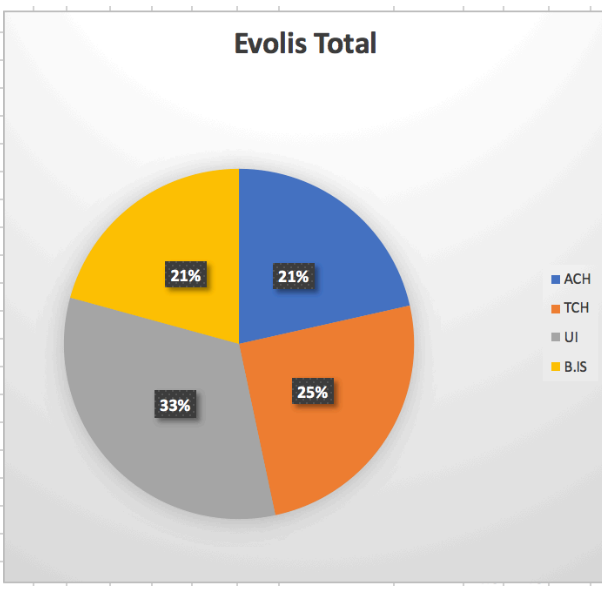
[15] N. Hillah, “Severe Software Defects Trigger Factors: A Case Study of a School Management System,” in *Digital Science*, vol. 850, T. Antipova and A. Rocha, Eds. Cham: Springer International Publishing, 2019, pp. 389–396.

Appendix 5. Data of System A

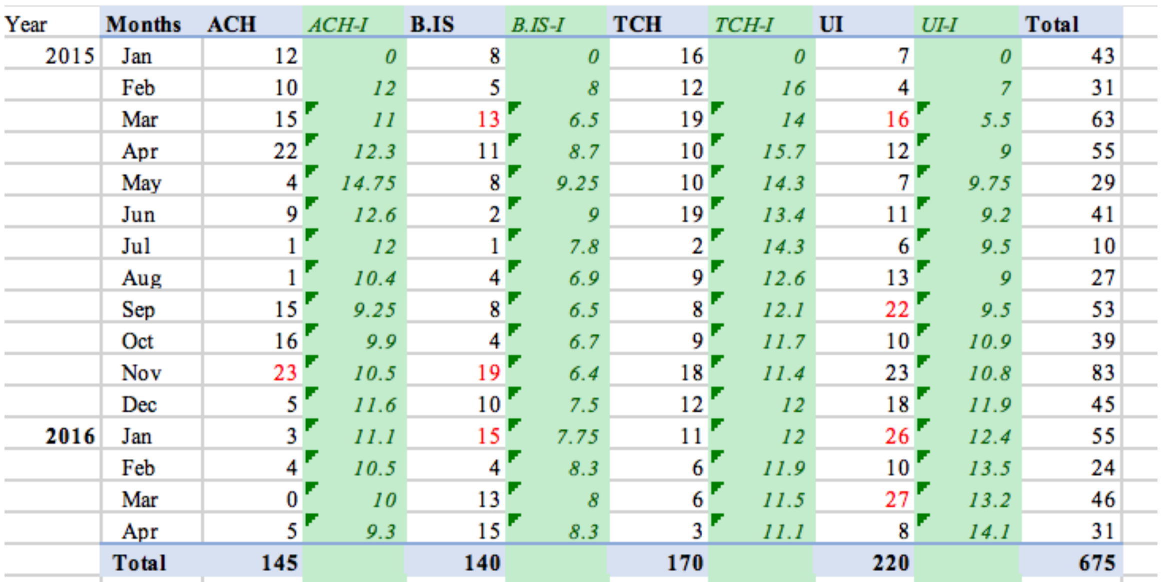
5.1 EVOLIS Classification



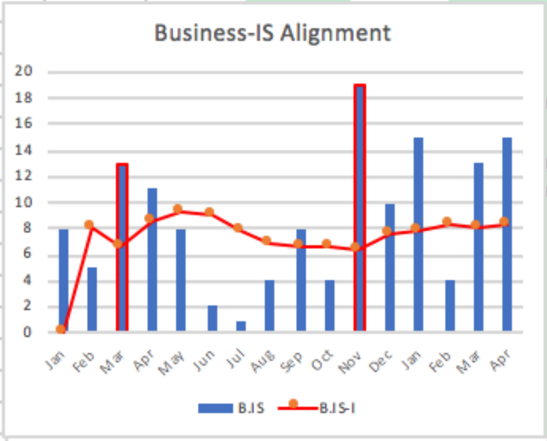
Count of Evolis	Column Labels				
Row Labels	ACH	TCH	UI	B.IS	Grand Total
2015	**133**	**144**	**149**	**93**	**519**
Jan	12	16	7	8	43
Feb	10	12	4	5	31
Mar	15	19	16	13	63
Apr	22	10	12	11	55
May	4	10	7	8	29
Jun	9	19	11	2	41
Jul	1	2	6	1	10
Aug	1	9	13	4	27
Sep	15	8	22	8	53
Oct	16	9	10	4	39
Nov	23	18	23	19	83
Dec	5	12	18	10	45
2016	**12**	**26**	**71**	**47**	**156**
Jan	3	11	26	15	55
Feb	4	6	10	4	24
Mar		6	27	13	46
Apr	5	3	8	15	31
Grand Total	**145**	**170**	**220**	**140**	**675**

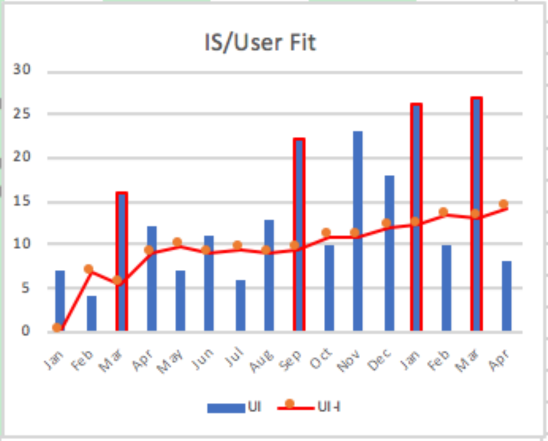


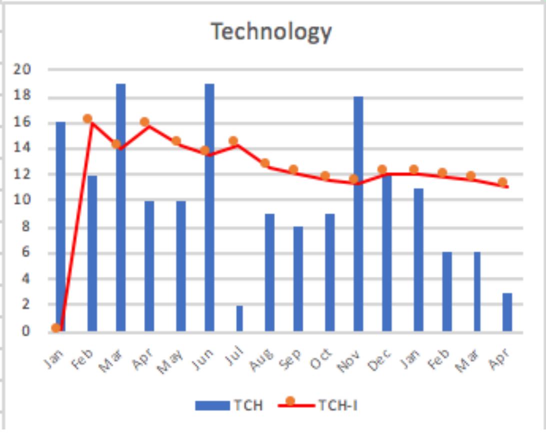
5.2 EVOLIS Indicator

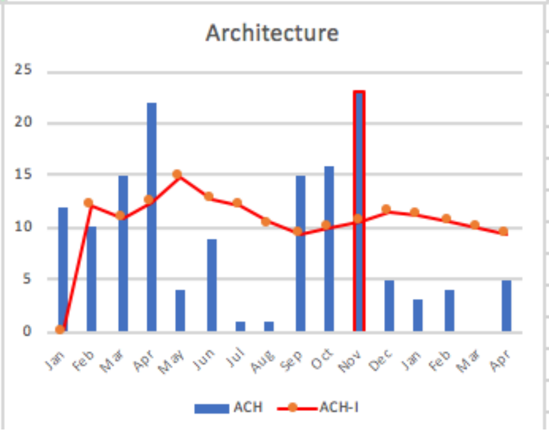


Year	Months	ACH	*ACH-I*	B.IS	*B.IS-I*	TCH	*TCH-I*	UI	*UI-I*	Total
2015	Jan	12	*0*	8	*0*	16	*0*	7	*0*	43
	Feb	10	*12*	5	*8*	12	*16*	4	*7*	31
	Mar	15	*11*	13	*6.5*	19	*14*	16	*5.5*	63
	Apr	22	*12.3*	11	*8.7*	10	*15.7*	12	*9*	55
	May	4	*14.75*	8	*9.25*	10	*14.3*	7	*9.75*	29
	Jun	9	*12.6*	2	*9*	19	*13.4*	11	*9.2*	41
	Jul	1	*12*	1	*7.8*	2	*14.3*	6	*9.5*	10
	Aug	1	*10.4*	4	*6.9*	9	*12.6*	13	*9*	27
	Sep	15	*9.25*	8	*6.5*	8	*12.1*	22	*9.5*	53
	Oct	16	*9.9*	4	*6.7*	9	*11.7*	10	*10.9*	39
	Nov	23	*10.5*	19	*6.4*	18	*11.4*	23	*10.8*	83
	Dec	5	*11.6*	10	*7.5*	12	*12*	18	*11.9*	45
2016	Jan	3	*11.1*	15	*7.75*	11	*12*	26	*12.4*	55
	Feb	4	*10.5*	4	*8.3*	6	*11.9*	10	*13.5*	24
	Mar	0	*10*	13	*8*	6	*11.5*	27	*13.2*	46
	Apr	5	*9.3*	15	*8.3*	3	*11.1*	8	*14.1*	31
	Total	**145**		**140**		**170**		**220**		**675**

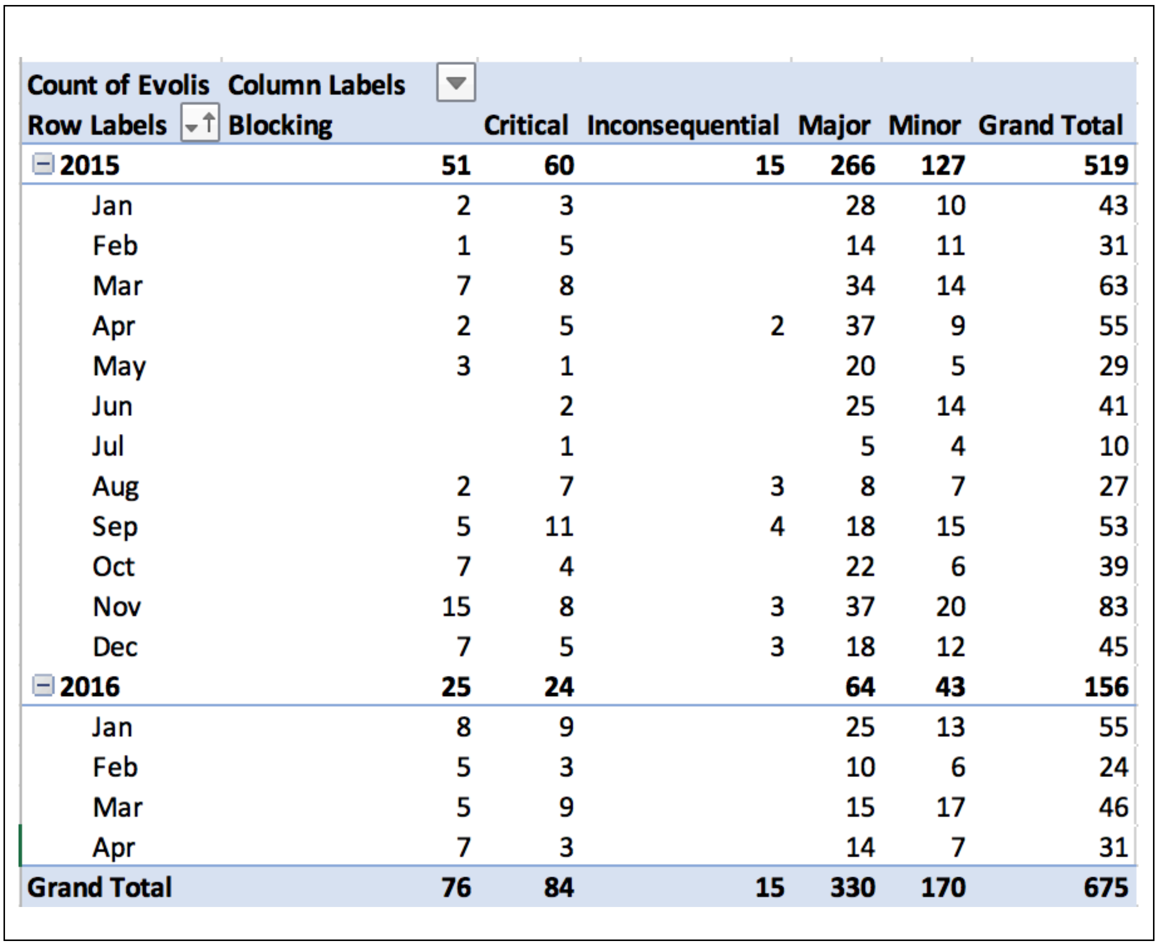








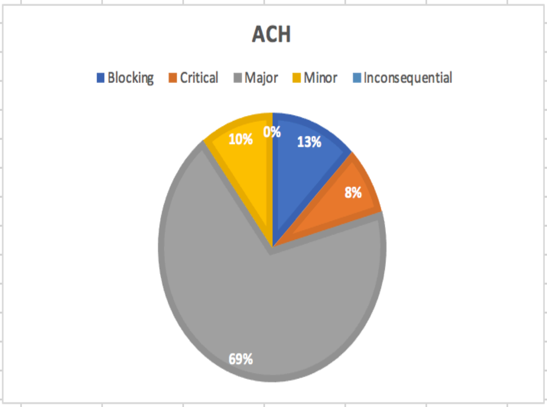
5.3 Severity Classification

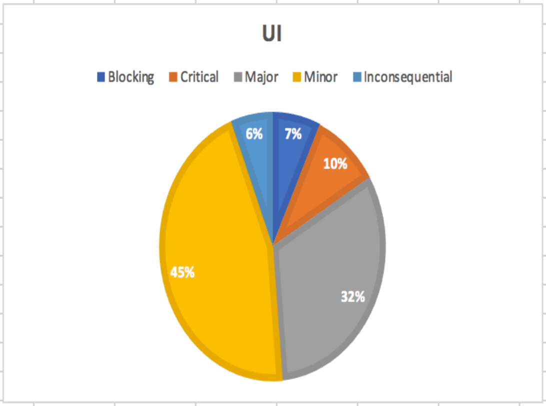


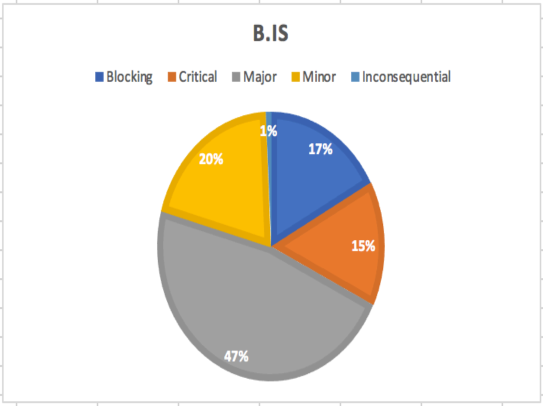
Count of Evolis	Column Labels					
Row Labels	Blocking	Critical	Inconsequential	Major	Minor	Grand Total
2015	**51**	**60**	**15**	**266**	**127**	**519**
Jan	2	3		28	10	43
Feb	1	5		14	11	31
Mar	7	8		34	14	63
Apr	2	5	2	37	9	55
May	3	1		20	5	29
Jun		2		25	14	41
Jul		1		5	4	10
Aug	2	7	3	8	7	27
Sep	5	11	4	18	15	53
Oct	7	4		22	6	39
Nov	15	8	3	37	20	83
Dec	7	5	3	18	12	45
2016	**25**	**24**		**64**	**43**	**156**
Jan	8	9		25	13	55
Feb	5	3		10	6	24
Mar	5	9		15	17	46
Apr	7	3		14	7	31
Grand Total	**76**	**84**	**15**	**330**	**170**	**675**

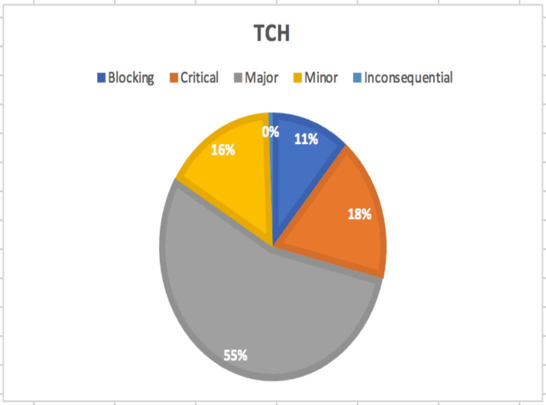
5.4 EVOLIS & Severity Classifications

	Blocking	Critical	Major	Minor	Inconsequential
ACH	18	12	100	15	0
TCH	19	30	93	27	1
UI	15	21	71	100	13
B.IS	24	21	66	28	1
Total	**76**	**84**	**330**	**170**	**15**



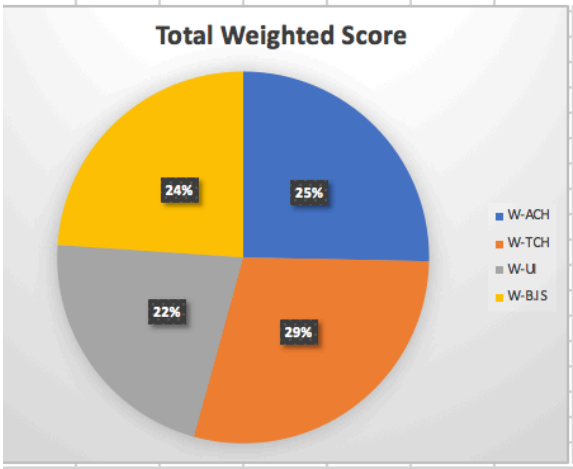


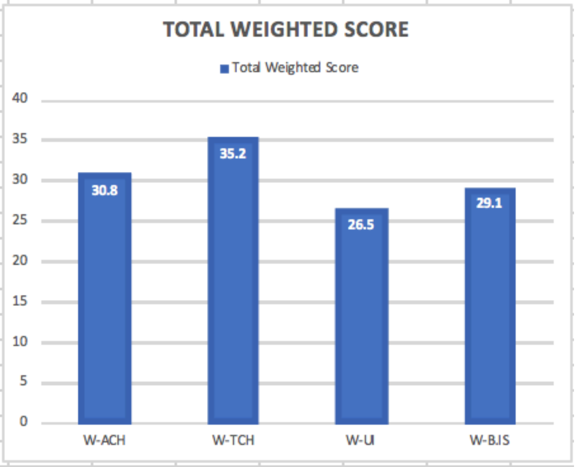


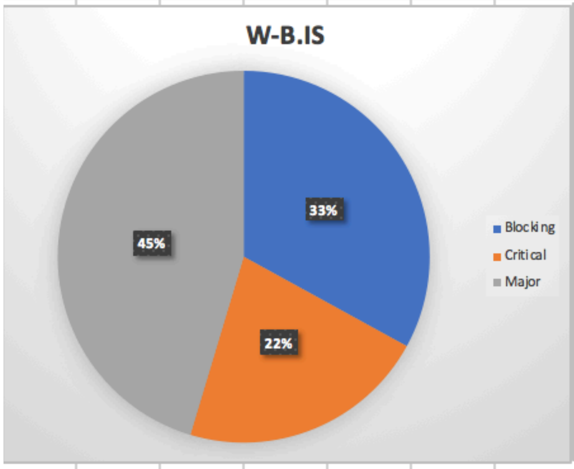


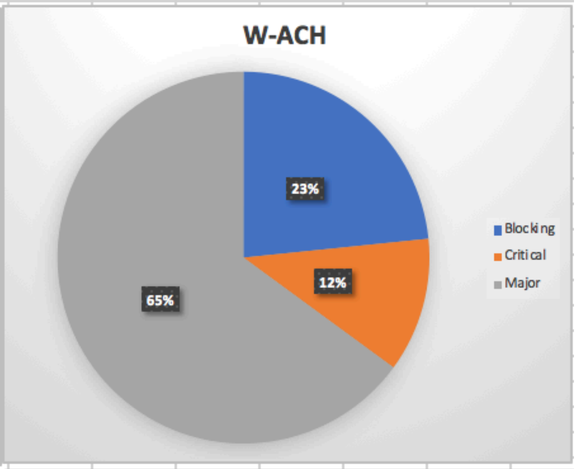
5.5 EVOLIS & Severity Weighted

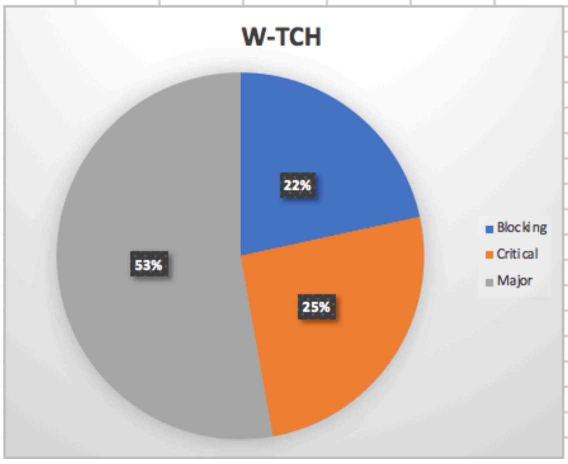
	Weight	ACH	*W-ACH*	TCH	*W-TCH*	UI	*W-UI*	B.IS	*W-B.IS*
Blocking	0.4	18	*7.2*	19	*7.6*	15	*6*	24	*9.6*
Critical	0.3	12	*3.6*	30	*9*	21	*6.3*	21	*6.3*
Major	0.2	100	*20*	93	*18.6*	71	*14.2*	66	*13.2*
Total We	0.9	130	*30.8*	142	*35.2*	107	*26.5*	111	*29.1*

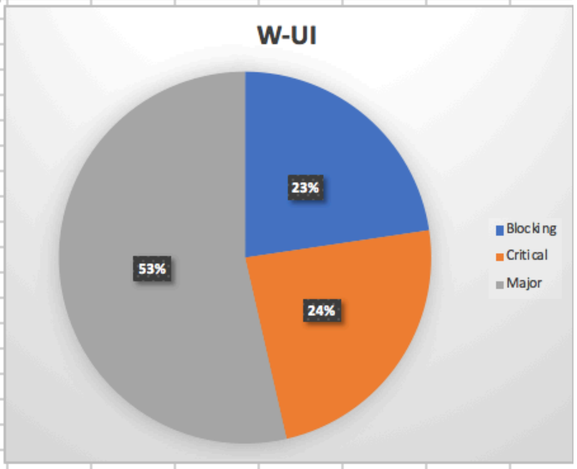




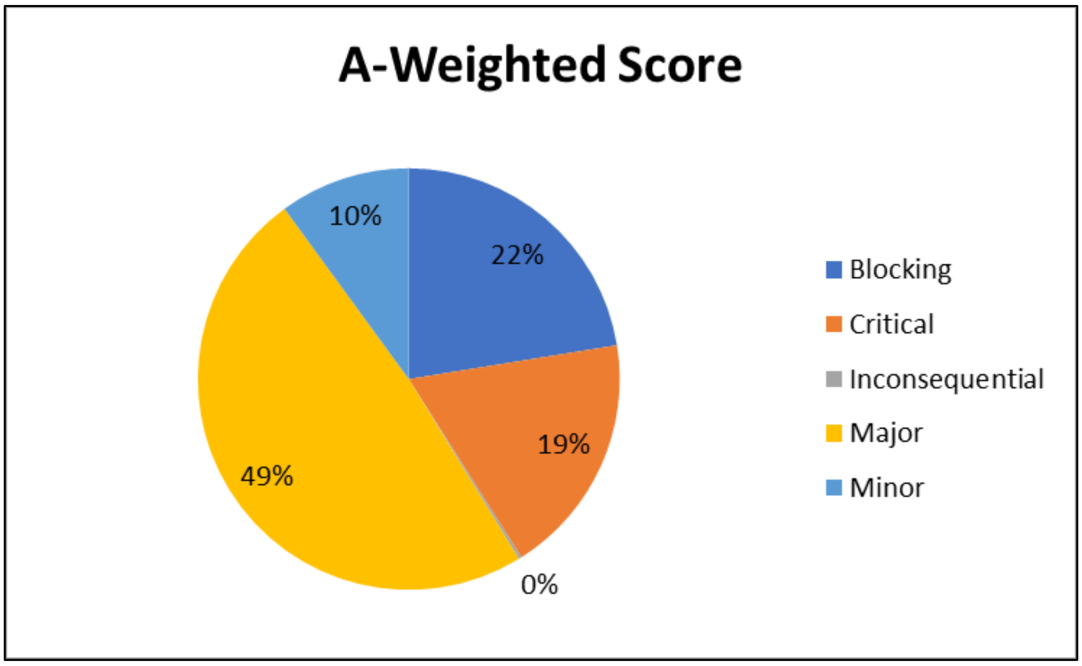






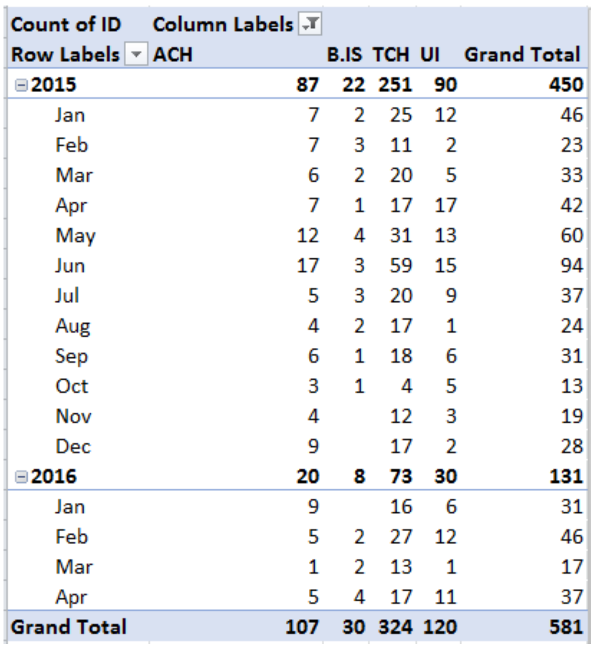


5.6 System A Weighted Score

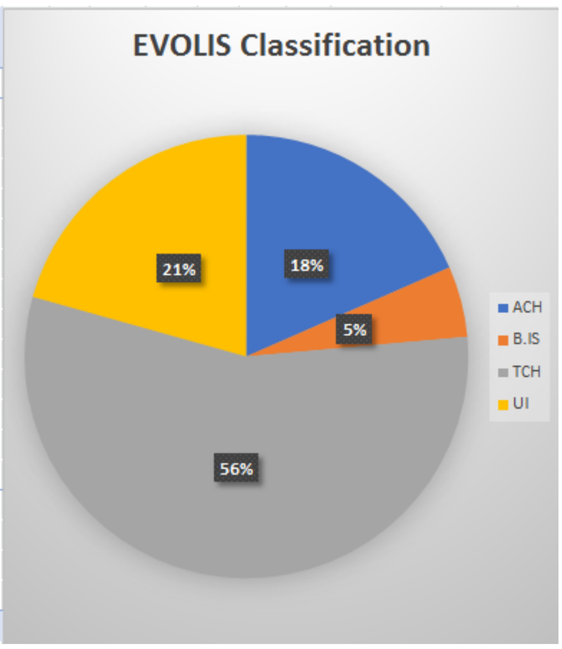


Appendix 6. Data of System B

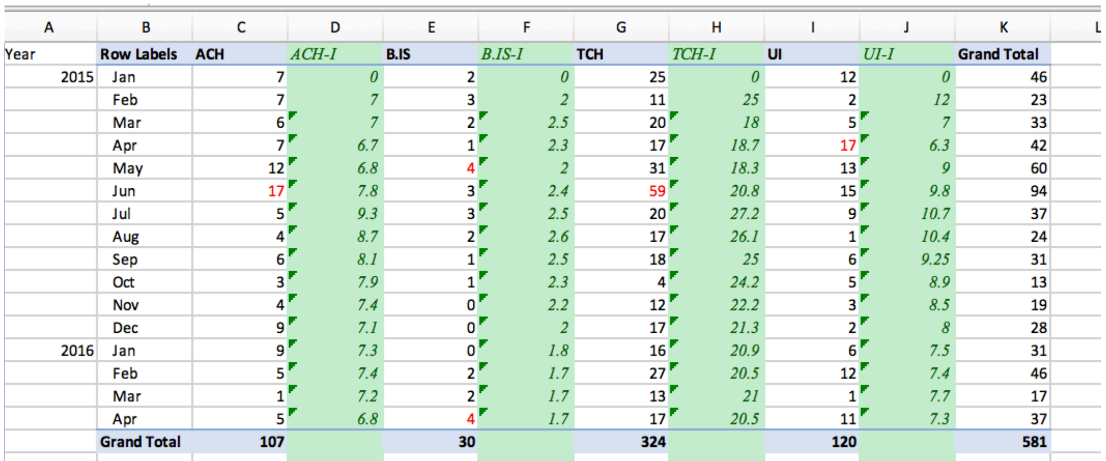
6.1 EVOLIS Classification



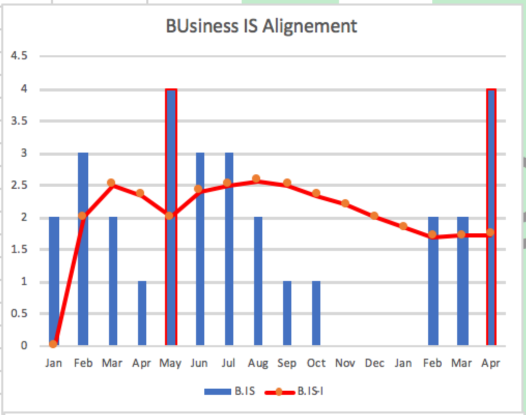
Count of ID	Column Labels				
Row Labels	ACH	B.IS	TCH	UI	Grand Total
2015	**87**	**22**	**251**	**90**	**450**
Jan	7	2	25	12	46
Feb	7	3	11	2	23
Mar	6	2	20	5	33
Apr	7	1	17	17	42
May	12	4	31	13	60
Jun	17	3	59	15	94
Jul	5	3	20	9	37
Aug	4	2	17	1	24
Sep	6	1	18	6	31
Oct	3	1	4	5	13
Nov	4		12	3	19
Dec	9		17	2	28
2016	**20**	**8**	**73**	**30**	**131**
Jan	9		16	6	31
Feb	5	2	27	12	46
Mar	1	2	13	1	17
Apr	5	4	17	11	37
Grand Total	**107**	**30**	**324**	**120**	**581**

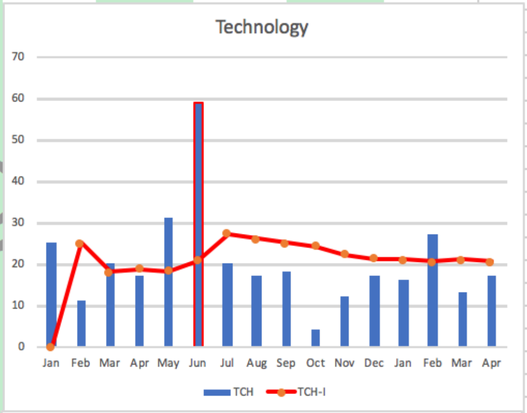


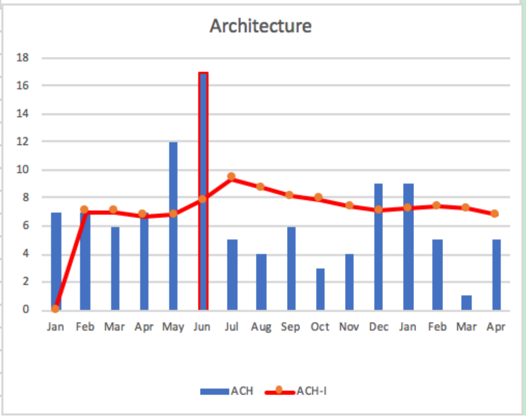
6.2 EVOLIS Indicator

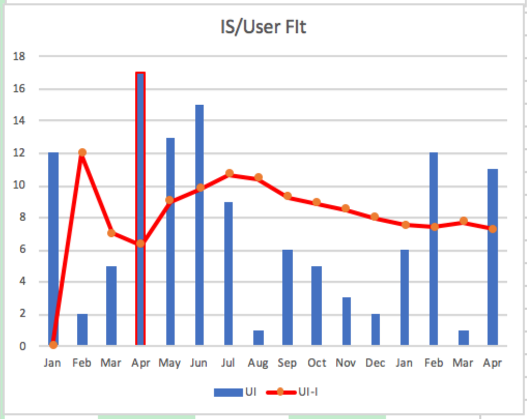


A	B	C	D	E	F	G	H	I	J	K
Year	**Row Labels**	**ACH**	*ACH-I*	**B.IS**	*B.IS-I*	**TCH**	*TCH-I*	**UI**	*UI-I*	**Grand Total**
2015	Jan	7	*0*	2	*0*	25	*0*	12	*0*	46
	Feb	7	*7*	3	*2*	11	*25*	2	*12*	23
	Mar	6	*7*	2	*2.5*	20	*18*	5	*7*	33
	Apr	7	*6.7*	1	*2.3*	17	*18.7*	17	*6.3*	42
	May	12	*6.8*	4	*2*	31	*18.3*	13	*9*	60
	Jun	17	*7.8*	3	*2.4*	59	*20.8*	15	*9.8*	94
	Jul	5	*9.3*	3	*2.5*	20	*27.2*	9	*10.7*	37
	Aug	4	*8.7*	2	*2.6*	17	*26.1*	1	*10.4*	24
	Sep	6	*8.1*	1	*2.5*	18	*25*	6	*9.25*	31
	Oct	3	*7.9*	1	*2.3*	4	*24.2*	5	*8.9*	13
	Nov	4	*7.4*	0	*2.2*	12	*22.2*	3	*8.5*	19
	Dec	9	*7.1*	0	*2*	17	*21.3*	2	*8*	28
2016	Jan	9	*7.3*	0	*1.8*	16	*20.9*	6	*7.5*	31
	Feb	5	*7.4*	2	*1.7*	27	*20.5*	12	*7.4*	46
	Mar	1	*7.2*	2	*1.7*	13	*21*	1	*7.7*	17
	Apr	5	*6.8*	4	*1.7*	17	*20.5*	11	*7.3*	37
	Grand Total	**107**		**30**		**324**		**120**		**581**

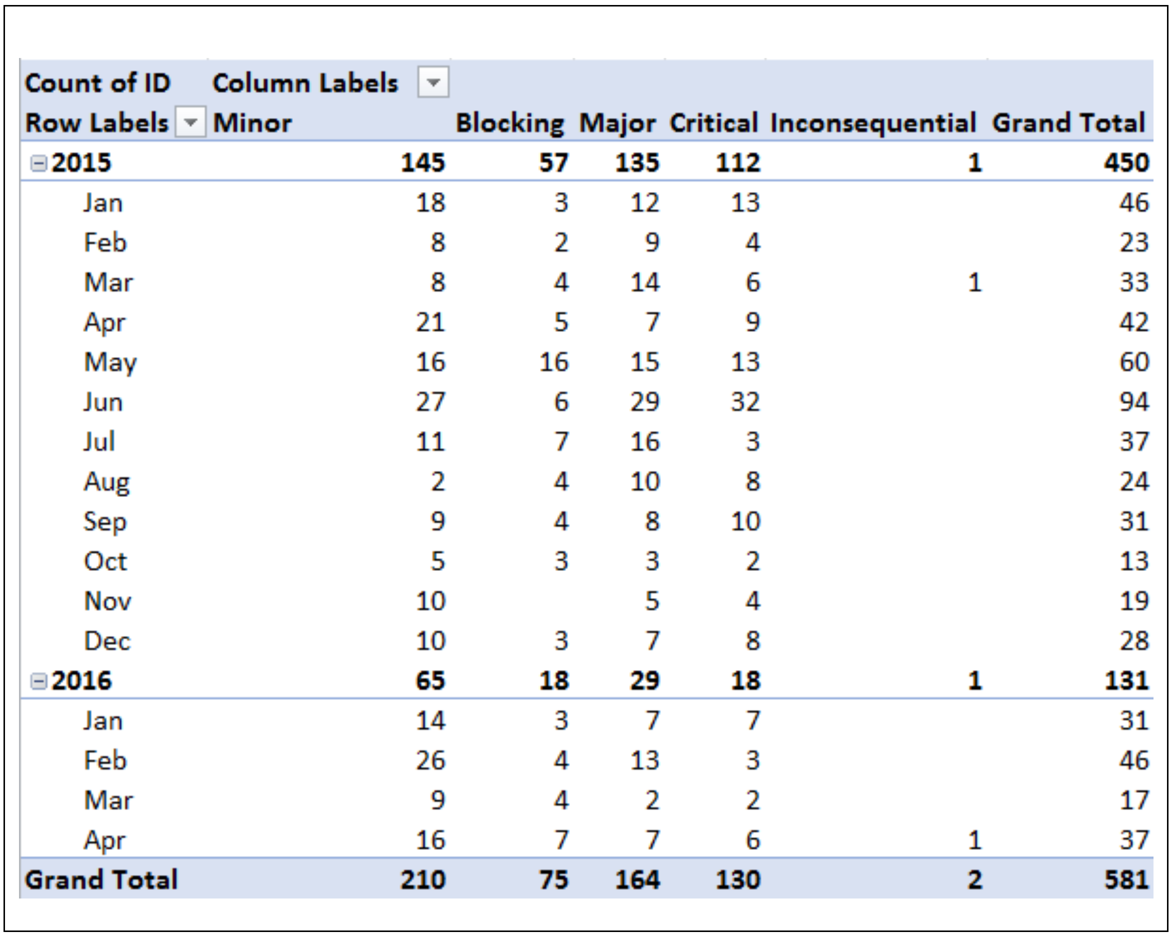








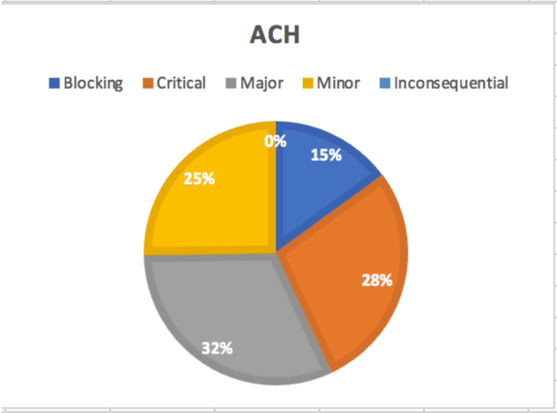
6.3 Severity Classification

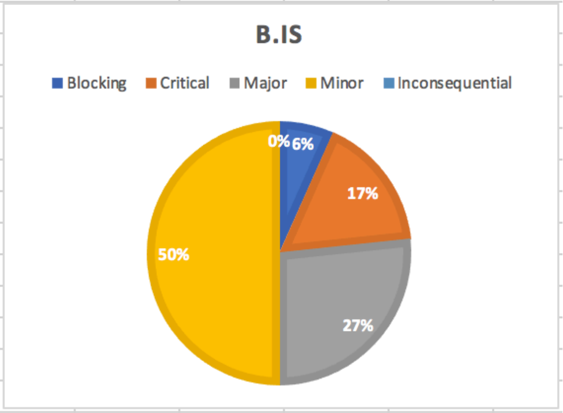


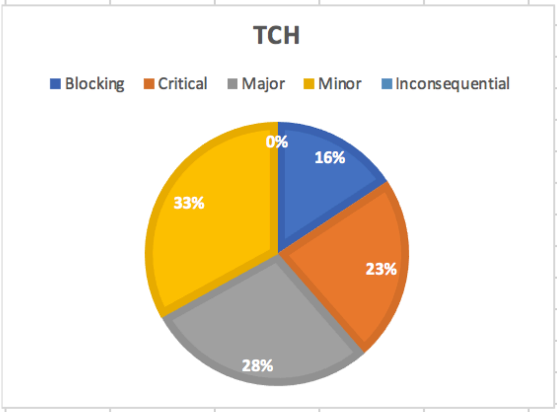
Count of ID	Column Labels					
Row Labels	Minor	Blocking	Major	Critical	Inconsequential	Grand Total
2015	**145**	**57**	**135**	**112**	**1**	**450**
Jan	18	3	12	13		46
Feb	8	2	9	4		23
Mar	8	4	14	6	1	33
Apr	21	5	7	9		42
May	16	16	15	13		60
Jun	27	6	29	32		94
Jul	11	7	16	3		37
Aug	2	4	10	8		24
Sep	9	4	8	10		31
Oct	5	3	3	2		13
Nov	10		5	4		19
Dec	10	3	7	8		28
2016	**65**	**18**	**29**	**18**	**1**	**131**
Jan	14	3	7	7		31
Feb	26	4	13	3		46
Mar	9	4	2	2		17
Apr	16	7	7	6	1	37
Grand Total	**210**	**75**	**164**	**130**	**2**	**581**

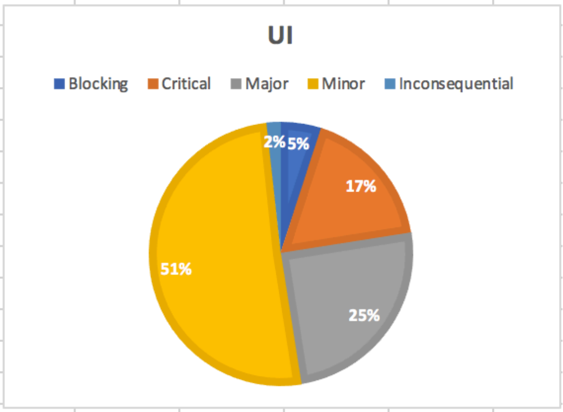
6.4 Severity & EVOLIS Classifications

	Blocking	Critical	Major	Minor	Inconsequential
ACH	16	30	34	27	0
B.IS	2	5	8	15	0
TCH	51	74	92	107	0
UI	6	21	30	61	2
Total	75	130	164	210	2



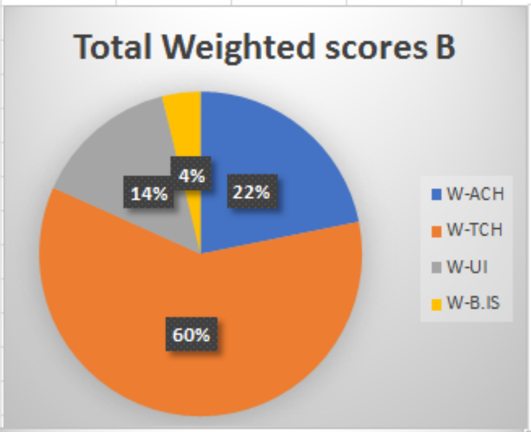


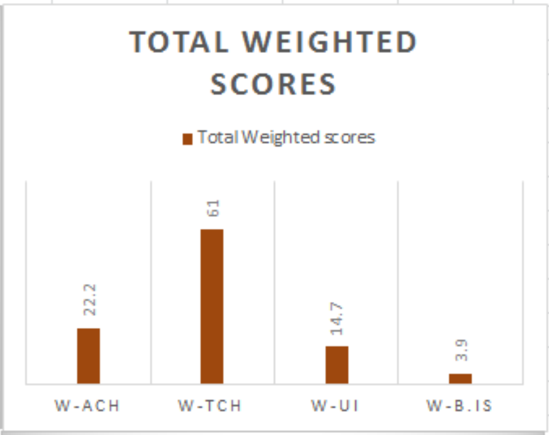


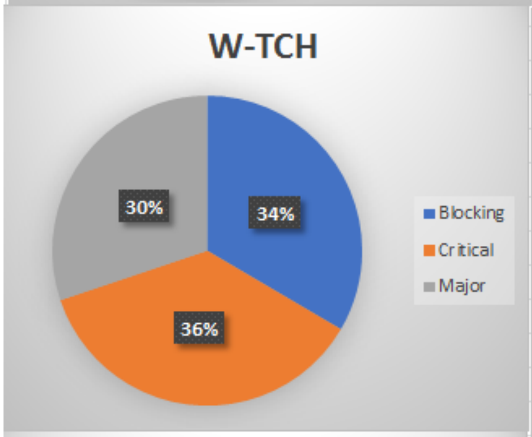


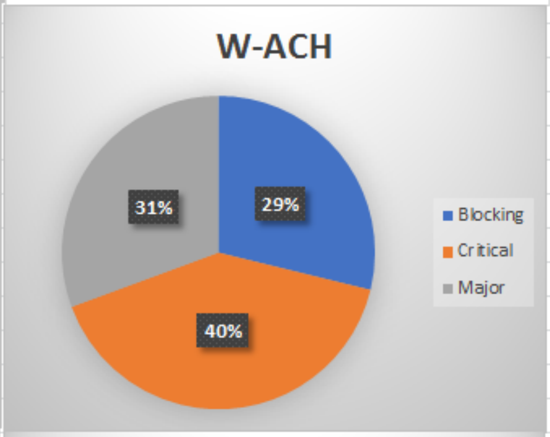
6.5 Severity & EVOLIS Weighted

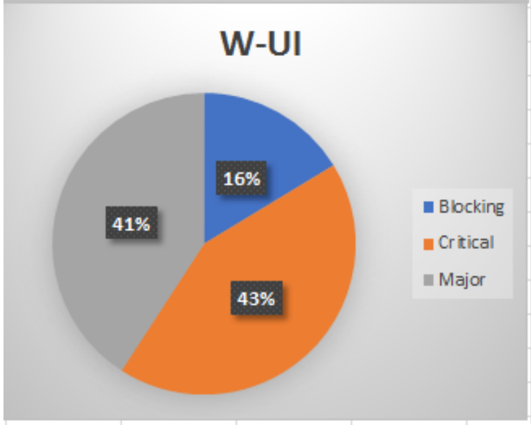
	Weight	ACH	*W-ACH*	TCH	*W-TCH*	UI	*W-UI*	B.IS	*W-B.IS*
Blocking	0.4	16	*6.4*	51	*20.4*	6	*2.4*	2	*0.8*
Critical	0.3	30	*9*	74	*22.2*	21	*6.3*	5	*1.5*
Major	0.2	34	*6.8*	92	*18.4*	30	*6*	8	*1.6*
Total Weig	0.9	80	*22.2*	217	*61*	57	*14.7*	15	*3.9*

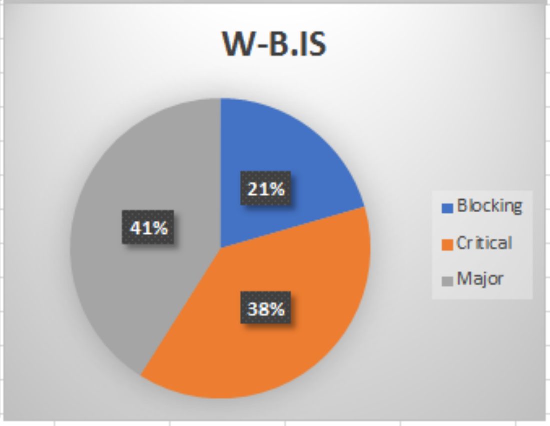




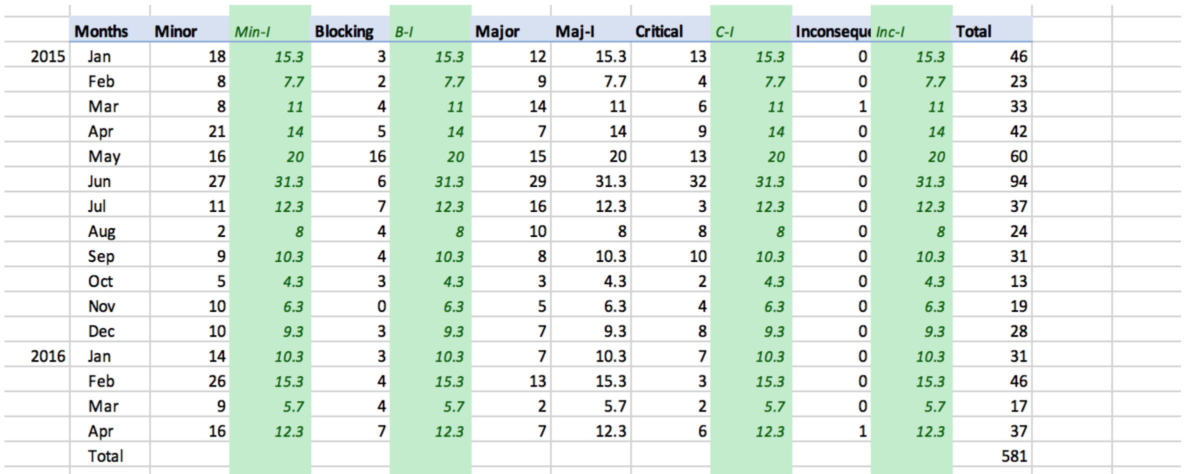




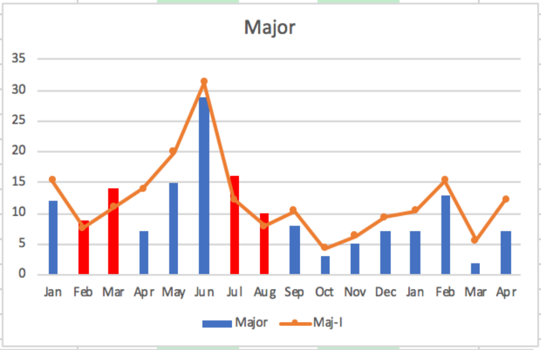


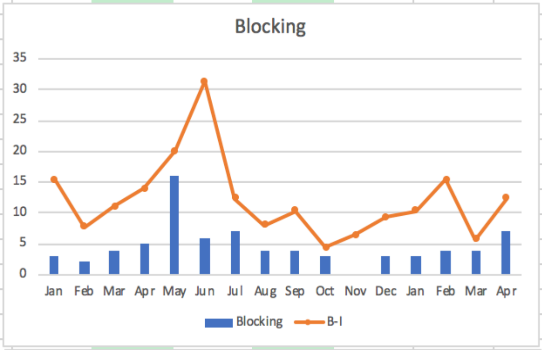


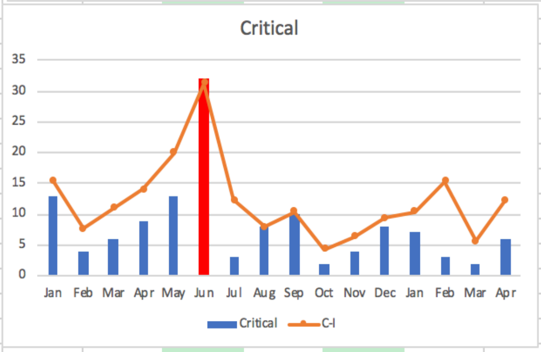
6.6 Severity Indicator

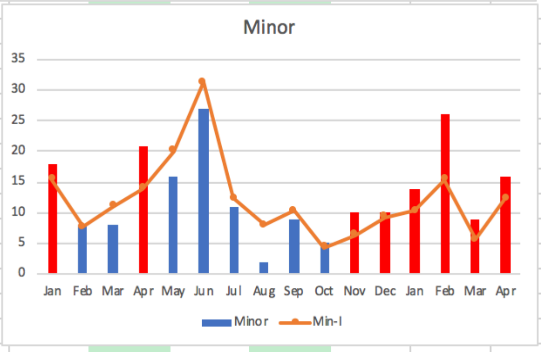


	Months	Minor	Min-I	Blocking	B-I	Major	Maj-I	Critical	C-I	Inconsequ	Inc-I	Total
2015	Jan	18	15.3	3	15.3	12	15.3	13	15.3	0	15.3	46
	Feb	8	7.7	2	7.7	9	7.7	4	7.7	0	7.7	23
	Mar	8	11	4	11	14	11	6	11	1	11	33
	Apr	21	14	5	14	7	14	9	14	0	14	42
	May	16	20	16	20	15	20	13	20	0	20	60
	Jun	27	31.3	6	31.3	29	31.3	32	31.3	0	31.3	94
	Jul	11	12.3	7	12.3	16	12.3	3	12.3	0	12.3	37
	Aug	2	8	4	8	10	8	8	8	0	8	24
	Sep	9	10.3	4	10.3	8	10.3	10	10.3	0	10.3	31
	Oct	5	4.3	3	4.3	3	4.3	2	4.3	0	4.3	13
	Nov	10	6.3	0	6.3	5	6.3	4	6.3	0	6.3	19
	Dec	10	9.3	3	9.3	7	9.3	8	9.3	0	9.3	28
2016	Jan	14	10.3	3	10.3	7	10.3	7	10.3	0	10.3	31
	Feb	26	15.3	4	15.3	13	15.3	3	15.3	0	15.3	46
	Mar	9	5.7	4	5.7	2	5.7	2	5.7	0	5.7	17
	Apr	16	12.3	7	12.3	7	12.3	6	12.3	1	12.3	37
	Total											581

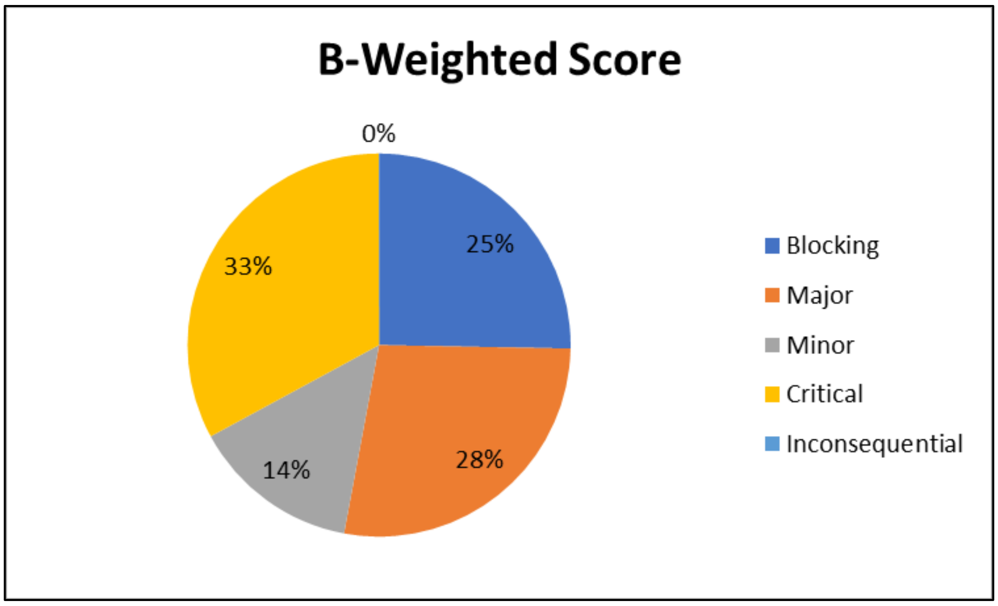








6.7 System B Weighted Score



Appendix 7. Raw Data - Examples of a Software Defect

- **Example I**

Projet	ABX
Clé	A-1[redacted]
Résumé	org.apache.jasper.el.JspELException: /WEB-INF/u[redacted]eTS.jsp(12,1)
Type de demande	Correction
Etat	Fermée
Severité	Majeur
Résolution	Validée
Attribution	Non attribuée
Rapporteur	XXXXXXX
Création	05.01.15
Dernier affichage	
Mise à jour	15.01.15
Résolue	15.01.15
Affecte la/les version(s)	A[redacted]

Version(s) corrigée(s)	[redacted] 0
Composants	A
Date d'échéance	
Gérer les observateurs	1
Images	
Estimation originale	
Estimation restante	
Temps consacré	
Ratio du travail réel comparé à l'estimation	
Sous-tâches	
Demandes liées	
Environnement	
Descriptif	Hello, On a une forte occurrence de l'erreur suivante en production. En fait cette erreur apparaît [redacted] er' [redacted] aucune note pou [redacted] TS. Il serait préférable d'avoir le même comportement qui existe sur la page de saisie des [redacted] Serait il possible de ce corriger SVP? XX --- Stack-trace org.apache.jasper.JasperException: org.apache.jasper.el.JspELException: /WEB-INF/ui/saisieNoteTS.jsp(12,1) '${A:isSaisieNotesReadOnly(evaluationsTravail.travailEvalue, isBlocageEnCours, auteurs)}' Problems calling function 'A:isSaisieNotesReadOnly'

	at org.apache.jasper.servlet.JspServletWrapper.handleJspException(JspServletWrapper.java:549) at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:470) at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:390) ... 114 more
Niveau de sécurité	
Progression	
Σ Progrès	
Σ Temps consacré	
Σ Estimation restante	
Σ Estimation originale	
Étiquettes	P1
Épopée/thème	
Sprint	
Logbook	
Date d'Annonce	05/janv./15 3:21 PM
Lien d'épopée	
Effort estimé	
Origine de la demande	AAA-BBB
Version(s) vérifiée(s)	[redacted]
Validation	

Version planifiée	
Impact Migration	
Complexité	
QcBugId	
QC Synchronisation	N
Aléas	
indicateur	
Environnement -liste	[redacted]ion
Estimation Spécifications	
Non-Qualité	
Classement	[redacted]n:
Estimation Tests	
Estimation Développement	
Résolue	
Date de livraison	15/janv./15 9:45 AM
Date début de traitement	07/janv./15 3:59 PM

- **Example II**

Projet	ABX
Clé	A[redacted]
Résumé	Totaux des points de groupe avec discipline famille
Type de demande	Correction
Etat	Livrée
Sévérité	Bloquant
Résolution	Déployée
Attribution	XXXX
Rapporteur	XXXXXXX
Création	15.04.2016
Dernier affichage	20.04.2016
Mise à jour	19.04.2016
Résolue	19.04.2016
Affecte la/les version(s)	A[redacted]1.1
Version(s) corrigée(s)	AB[redacted].2
Composants	ABX
Date d'échéance	
Gérer les observateurs	1
Images	
Estimation originale	

Estimation restante	
Temps consacré	
Ratio du travail réel comparé à l'estimation	
Sous-tâches	
Demandes liées	
Environnement	
Descriptif	Le calcul des totaux des [redacted] le est faux. C'est la moyenne finale de la discipline famille qui doit être prise en compte et non la moyenne finale de la discipline mère.
Niveau de sécurité	
Progression	
Σ Progrès	
Σ Temps consacré	
Σ Estimation restante	
Σ Estimation originale	
Étiquettes	
Épopée/thème	
Sprint	
Logbook	
Date d'Annonce	18/avr./16 5:59 PM

Lien d'épopée	
Effort estimé	
Origine de la demande	X[redacted]st
Version(s) vérifiée(s)	
Validation	
Version planifiée	
Impact Migration	
Complexité	
QcBugId	
QC Synchronisation	N
Aléas	15
indicateur	
Environnement-liste	[redacted]on
Estimation Spécifications	
Non-Qualité	15
Classement	[redacted]
Estimation Tests	
Estimation Développement	

Résolue	
Date de livraison	19/avr./16 7:33 AM
Date début de traitement	18/avr./16 5:59 PM